

# 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
- ✓ Example, which solution is best?
  - $F = BD + \overline{A}\overline{B}C + A\overline{C}\overline{D}$   $L = 8$
  - $F = BD + \overline{A}\overline{B}C + \overline{A}B\overline{D} + ABC\overline{D}$   $L = 11$
  - $F = (A + B)(A + D)(B + C + \overline{D})(\overline{B} + \overline{C} + D)$   $L = 10$

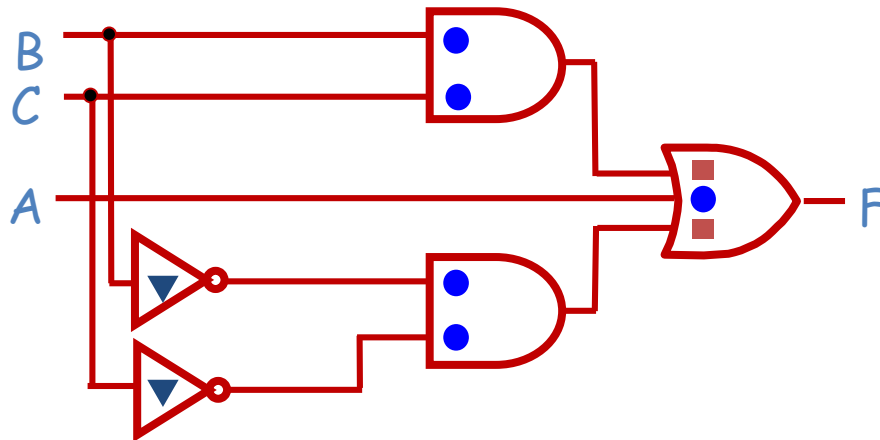
# Gate Input Cost

---

- ✓ 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 found 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, which solution is best?
  - $F = BD + A\bar{B}C + A\bar{C}\bar{D}$   $G = 11, GN = 14$
  - $F = BD + A\bar{B}C + A\bar{B}\bar{D} + ABC\bar{C}$   $G = 15, GN = 18$
  - $F = (A + \bar{B})(A + D)(B + C + \bar{D})(\bar{B} + \bar{C} + D)$   $G = 14, GN = 17$

# Cost Criteria (continued)

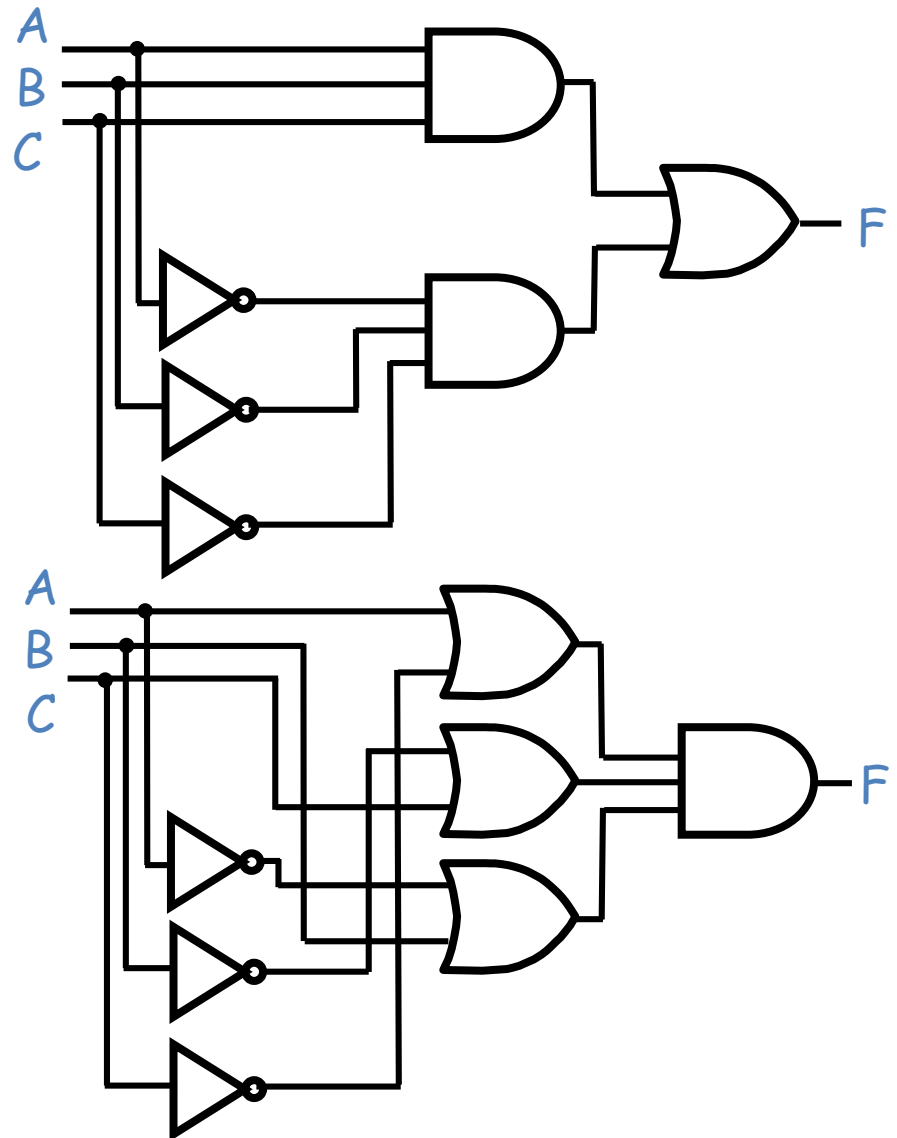
✓ Example 1:  $\nabla \nabla GN = G + 2 = 9$   
 ✓  $F = \overset{\bullet}{A} + \overset{\bullet}{B} \overset{\bullet}{C} + \overset{\bullet}{\overline{B}} \overset{\bullet}{\overline{C}}$   $L = 5$   
 $G = L + 2 = 7$



- 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)

- $F = A B C + \bar{A} \bar{B} \bar{C}$
- $L = 6 \quad G = 8 \quad GN = 11$
  
- $F = (A + \bar{C})(\bar{B} + C)(\bar{A} + B)$
- $L = 6 \quad G = 9 \quad GN = 12$
- ✓ Same function and same literal cost
- ✓ But first circuit has better gate input count and better gate input count with NOTs



# 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 (corresponding to cubes)
- ✓ The K-map can be viewed as
  - **A reorganized version of the truth table or a particular cube representation**

# Some Uses of K-Maps

---

## ✓ Provide a means for:

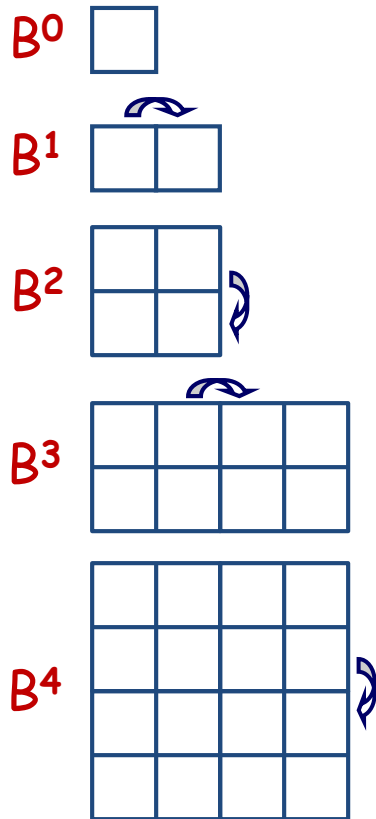
- Finding **optimum**
  - SOP and POS standard forms, and
  - two-level AND/OR and OR/AND circuit implementationsfor **functions with small numbers of variables**
- Visualizing concepts related to manipulating Boolean expressions
- Demonstrating concepts used by computer-aided design programs to simplify large circuits



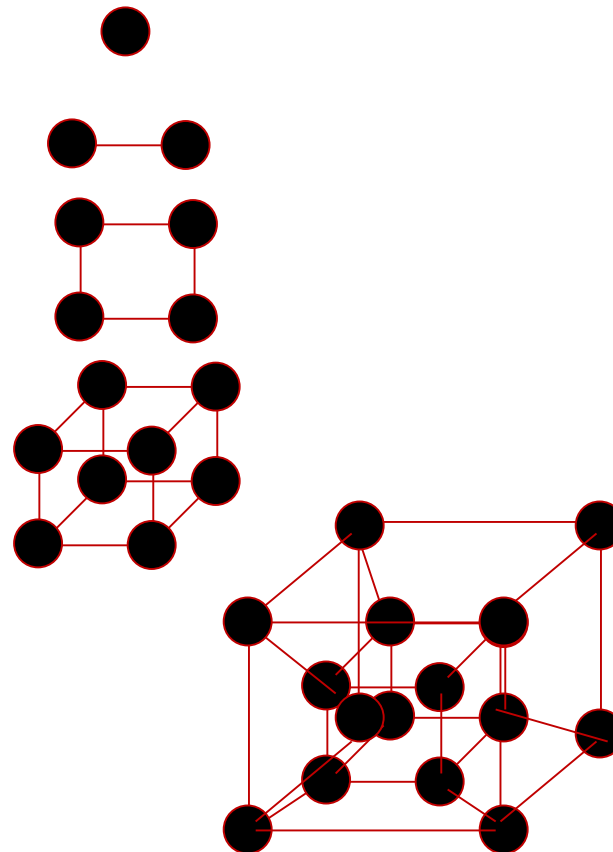
# The Boolean Space $B^n$

- ✓  $B = \{0, 1\}$
- ✓  $B^2 = \{0, 1\} \times \{0, 1\} = \{00, 01, 10, 11\}$

Karnaugh Maps:



Boolean Cubes:



# Two Variable Maps

---

## ✓ A 2-variable Karnaugh Map:

- Note that minterm  $m_0$  and minterm  $m_1$  are "adjacent" and differ in the value of the variable  $y$
- Similarly, minterm  $m_0$  and minterm  $m_2$  differ in the  $x$  variable.
- Also,  $m_1$  and  $m_3$  differ in the  $x$  variable as well.
- Finally,  $m_2$  and  $m_3$  differ in the value of the variable  $y$

$x \backslash y$	$y = 0$	$y = 1$
$x = 0$	$m_0$ $\bar{x}\bar{y}$	$m_1$ $\bar{x}y$
$x = 1$	$m_2$ $x\bar{y}$	$m_3$ $xy$

# 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

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

# K-Map Function Representation

---

- ✓ 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:

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

# K-Map Function Representation

---

✓ Example:  $G(x,y) = \bar{x}y + x\bar{y} + xy$

$G=x+y$	$y = 0$	$y = 1$
$x = 0$	0	1
$x = 1$	1	1

✓ For  $G(x,y)$ , two pairs of adjacent cells containing 1's can be combined using the Minimization Theorem:

$$G(x,y) = (x\bar{y} + xy) + (\bar{x}y + xy) = x + y$$

Duplicate  $xy$

# Three Variable Maps

- ✓ A three-variable K-map:

$x \backslash yz$	$yz=00$	$yz=01$	$yz=11$	$yz=10$
$x=0$	$m_0$	$m_1$	$m_3$	$m_2$
$x=1$	$m_4$	$m_5$	$m_7$	$m_6$

- ✓ Where each minterm corresponds to the product terms:

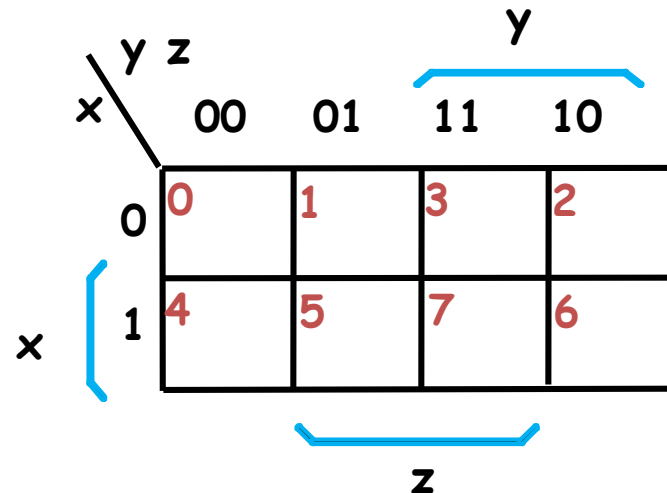
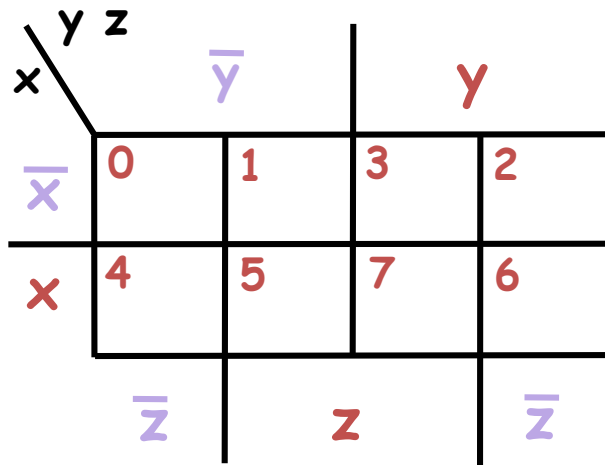
$x \backslash yz$	$yz=00$	$yz=01$	$yz=11$	$yz=10$
$x=0$	$\bar{x} \bar{y} \bar{z}$	$\bar{x} \bar{y} z$	$\bar{x} y z$	$\bar{x} y \bar{z}$
$x=1$	$x \bar{y} \bar{z}$	$x \bar{y} z$	$x y z$	$x y \bar{z}$

