DIGITAL DESIGN AND COMPUTER ORGANIZATION

Gate-Level Minimization : Karnaugh Maps

Team DDCO
Department of Computer Science and Engineering

Gate-Level Minimization and Combinational logic Introduction :Chapter 3: 3.1,3.2,3.3,3.5

Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.

Logic minimizations

Boolean Identities

K-map.(Karnaugh map)

Simplest algebraic expression is an algebraic expression with a **minimum number of terms** and with the **smallest possible number of literals in each term**. This expression produces a circuit diagram with a minimum number of gates and the minimum number of inputs to each gate.

Introduction

- The complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented.
- Boolean expressions may be simplified by algebraic means. However, this procedure of minimization is awkward because it lacks specific rules to predict each succeeding step in the manipulative process.
- The map method provides a simple, straightforward procedure for minimizing Boolean functions.
- This method may be regarded as a pictorial form of a truth table. The map method is also known as the ***Karnaugh map*** or ***K-map***

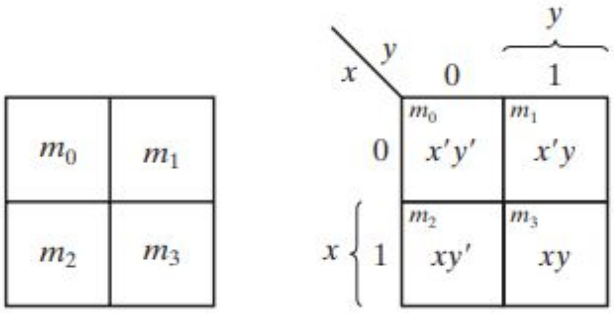
Karnaugh Map

- **K-Map** is a **visual tool** used to simplify Boolean expressions.
- Introduced by **Maurice Karnaugh in 1953**, based on earlier work from 1881.
- Each **square in the K-map represents a minterm** of a Boolean function.
- A Boolean function can be expressed as a **sum of minterms** – this makes it easy to map onto a K-map.
- **Adjacent squares** represent minterms that differ by **only one literal**.
- Uses our brain's **pattern recognition ability** to simplify logic.
- Groups of 1s (e.g., 1, 2, 4, 8...) are formed to find **common literals**, which helps in **minimization**.

Two-Variable K-Map

- There are four minterms for two variables; hence, the map consists of four squares, one for each minterm.
- The map is redrawn in (b) to show the relationship between the squares and the two variables x and y.
- The 0 and 1 marked in each row and column designate the values of variables.

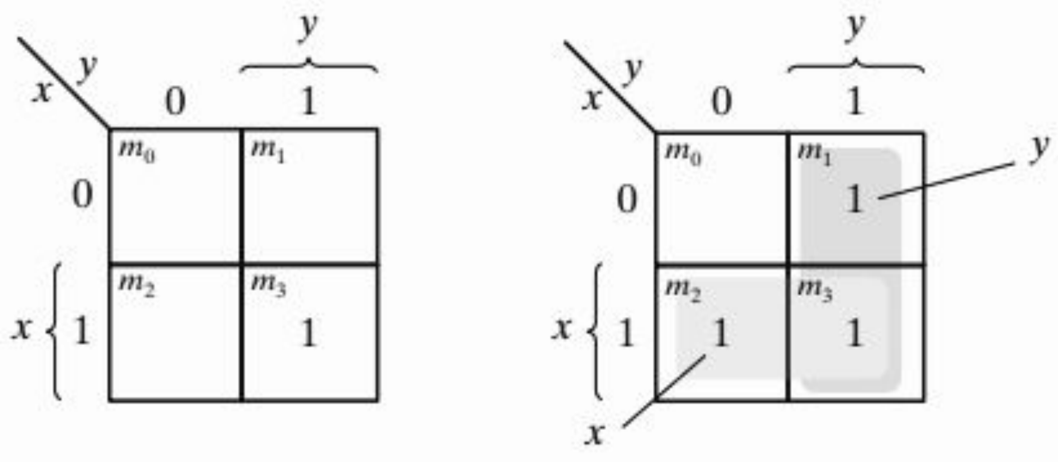
$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y$$



(a) (b)

Two-Variable K-Map

Representation of functions in the map: The function x + y is represented in the map as follows:



(a) xy (b) $x + y$

Two-Variable K-Map

- The three squares could also have been determined from the intersection of variable x in the second row and variable y in the second column, which encloses the area belonging to x or y .
- In each example, the minterms at which the function is asserted are marked with a 1.