- ✓ Note that if the binary value for an **index differs in one bit position**, the minterms are adjacent on the K-Map

# Alternative Map Labeling

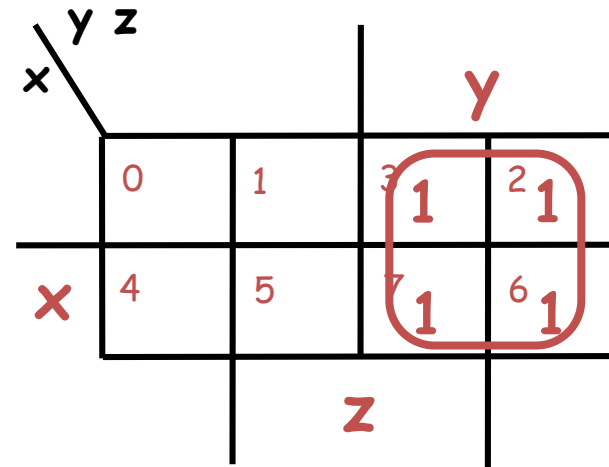
---

- ✓ Map use largely involves:
  - Entering values into the map, and
  - Reading off product terms from the map.
- ✓ Alternate labelings are useful:



# Example: Combining Squares

- ✓ Example: Let
- ✓  $F(x, y, z) = \sum_m (2, 3, 6, 7)$



- ✓ Applying the Minimization Theorem three times:

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

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



# Combining Squares

---

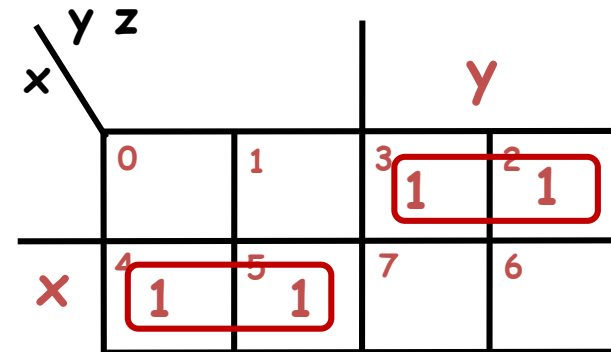
- ✓ By combining squares, we reduce number of literals in a product term, reducing the literal cost, thereby reducing the other two cost criteria
- ✓ On a 3-variable K-Map:
  - One square represents a minterm with three variables
  - Two adjacent squares represent a cube that is product term with two variables
  - Four "adjacent" terms represent a cube that is product term with one variable
  - Eight "adjacent" terms is the function of all ones (no variables) is a tautology  $f^1=1$ .

# Example Functions

- ✓ By convention, we represent the minterms of  $F$  by a "1" in the map and leave the minterms of blank  $\bar{F}$

- ✓ Example:

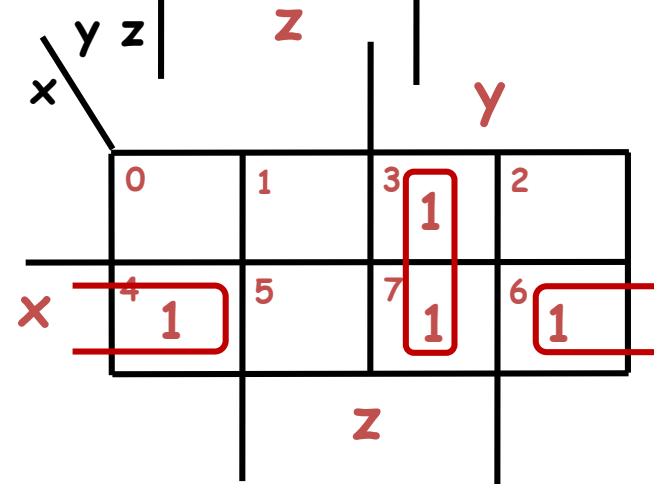
$$F(x, y, z) = \sum_m (2, 3, 4, 5)$$



- ✓ Example:

- ✓  $F(x, y, z) = \sum_m (3, 4, 6, 7)$

- ✓ Learn the locations of the 8 indices based on the **variable order** shown ( $x$ , most significant and  $z$ , least significant) on the map boundaries



# Three-Variable Maps

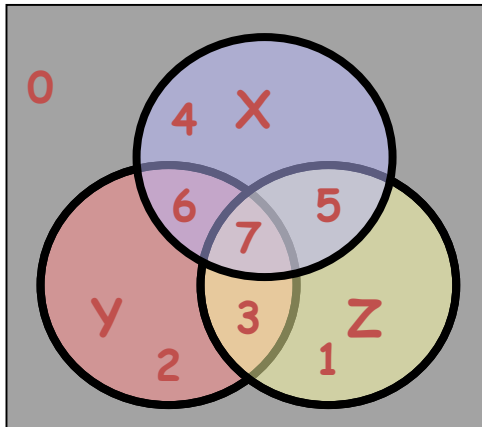
---

- ✓ Reduced literal product terms for SOP standard forms correspond to cubes i.e. to **rectangles** on the K-maps containing cell counts that are powers of 2.
- ✓ Rectangles of 2 cells represent 2 adjacent minterms; of 4 cells represent 4 minterms that form a "pairwise adjacent" ring.

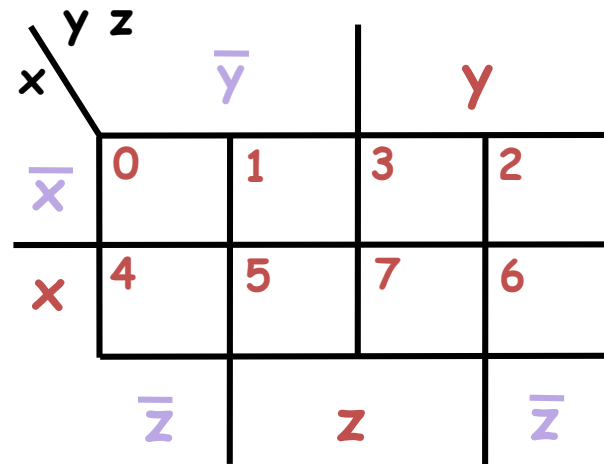
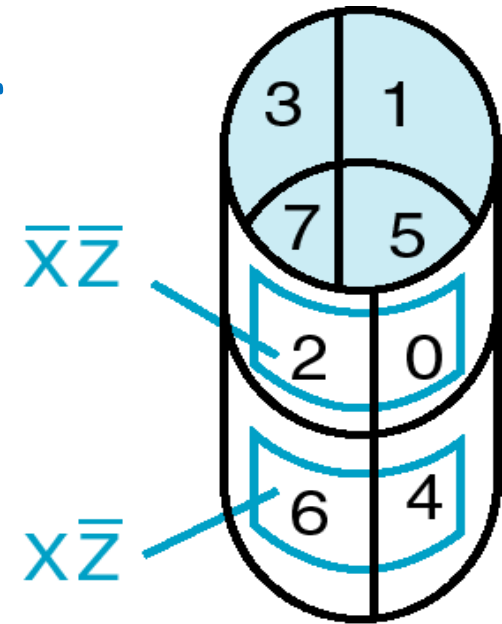
# Three-Variable Maps

✓ Topological warps of 3-variable K-maps that show *all* adjacencies:

■ Venn Diagram

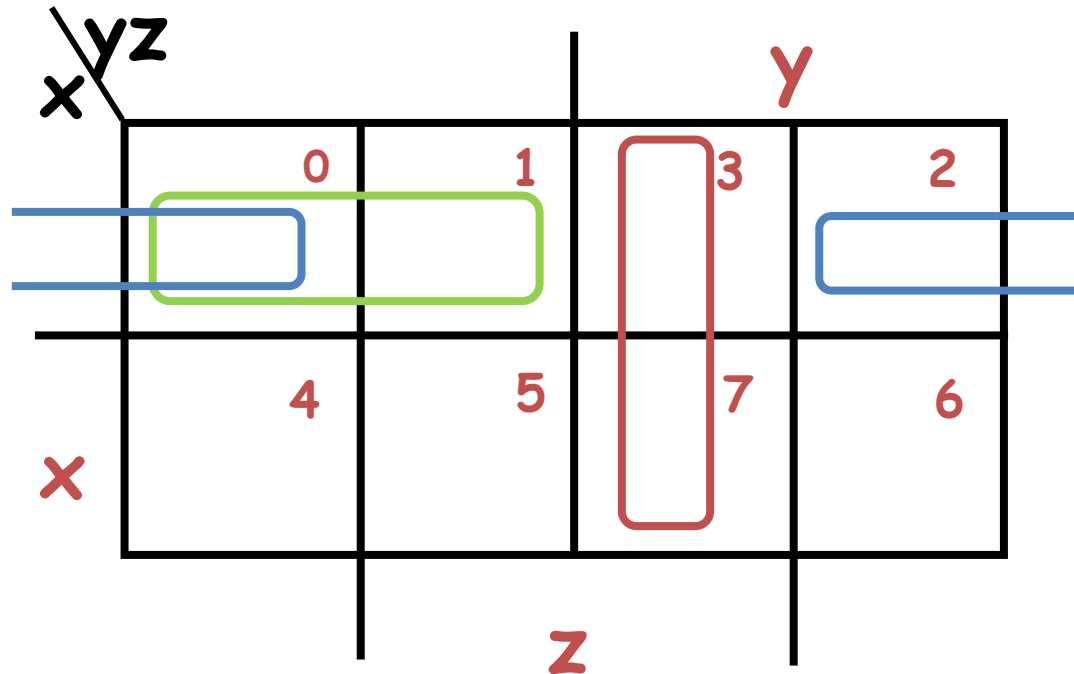


■ Cylinder



# Three-Variable Maps

- ✓ Example Shapes of 2-cell Rectangles:



- ✓ Read off the product terms for the rectangles shown

$x'z'$

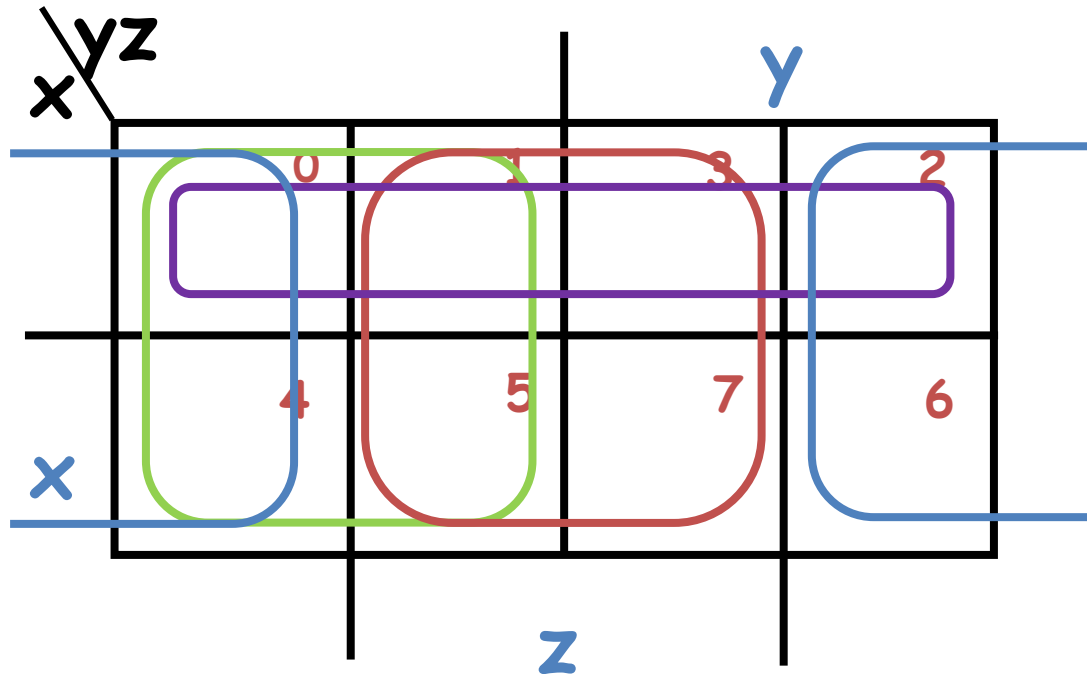
$x'y'$

$yz$

# Three-Variable Maps

---

- ✓ Example Shapes of 4-cell Rectangles:

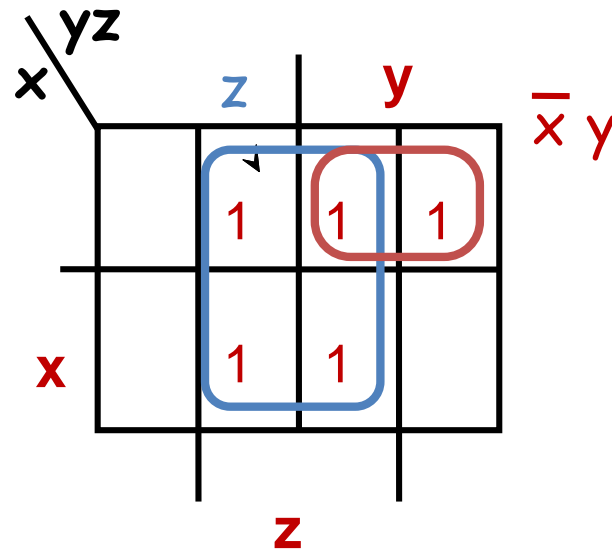


- ✓ Read off the product terms for the rectangles shown

$z'$     $z$     $x'$     $y'$

# Three Variable Maps

- ✓ K-Maps can be used to simplify Boolean functions by a systematic methods. Terms are selected to cover the "1s" in the map.
- ✓ Example: Simplify  $F(x, y, z) = \sum_m (1, 2, 3, 5, 7)$



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

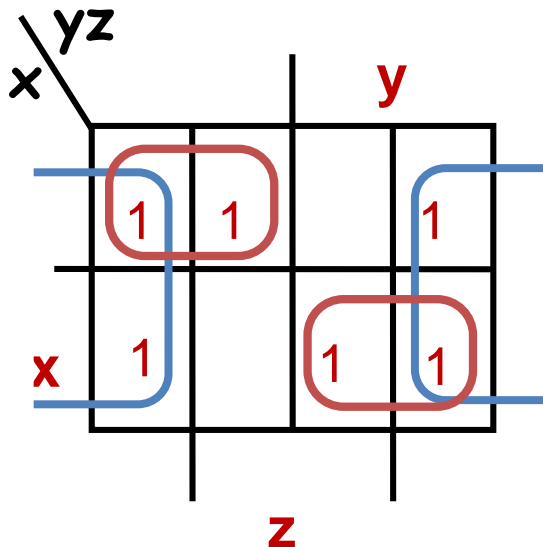
# Three-Variable Map Simplification

---

- ✓ Use a K-map to find an optimum SOP equation for

$$F(x, y, z) = \sum_m (0, 1, 2, 4, 6, 7)$$

$$F = \bar{z} + \bar{x}\bar{y} + xy$$





# Four Variable Maps

---

- ✓ Map and location of minterms:

$wx \backslash yz$		$y$		
		$yz=00$	$yz=01$	$yz=11$
$wx=00$	0	1	3	2
$wx=01$	4	5	7	6
$wx=11$	12	13	15	14
$wx=10$	8	9	11	10

$W$

$Z$

$X$

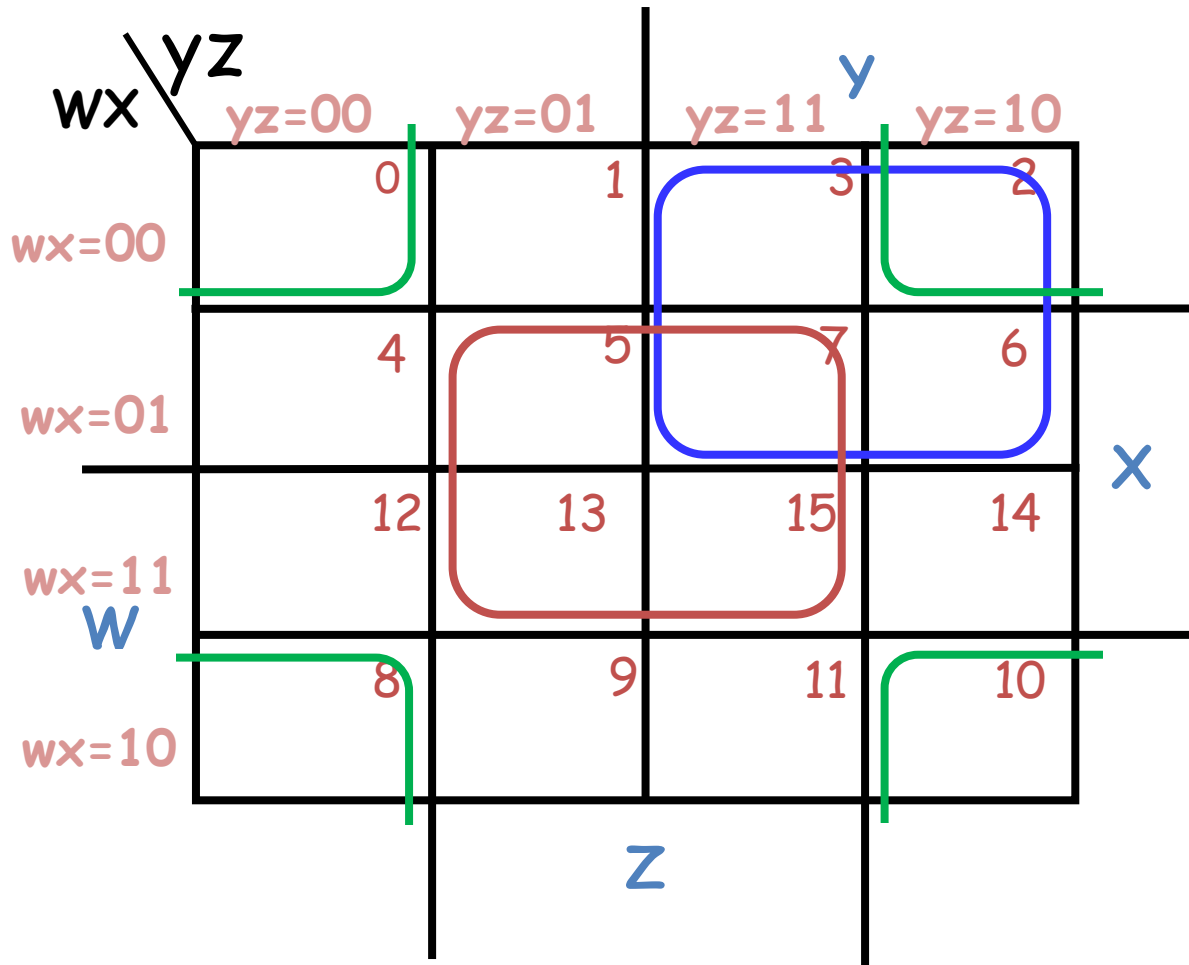
# Four Variable Terms

---

- ✓ Four variable maps can have rectangles corresponding to:
  - A single 1 = 4 variables, (i.e. Minterm)
  - Two 1s = 3 variables,
  - Four 1s = 2 variables
  - Eight 1s = 1 variable,
  - Sixteen 1s = zero variables (i.e. Constant "1")

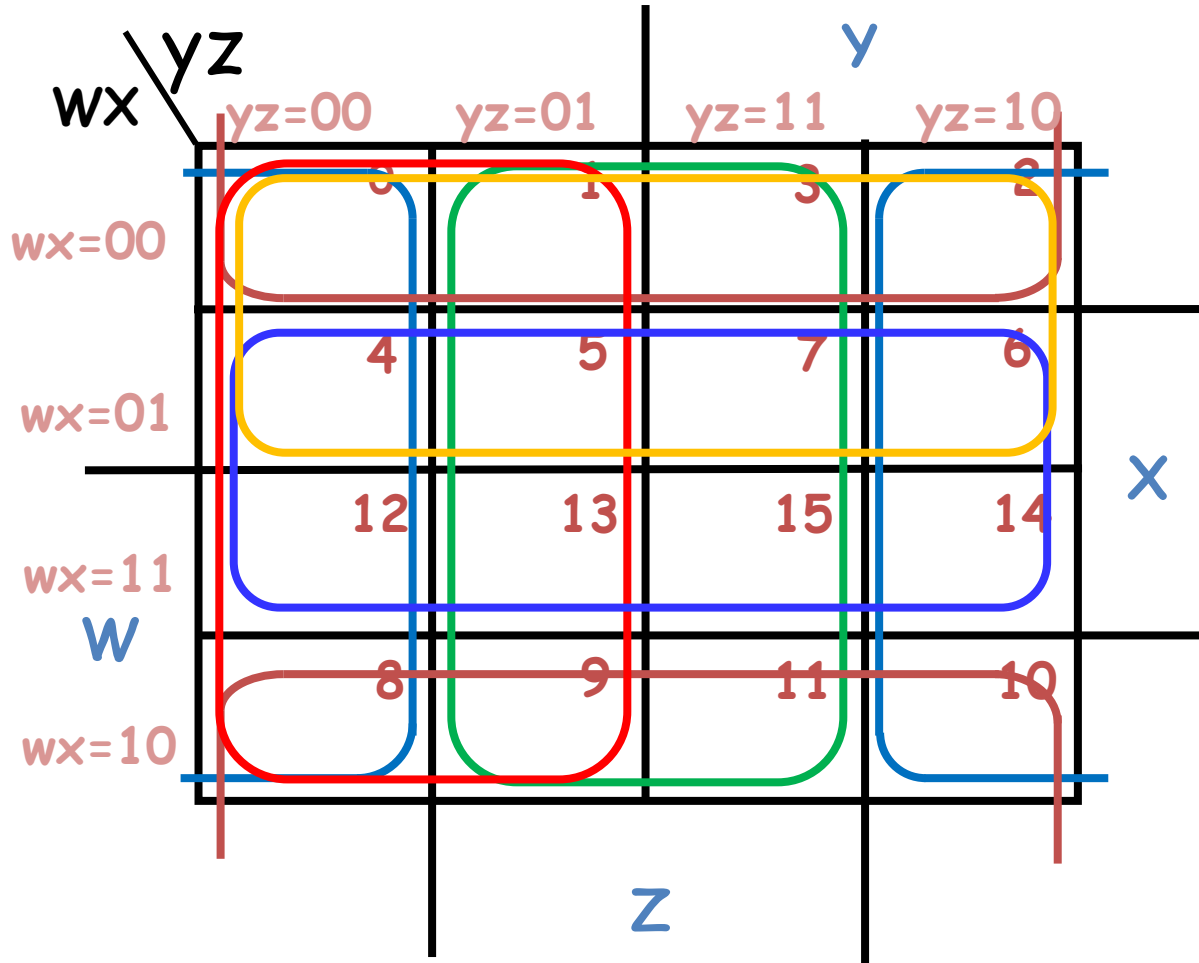
# Four-Variable Maps

- ✓ Example Shapes of Rectangles:



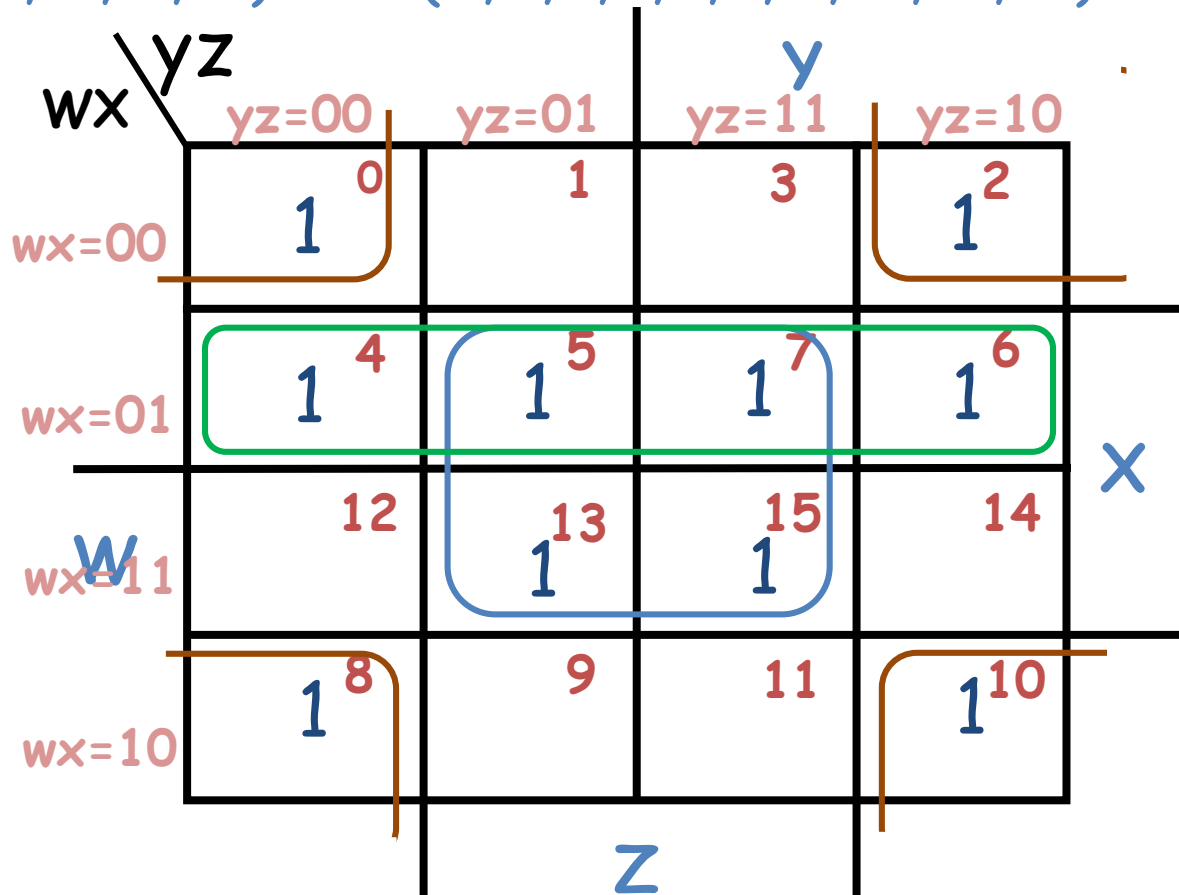
# Four-Variable Maps

- ✓ Example Shapes of Rectangles:



# Four-Variable Map Simplification

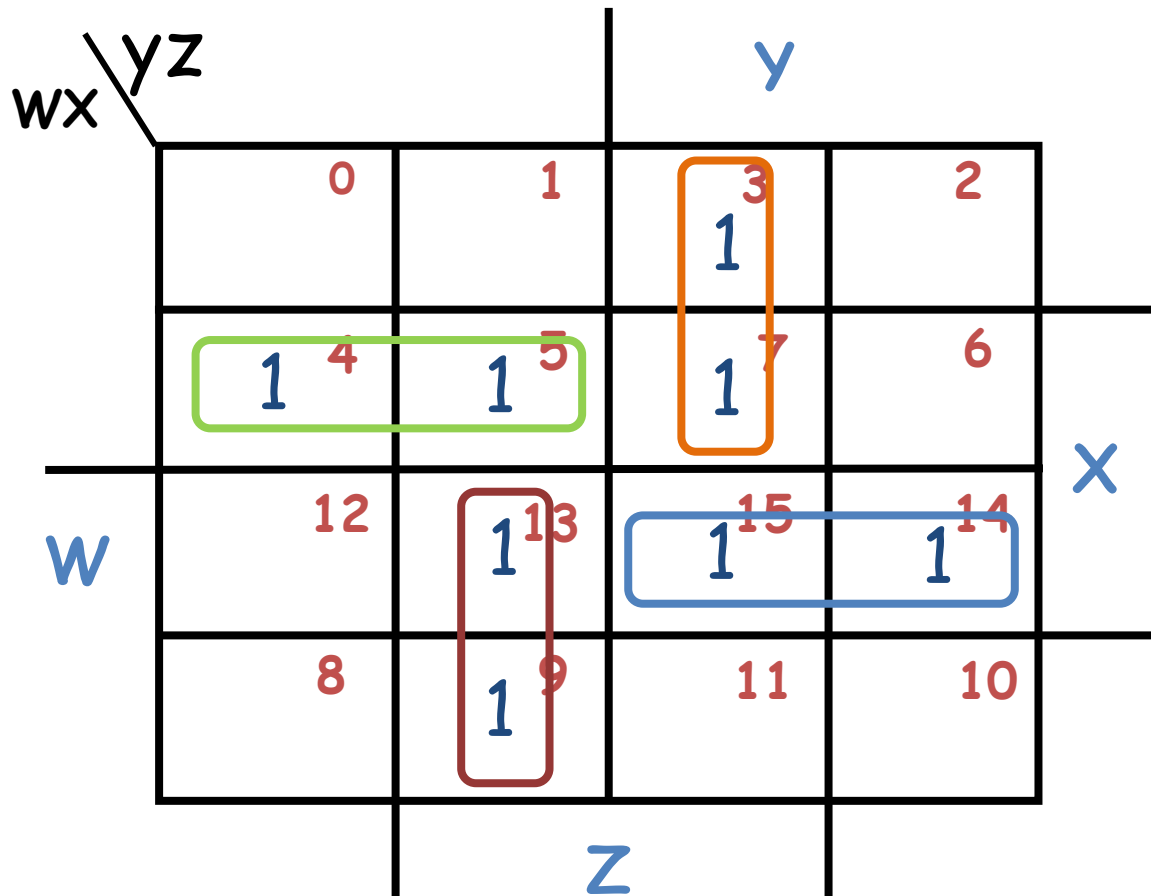
$$F(W, X, Y, Z) = \sum_m (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$$



$$F = XZ + X'Z' + W'X$$

# Four-Variable Map Simplification

$$F(W, X, Y, Z) = \sum_m(3, 4, 5, 7, 9, 13, 14, 15)$$



$$F = W' X Y' + W' Y Z + W X Y + W Y' Z$$

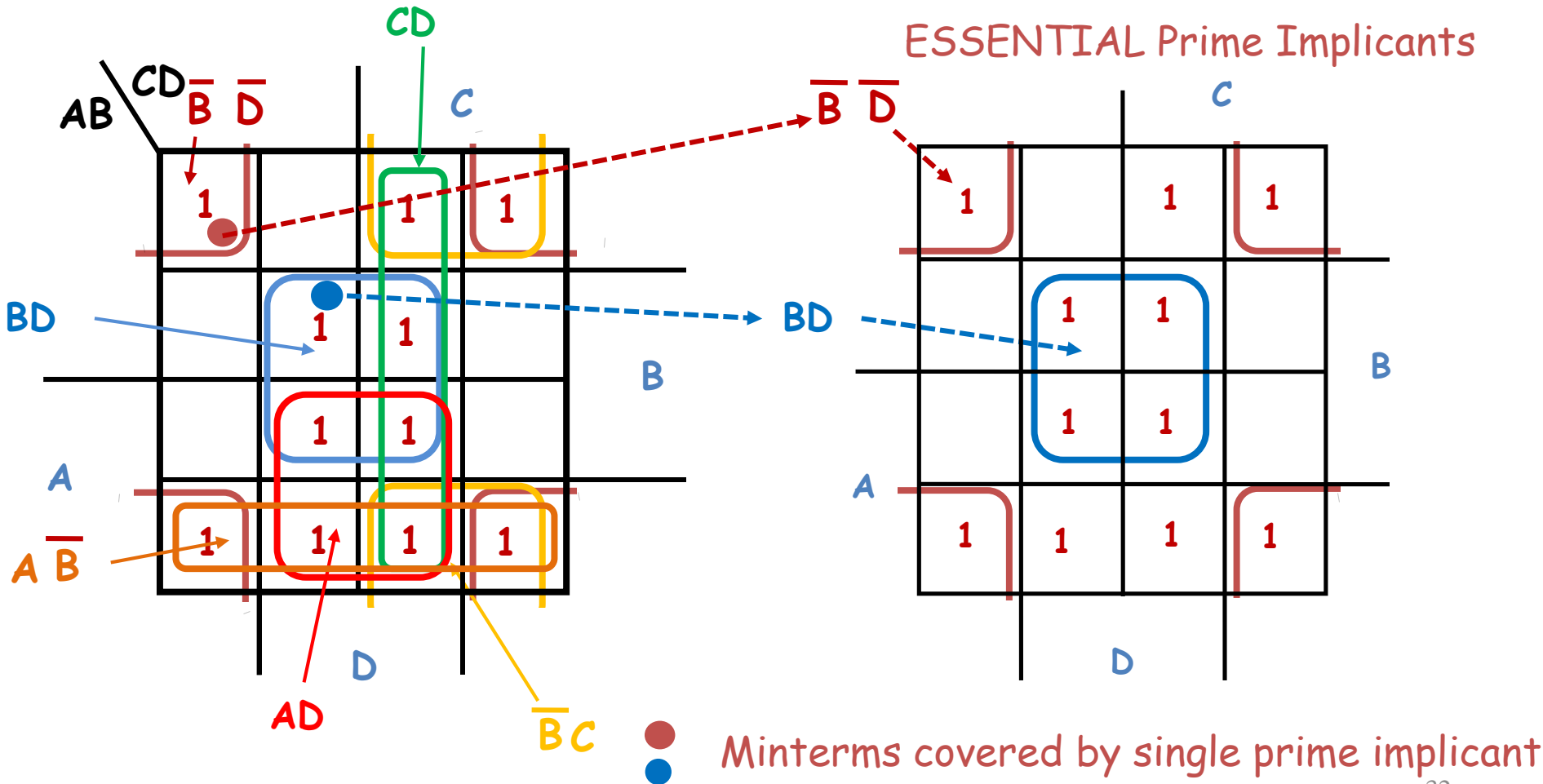
# Systematic Simplification

---

- ✓ A **Prime Implicant** is a cube i.e. 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 **only** 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

✓ Find ALL Prime Implicants

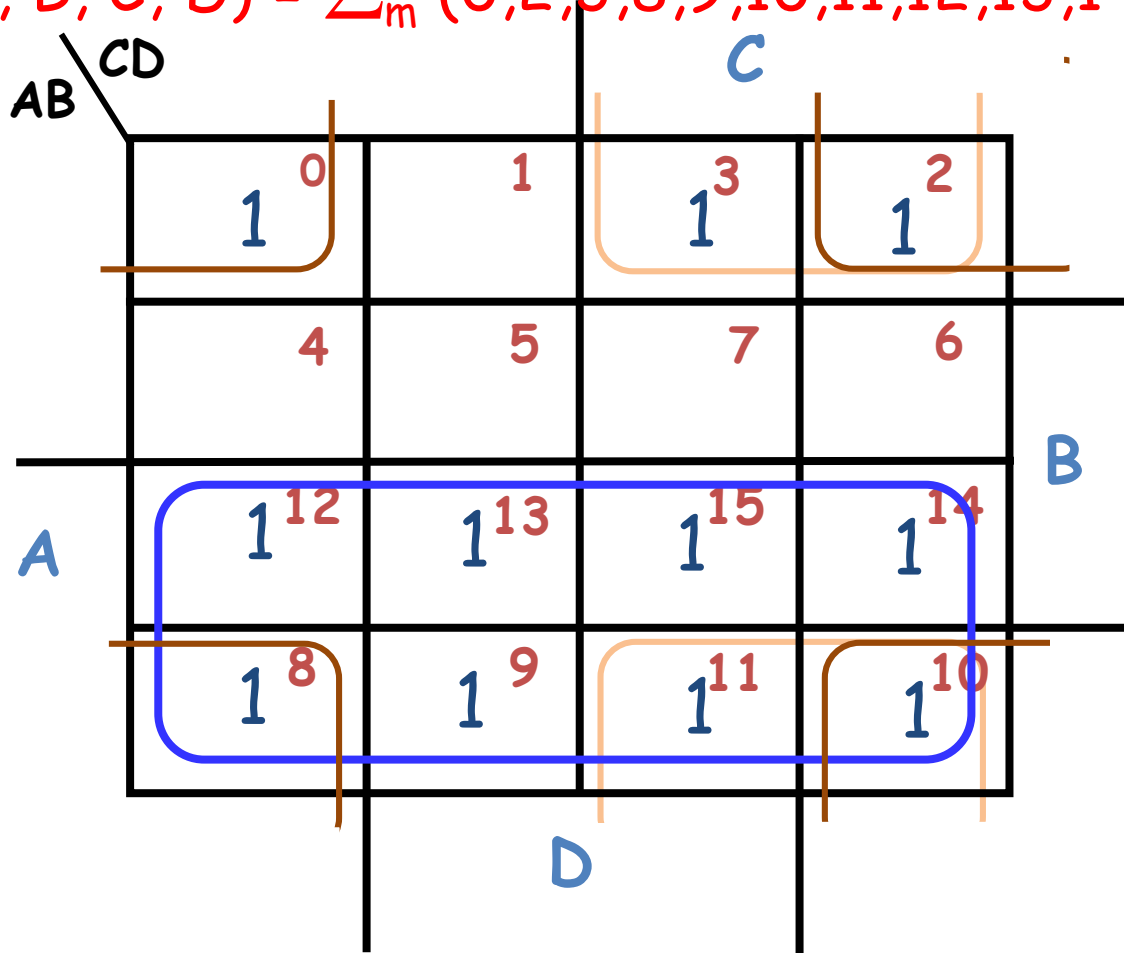




# Prime Implicant Practice

✓ Find all prime implicants for:

✓  $F(A, B, C, D) = \sum_m (0, 2, 3, 8, 9, 10, 11, 12, 13, 14, 15)$

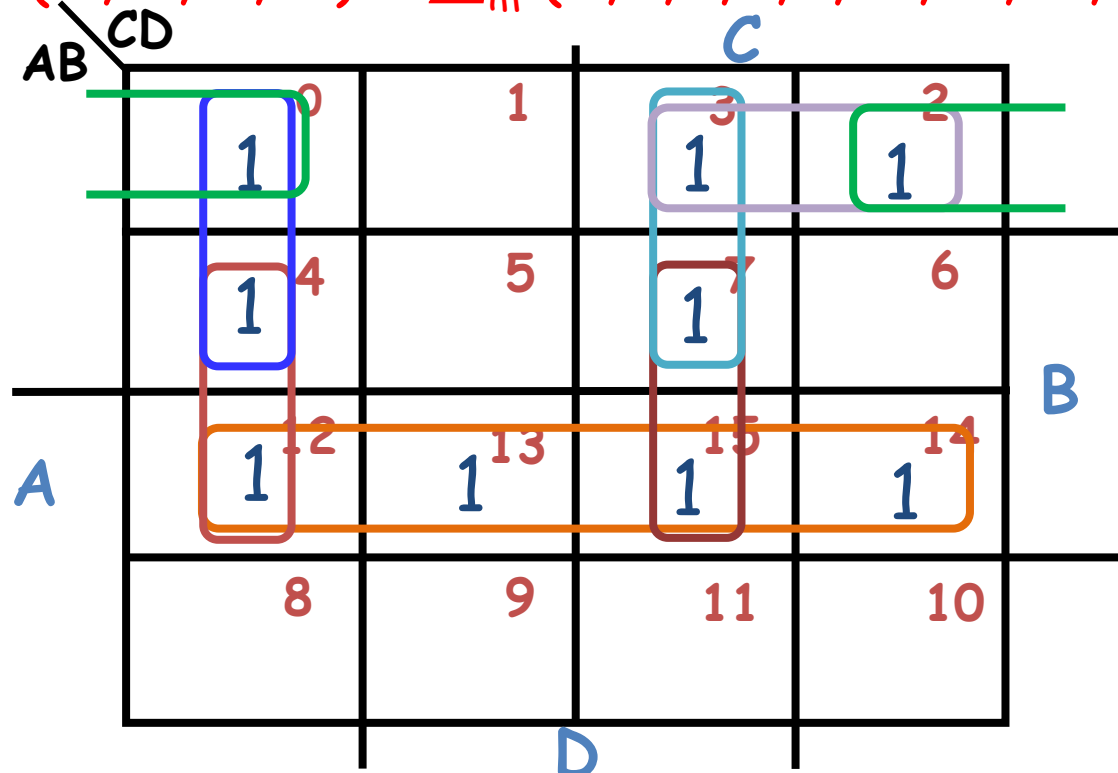


Prime implicants are:  $A$ ,  $\bar{B}C$ , and  $\bar{B}\bar{D}$

# Another Example

✓ Find all prime implicants for:

$$F(A, B, C, D) = \sum_m (0, 2, 3, 4, 7, 12, 13, 14, 15)$$



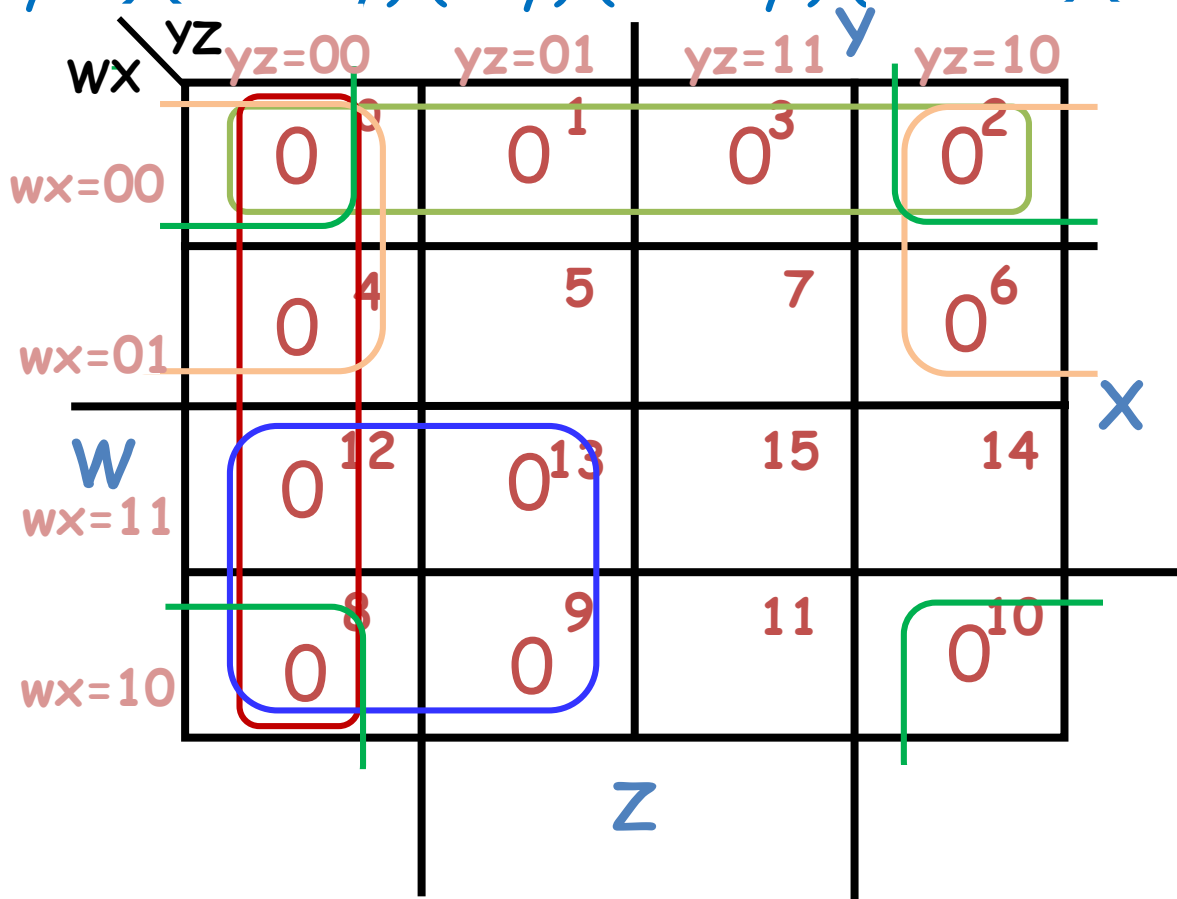
$$AB, B\bar{C}\bar{D}, \bar{A}\bar{C}\bar{D}, \bar{A}B\bar{D}, \bar{A}\bar{B}C, \bar{A}C\bar{D}, BCD$$

$$\sum_m = AB + \bar{A}\bar{C}\bar{D} + \bar{A}C\bar{D} + \bar{A}\bar{B}C \quad \sum_c = \sum_m + \bar{A}\bar{B}\bar{D} + BCD + B\bar{C}\bar{D} \quad 34$$

# K-Maps, implicates

✓ Find all prime implicates for:

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



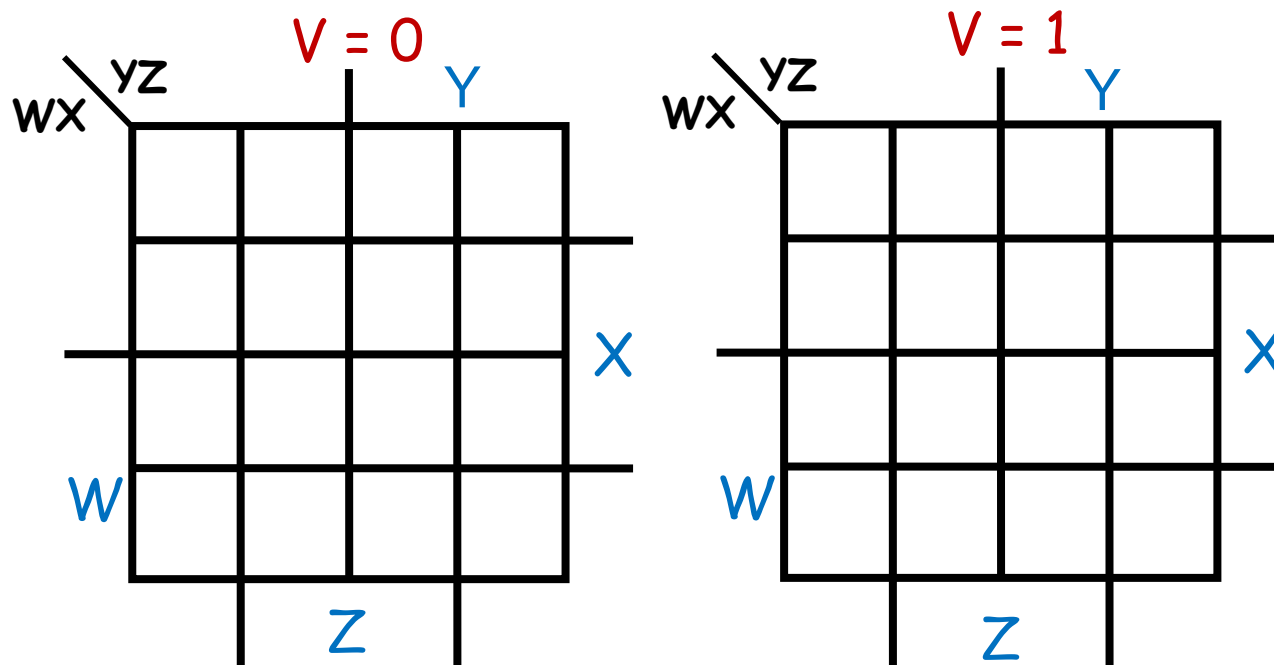
$$\Pi_m = (w+x)(w'+y)(w+z)(x+z)$$

$$\Pi_c = \Pi_m(y+z)$$

# Five Variable or More K-Maps

---

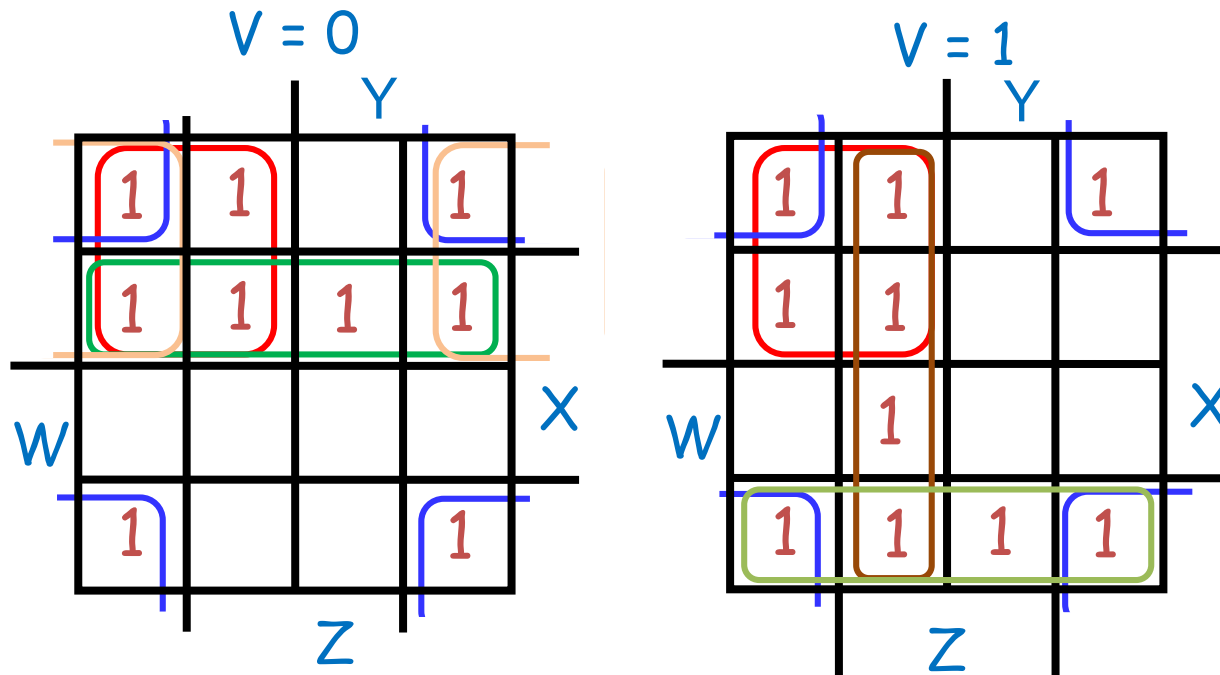
- ✓ For five variable problems, we use **two adjacent K-maps**. It becomes harder to visualize adjacent minterms for selecting PIs. You can extend the problem to six variables by using four K-Maps.