Three-Variable K-Map

- The characteristic of this sequence is that only one bit changes in value from one adjacent column to the next.

- The map drawn in part (b) is marked with numbers in each row and each column to show the relationship between the squares and the three variables.

- **Gray code** ensures that only one variable changes between each pair of adjacent cells.

Three-Variable K-Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

y

z

(b)

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Three-Variable K-Map

a	b	c	y	minterm
0	0	0	0	$\bar{a}\bar{b}\bar{c}$
0	0	1	0	$\bar{a}\bar{b}c$
0	1	0	0	$\bar{a}b\bar{c}$
0	1	1	1	$\bar{a}bc$
1	0	0	1	$a\bar{b}\bar{c}$
1	0	1	0	$a\bar{b}c$
1	1	0	1	$ab\bar{c}$
1	1	1	1	abc

a \ bc	00	01	11	10
0	$\bar{a}\bar{b}\bar{c}_0$	$\bar{a}\bar{b}c_1$	$\bar{a}bc_3$	$\bar{a}b\bar{c}_2$
1	$a\bar{b}\bar{c}_4$	$a\bar{b}c_5$	abc_7	$ab\bar{c}_6$

a \ bc	00	01	11	10
0	0_0	0_1	1_3	0_2
1	1_4	0_5	1_7	1_6

- Each square corresponds to a row of the truth table
- Any two adjacent squares differ only in one literal
- Achieved using two rows and binary order 00,01,11,10
- Notion of “wrap-around”: far left and right squares are adjacent

K-Map Method

K-Map Implicants

- Implicant
  - K-Map area composed of squares containing 1's
  - Area is square or rectangular (wraparound allowed)
  - No. of squares in area is a power of two (1, 2, 4, . . .)
  - Each implicant corresponds to a product of literals
    - Double the area, one less literal
- Prime implicant
  - Implicant having largest number of squares obeying above rules
- Essential prime implicant
  - Prime implicant containing a square not in any other prime implicant

K-Map Method

- Include all required prime implicants
  - Include all essential prime implicants
  - Include other prime implicants such that:
    - Each square containing 1 is covered
    - Boolean formula is minimal (may not be unique)
- Convert required implicants to Boolean formula
  - Each implicant is a product of literals
  - Include literals which do not change over its area

K-Map Method

In choosing adjacent squares in a map, we must ensure that

(1) all the minterms of the function are covered when we combine the squares,
(2) the number of terms in the expression is minimized, and
(3) there are no redundant terms (i.e., minterms already covered by other terms)

A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.
If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential.

The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares.

K-Map Example

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour

a \ bc	00	01	11	10
0	0_{0}	0_{1}	1_{3}	0_{2}
1	1_{4}	0_{5}	1_{7}	1_{6}

a	b	c	y	minterm
0	0	0	0	$\bar{a}\bar{b}\bar{c}$
0	0	1	0	$\bar{a}\bar{b}c$
0	1	0	0	$\bar{a}b\bar{c}$
0	1	1	1	$\bar{a}bc$
1	0	0	1	$a\bar{b}\bar{c}$
1	0	1	0	$a\bar{b}c$
1	1	0	1	$ab\bar{c}$
1	1	1	1	abc

K-Map Example

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour

a \ bc	00	01	11	10
0	0_{0}	0_{1}	1_{3}	0_{2}
1	1_{4}	0_{5}	1_{7}	1_{6}

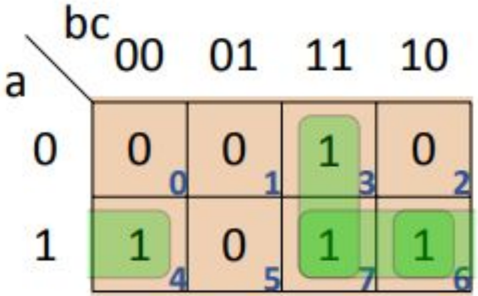
K-Map Example

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour

a \ bc	00	01	11	10
0	0_{0}	0_{1}	1_{3}	0_{2}
1	1_{4}	0_{5}	1_{7}	1_{6}

K-Map Example

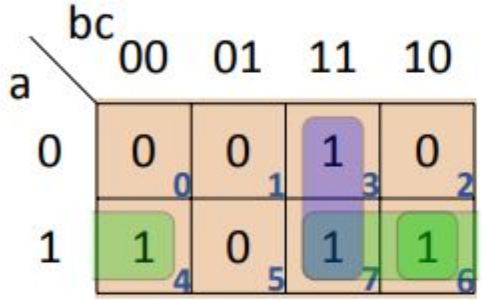
- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour



- Three prime implicants

K-Map Example

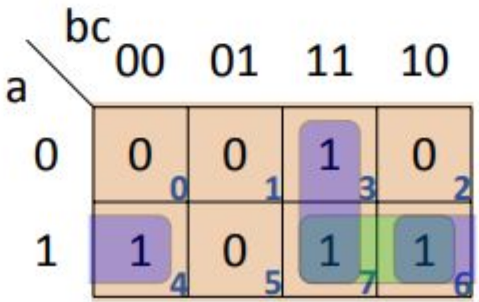
- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour



- Three prime implicants

K-Map Example

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour



- Three prime implicants
- Two essential prime implicants

K-Map Example

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour
- Three prime implicants
- Two essential prime implicants

a \ bc	00	01	11	10
0	0 (0)	0 (1)	1 (3)	0 (2)
1	1 (4)	0 (5)	1 (7)	1 (6)

K-Map Example

PES UNIVERSITY

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour

a \ bc	00	01	11	10
0	0_{0}	0_{1}	1_{3}	0_{2}
1	1_{4}	0_{5}	1_{7}	1_{6}

- Three prime implicants
- Two essential prime implicants
- Required prime implicants: bc, $a\bar{c}$

K-Map Example

- Prime implicants: green colour
- Essential prime implicants: blue colour
- Required prime implicants: dark red colour

a \ bc	00	01	11	10
0	0_{0}	0_{1}	1_{3}	0_{2}
1	1_{4}	0_{5}	1_{7}	1_{6}

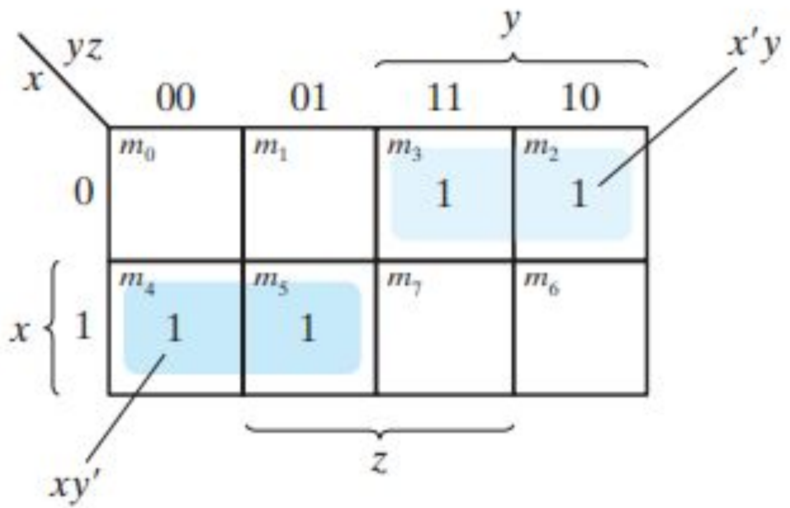
- Three prime implicants
- Two essential prime implicants
- Required prime implicants: bc, $a\bar{c}$
- Boolean formula:

$$f(a, b, c) = a\bar{c} + bc$$

Three Variable K-Map

Simplify the Boolean function

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



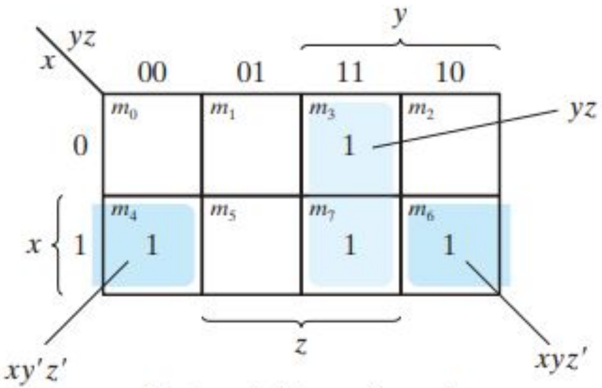
$$F = x'y + xy'$$

Three Variable K-Map

Simplify the Boolean function

$$F(x, y, z) = \Sigma(3, 4, 6, 7)$$

$$F = yz + xz'$$



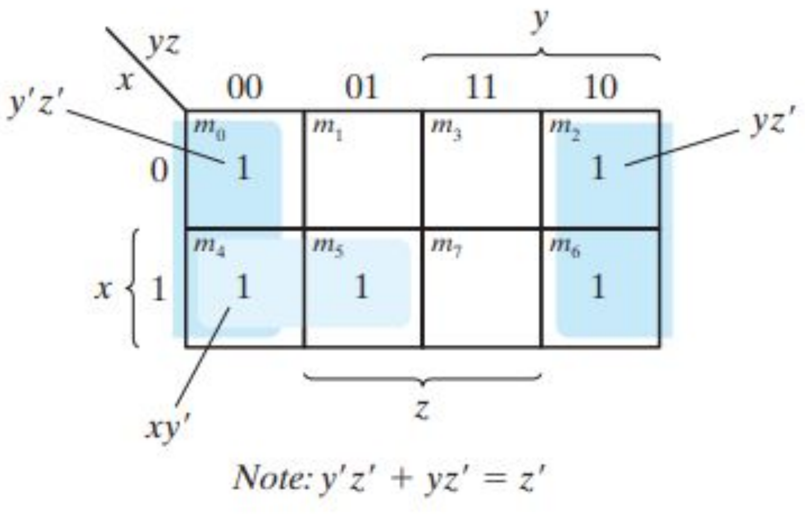
Note: $xy'z' + xyz' = xz'$

Three Variable K-Map

Simplify the Boolean function

$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$

$$F = z' + xy'$$



Quick Pointers

- The number of adjacent squares that may be combined must always represent a number that is a power of two, such as 1, 2, 4, and 8.
- As more adjacent squares are combined, we obtain a product term with fewer literals.
- One square represents one minterm, giving a term with three literals.
- Two adjacent squares represent a term with two literals.
- Four adjacent squares represent a term with one literal.
- Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

Example

For the Boolean function

$$F = A'C + A'B + AB'C + BC$$

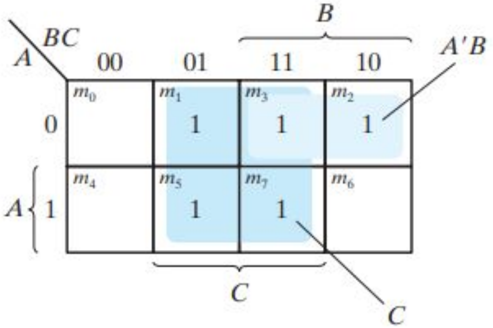
(a) Express this function as a sum of minterms.
(b) Find the minimal sum-of-products expression.

sum-of-minterms form as

$$F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$$

The sum-of-products expression, as originally given, has too many terms. It can be simplified, as shown in the map, to an expression with only two terms:

$$F = C + A'B$$



Four Variable K-Map

- The map minimization of four-variable Boolean functions is similar to the method used to minimize three-variable functions. Adjacent squares are defined to be squares next to each other.