# Five Variable K-Maps

- ✓ Find all prime implicants for:

$$G(v,w,x,y,z) = \sum_m(0,1,2,4,5,6,7,8,10,16,17,18,20, 21,24,25,26,27,29)$$



$$\sum_m = w'y' + x'z' + v'w'x + vy'z + vw x' \quad \sum_c = \sum_m + v'w'z'$$

# Don't Cares in K-Maps

---

- ✓ Sometimes a function table or map contains entries for which it is known that:
  - 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: 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 **never occur**, so the output values for these codes are **x** to represent "don't cares."

# Incompletely Specified Functions

---

✓  $F = (f, d, r) : B^n \rightarrow \{0, 1, \mathbf{x}\}$

where  $\mathbf{x}$  represents "don't care".

- $f$  = onset function -  $f(x)=1 \leftrightarrow F(x)=1$
- $r$  = offset function -  $r(x)=1 \leftrightarrow F(x)=0$
- $d$  = don't care function -  $d(x)=1 \leftrightarrow F(x)=*$

$(f, d, r)$  forms a partition of  $B^n$ . i.e.

- $f + d + r = B^n$
- $fd = fr = dr = \emptyset$  (pairwise disjoint)

# Example: Logic Minimization

- ✓ Consider  $F(a,b,c) = (f,d,r)$ , where  $f = \{\bar{a}b\bar{c}, \bar{a}bc, abc\}$  and  $d = \{\bar{a}b\bar{c}, abc\}$ , and the sequence of covers illustrated below:

- On
- Off
- Don't care

$$F^1 = \bar{a}b\bar{c} + \bar{a}bc + abc$$

Expand  $abc \rightarrow a$



$$F^2 = a + \bar{a}b\bar{c} + \bar{a}bc$$

$\bar{a}bc$  is redundant

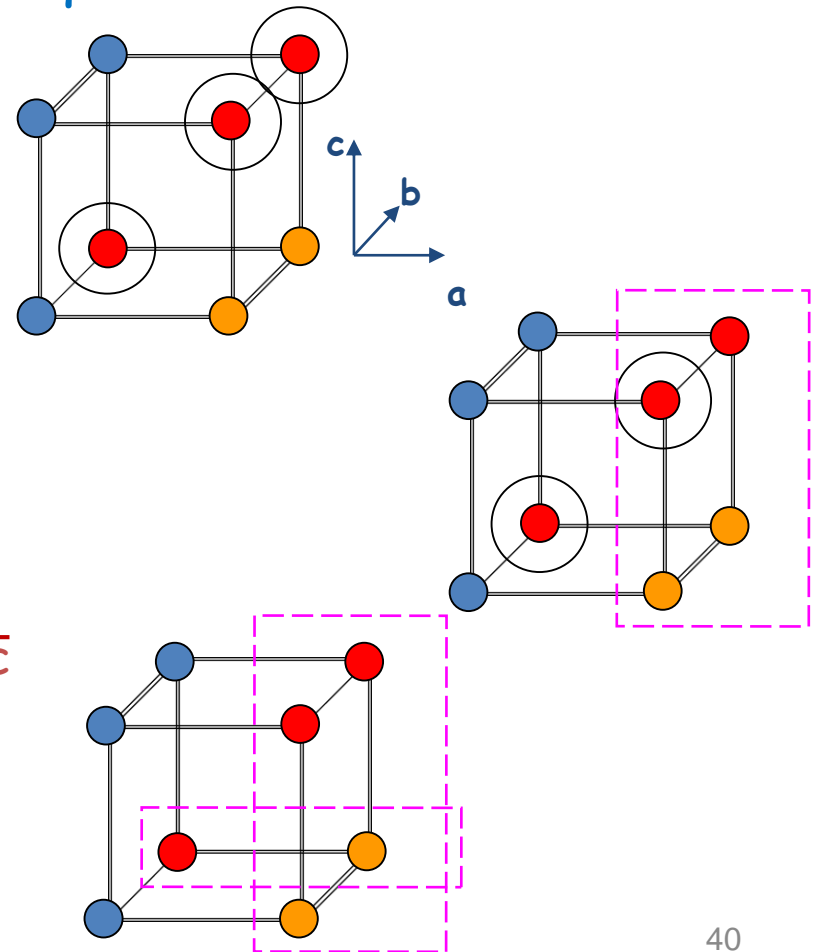
$a$  is prime

$$F^3 = a + \bar{a}b\bar{c}$$

Expand  $\bar{a}b\bar{c} \rightarrow b\bar{c}$



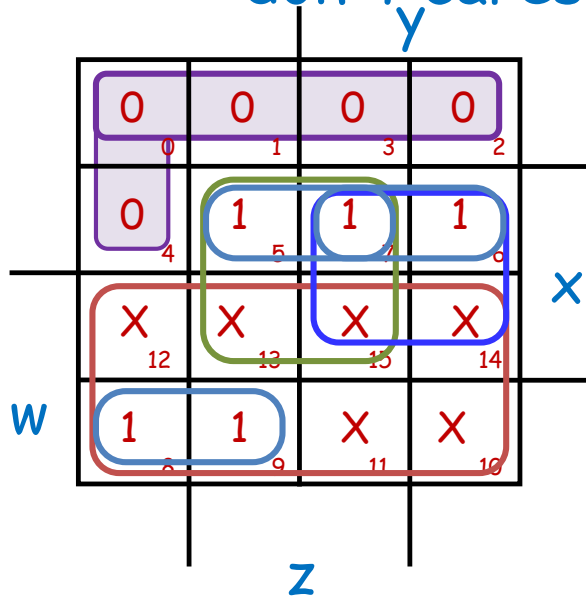
$$F^4 = a + b\bar{c}$$





# Example: BCD "5 or More"

- ✓ In the map below gives a function  $F1(w,x,y,z)$  which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:



$$F1(w, x, y, z) = \bar{w} x z + \bar{w} x y + w \bar{x} \bar{y}$$

$$F2(w, x, y, z) = w + x z + x y$$

$$G = 7$$

This is much lower in cost than F1 where the "don't cares" were treated as "0s" having  $G = 12$

For this particular function, cost  $G$  for the POS solution for  $F1(w,x,y,z)$  is not changed by using the don't cares.

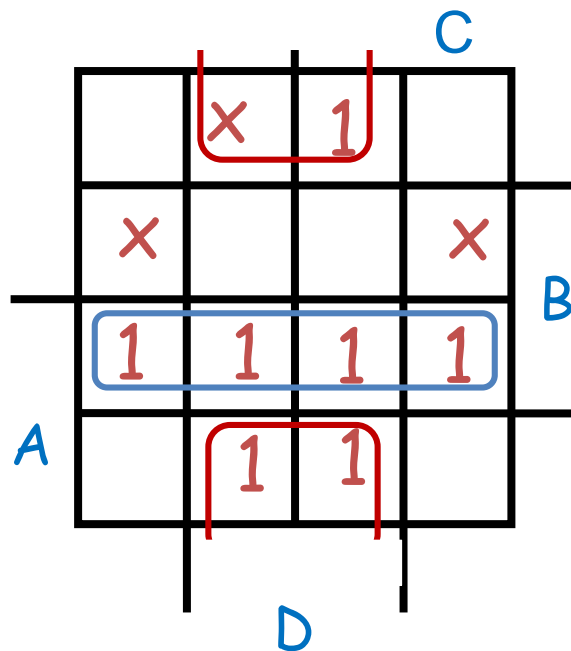
$$F3(w, x, y, z) = (w + x) (w + y + z) \quad G = 7$$

# Product of Sums Example

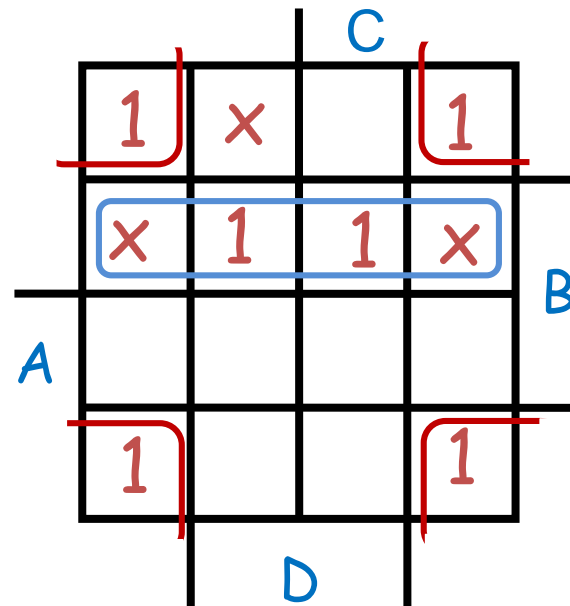
- ✓ Find the optimum POS solution:

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

- Hint: Use  $F'$  and complement it to get the result.



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

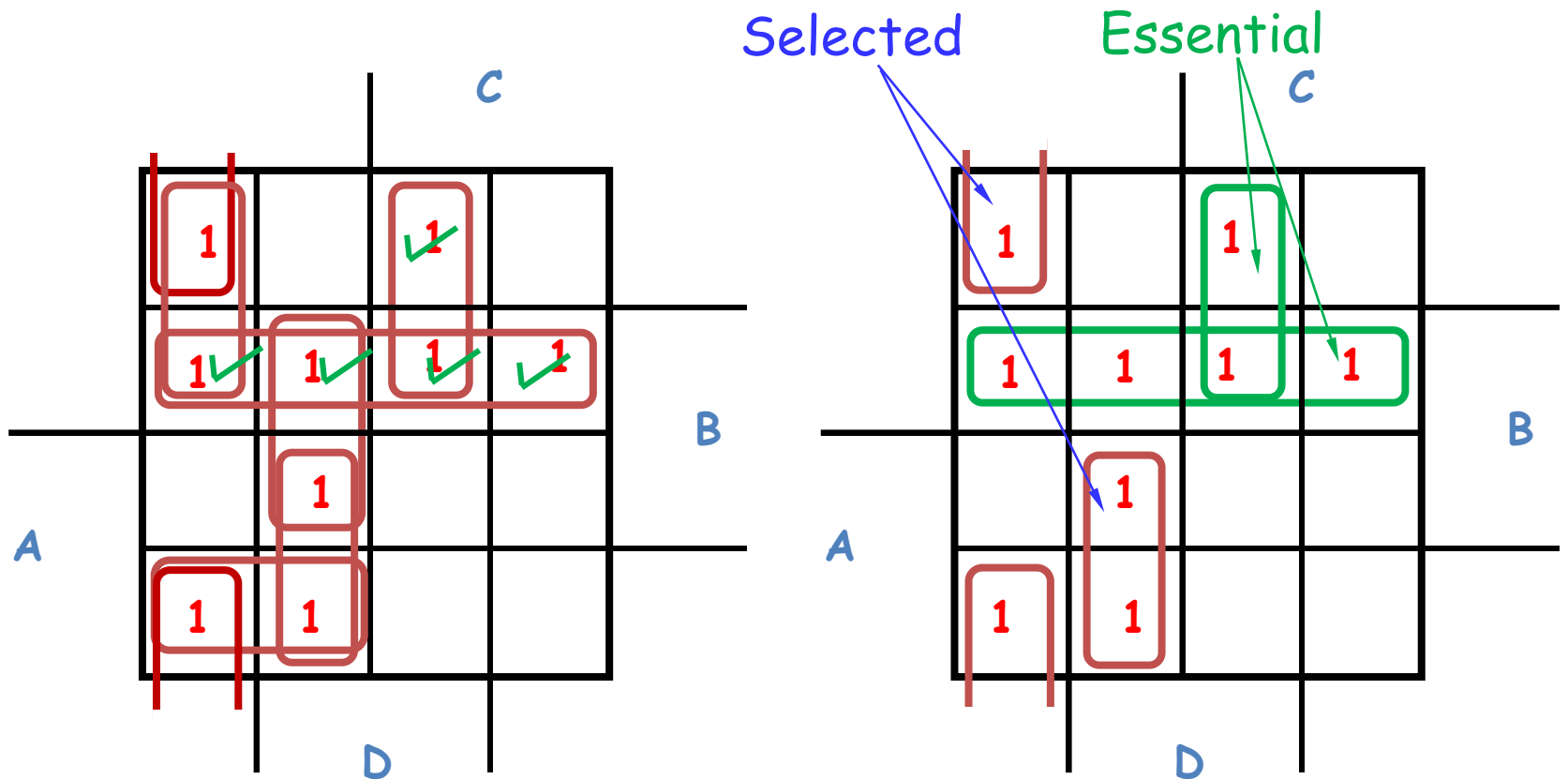


$$F' = B' D' + A' B$$

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

# Selection Rule Example

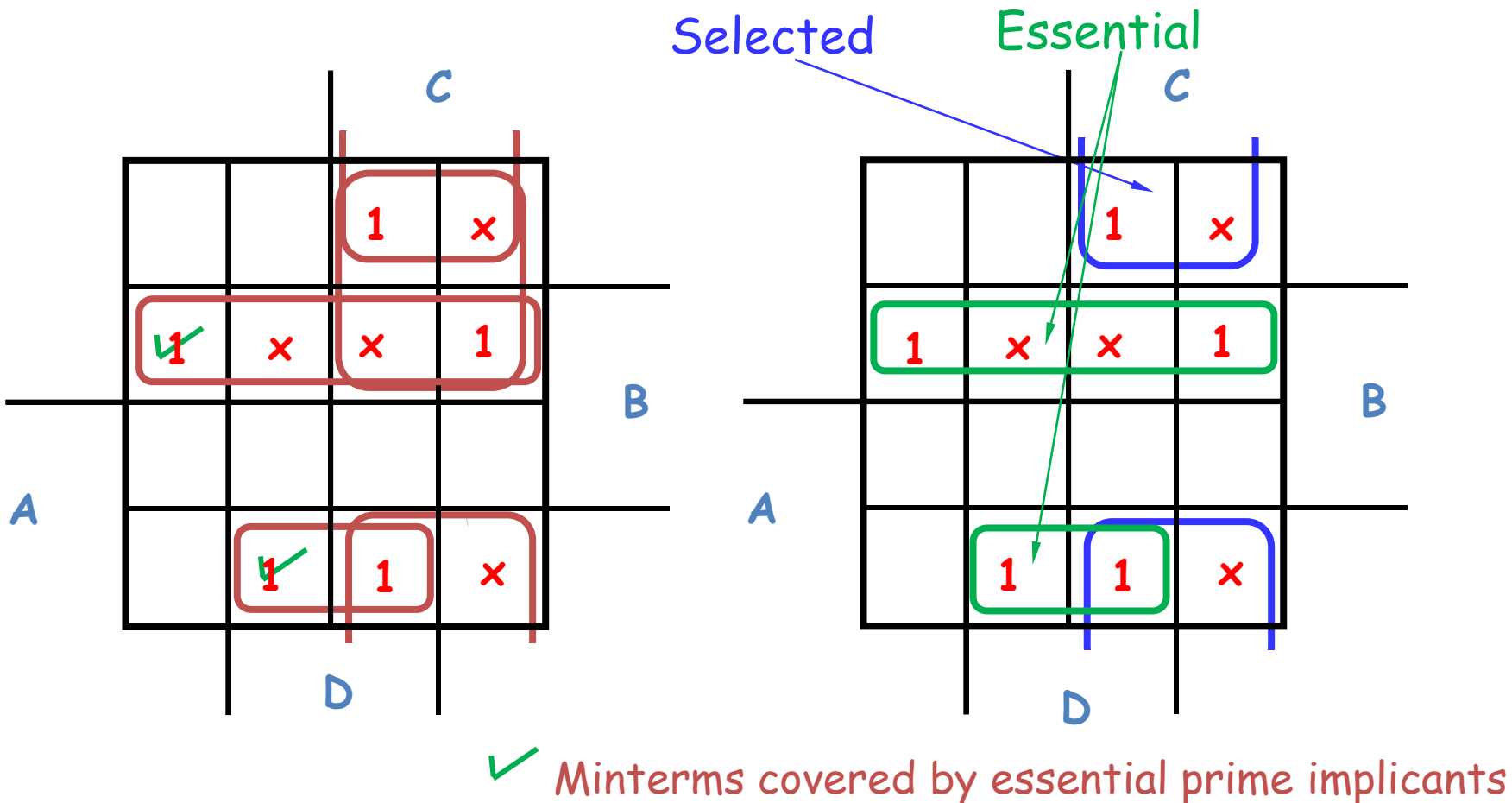
- ✓ Simplify  $F(A, B, C, D)$  given on the K-map.



✓ Minterms covered by essential prime implicants

# Selection Rule Example with Don't Cares

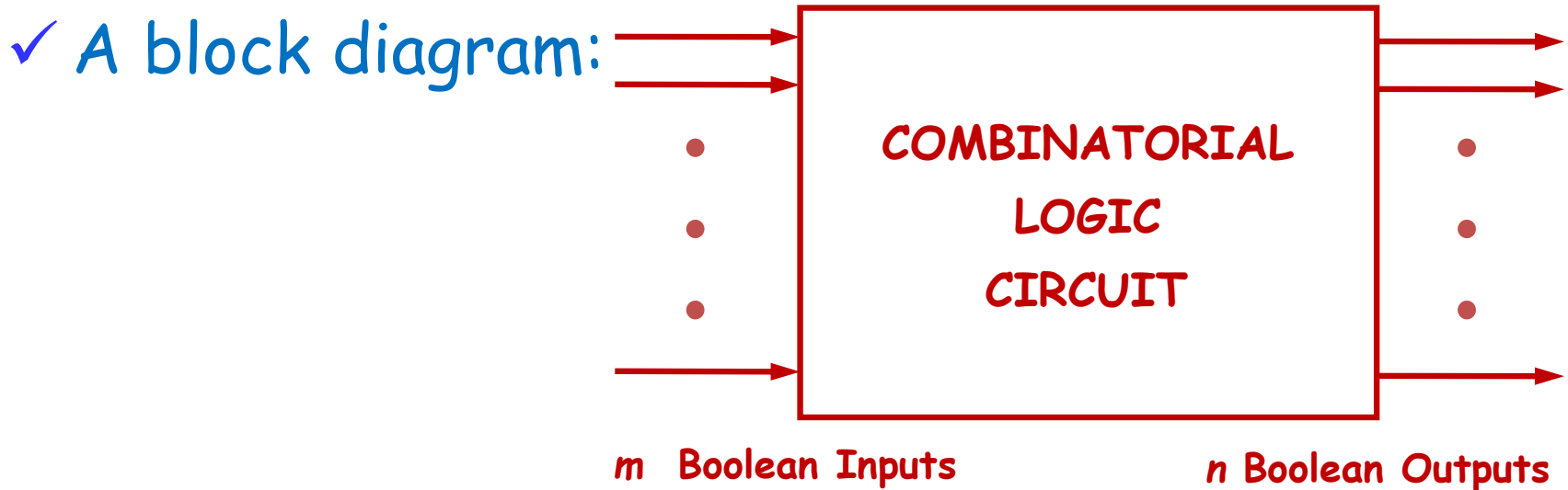
- ✓ Simplify  $F(A, B, C, D)$  given on the K-map.



# Combinational Circuits

---

- ✓ A combinational logic circuit has:
  - A set of  $m$  Boolean inputs,
  - A set of  $n$  Boolean outputs, and
  - $n$  switching functions, each mapping the  $2^m$  input combinations to an output such that the current output depends only on the current input values



# Design Procedure

---

## ✓ Specification

- Write a specification for the circuit

## ✓ Formulation

- Derive a truth table or initial Boolean equations that define the required relationships between the inputs and outputs

## ✓ Optimization

- Apply 2-level and multiple-level optimization
- Draw a logic diagram for the resulting circuit using ANDs, ORs, and inverters

## ✓ Technology Mapping

- Map the logic diagram to the implementation technology selected

## ✓ Verification

- Verify the correctness of the final design

# Design Example BCD to Excess-3 code converter

---

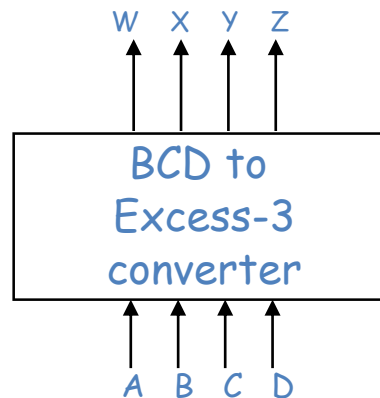
## ✓ Specification

- BCD to Excess-3 code converter
  - Transforms BCD code for the decimal digits to Excess-3 code for the decimal digits
- BCD code words for digits 0 through 9
  - 4-bit patterns 0000 to 1001, respectively
- Excess-3 code words for digits 0 through 9
  - 4-bit patterns consisting of 3 (binary 0011) added to each BCD code word
- Implementation:
  - multiple-level circuit
  - NAND gates (including inverters)

# Specification of BCD-to-Excess3

---

- ✓ Inputs: a BCD input,  $A, B, C, D$  with  $A$  as the most significant bit and  $D$  as the least significant bit.
- ✓ Outputs: an Excess-3 output  $W, X, Y, Z$  that corresponds to the BCD input.
- ✓ Internal operation - circuit to do the conversion in combinational logic.





# Formulation of BCD-to-Excess-3

---

- ✓ Excess-3 code is easily formed by adding a binary 3 to the binary or BCD for the digit.
- ✓ There are 16 possible inputs for both BCD and Excess-3.
- ✓ It can be assumed that only valid BCD inputs will appear so the six combinations not used can be treated as don't cares.

# BCD-to-Excess-3 Code converter

---

<b>Decimal Digit</b>	<b>Input BCD</b>	<b>Output Excess-3</b>
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

# Expressions for W X Y Z

---

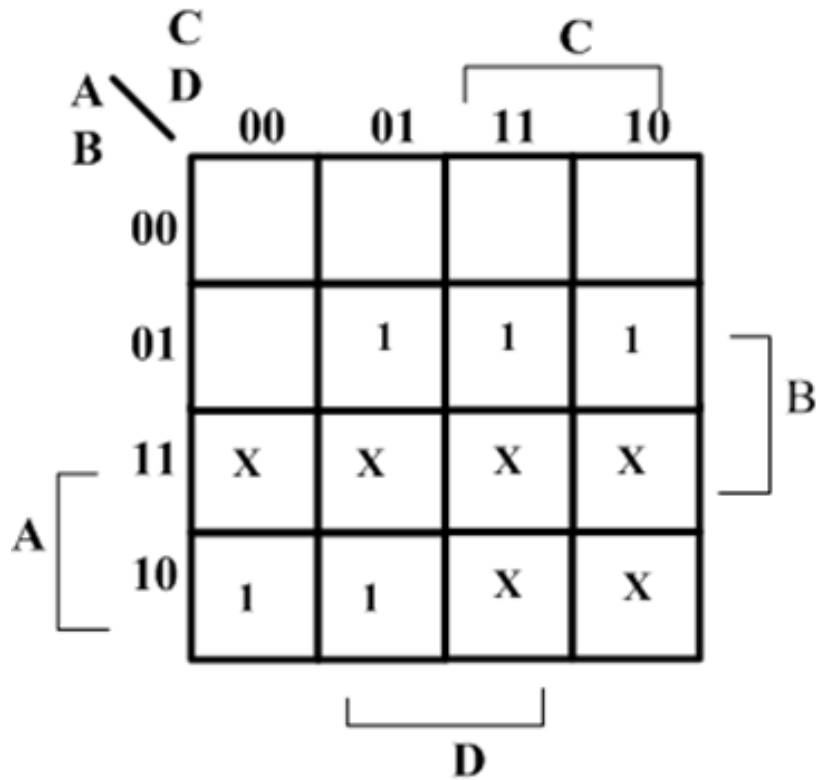
- ✓  $W(A,B,C,D) = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$
- ✓  $X(A,B,C,D) = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$
- ✓  $Y(A,B,C,D) = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$
- ✓  $Z(A,B,C,D) = \Sigma m(0,2,4,6,8) + d(10,11,12,13,14,15)$

<b>Decimal Digit</b>	<b>Input BCD</b>	<b>Output Excess-3</b>
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

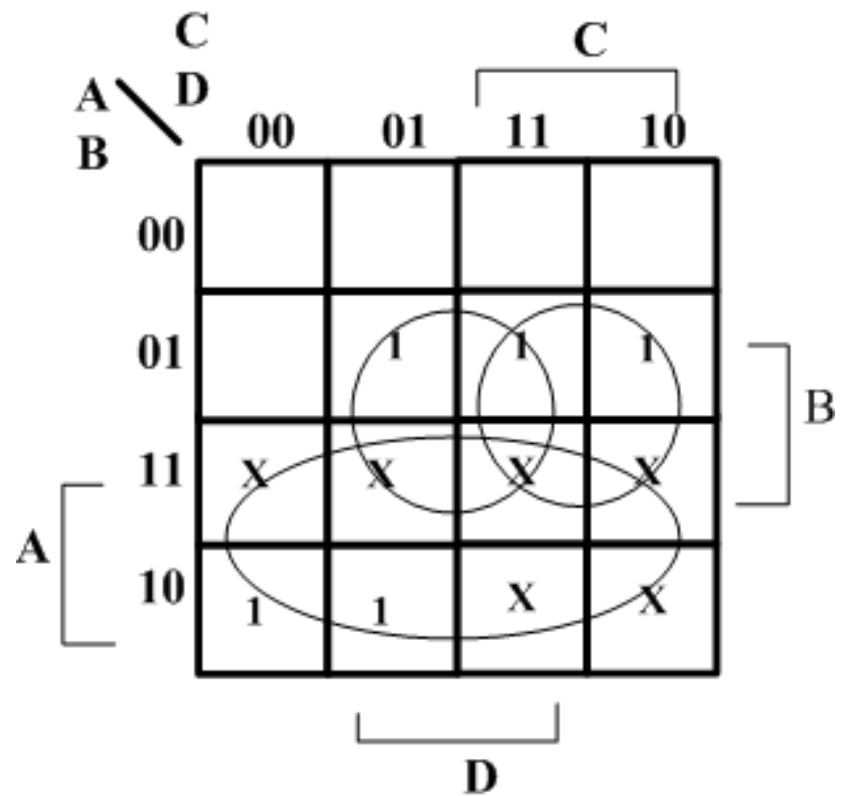
# Optimization - BCD-to-Excess-3

- ✓ Lay out K-maps for each output, W X Y Z

K-map for W  $\Sigma_m(5,6,7,8,9)$



K-map for W

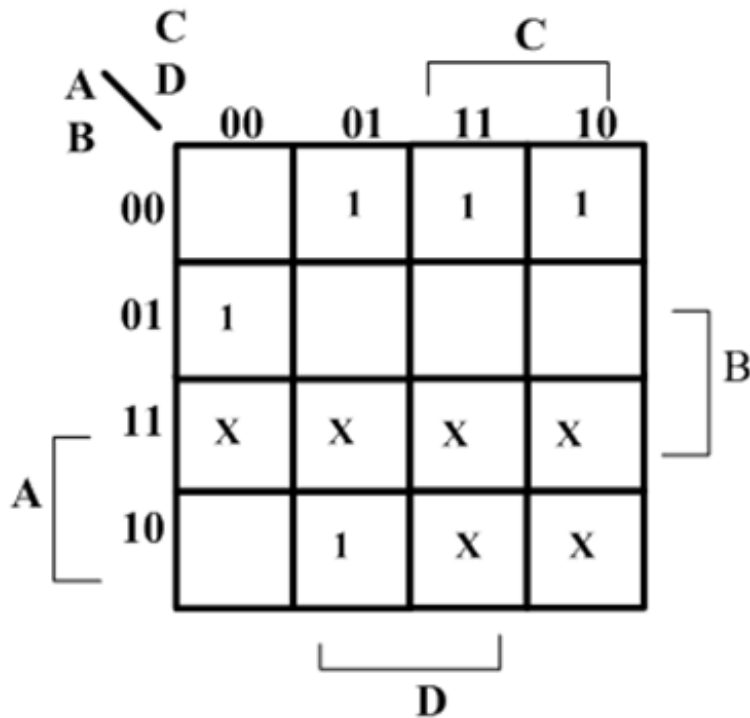


$$W = A + BC + BD$$

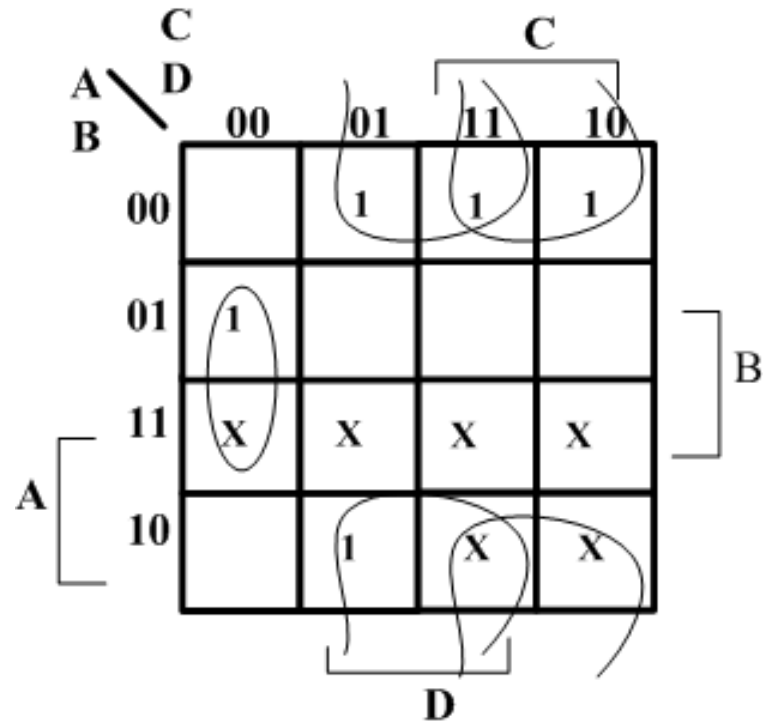
# Optimization - BCD-to-Excess-3

- ✓ Lay out K-maps for each output, W X Y Z

K-map for X  $\Sigma_m(1,2,3,4,9)$



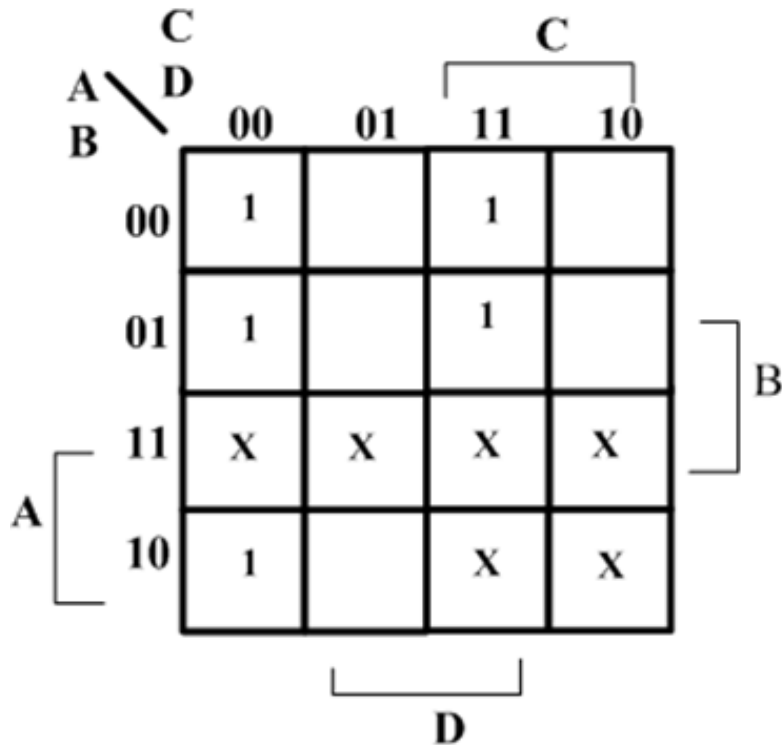
K-map for X



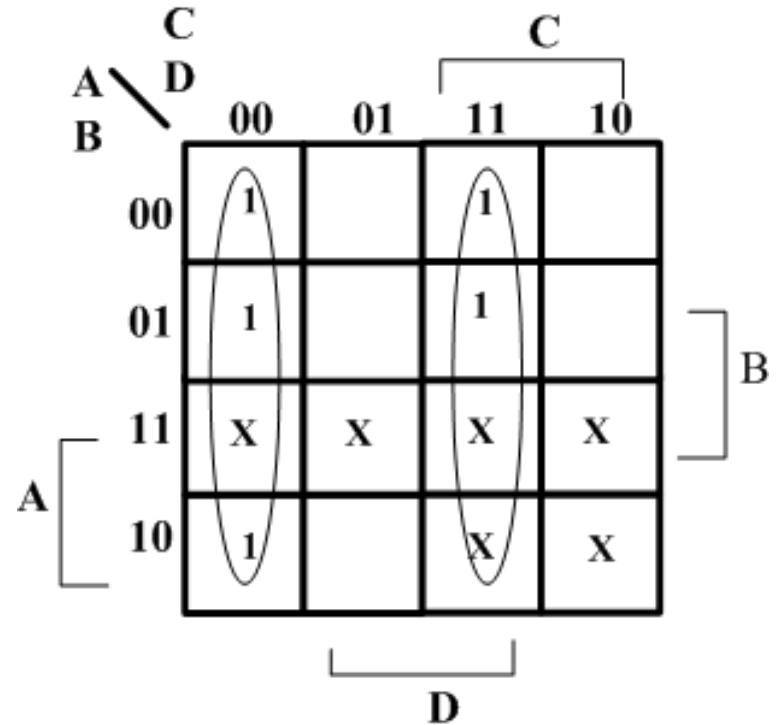
$$X = BC'D' + B'C + B'D$$

# Optimization - BCD-to-Excess-3

K-map for Y  $\Sigma_m(0,3,4,7,8)$



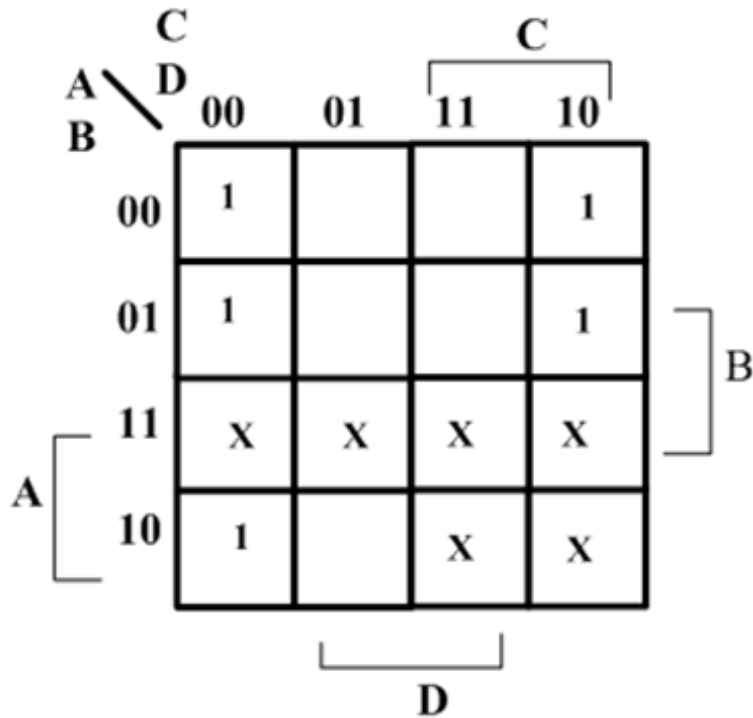
K-map for Y



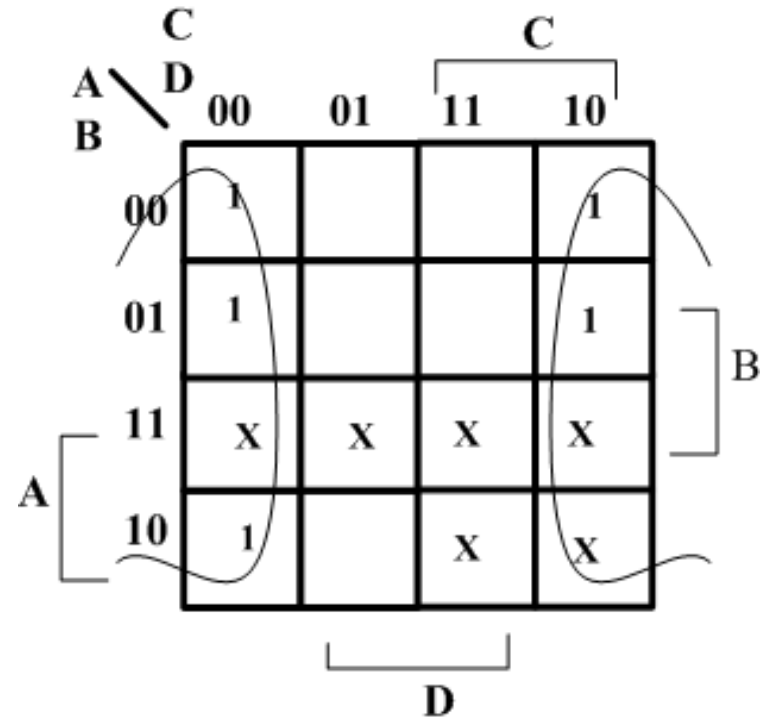
$$Y = C'D' + CD$$

# Minimize K-Maps

K-map for Z  $\Sigma_m(0,2,4,6,8)$



K-map for Z



$Z = D'$

# Two level versus multi-level circuit implementation

---

## ✓ Have equations

- $W = A + BC + BD$
- $X = B'C + B'D + BC'D'$
- $Y = CD + C'D'$
- $Z = D'$

## ✓ Call $T = C + D$ , then $T' = (C + D)' = C'D'$

- $W = A + BT$
- $X = B'T + BT'$
- $Y = CD + T'$
- $Z = D'$



# Create the digital circuit

---

- ✓ Implementing the second set of equations where  $T=C+D$  results in a lower gate count.
- ✓ This gate has a fanout of 3