- For example, m0 and m2 form adjacent squares, as do m3 and m11.

- The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four-variable map:

Four Variable K-Map

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)

$wx \backslash yz$	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

(b)

Quick Pointers

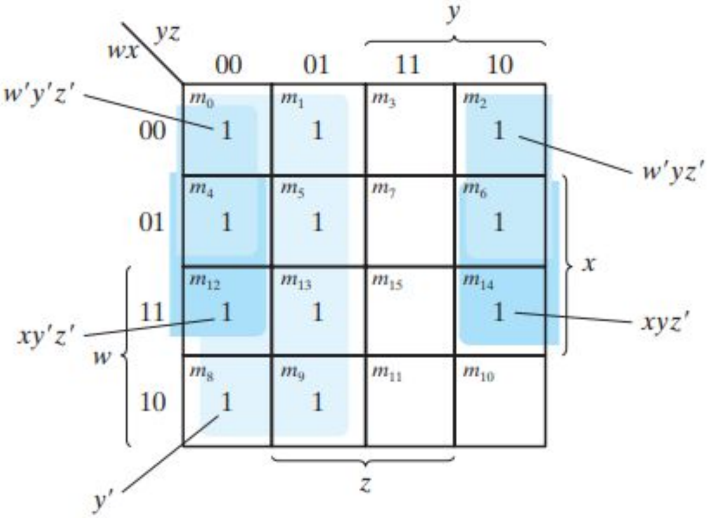
- One square represents one minterm, giving a term with four literals.
- Two adjacent squares represent a term with three literals.
- Four adjacent squares represent a term with two literals.
- Eight adjacent squares represent a term with one literal.
- Sixteen adjacent squares produce a function that is always equal to 1.

Example

Simplify the Boolean function

$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

$$F = y' + w'z' + xz'$$



Note: $w'y'z' + w'yz' = w'z'$

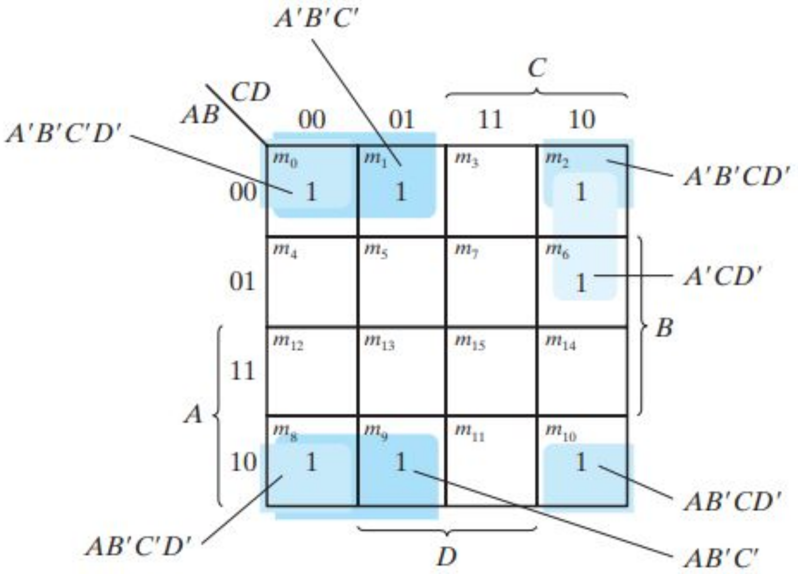
$xy'z' + xyz' = xz'$

Example

Simplify the Boolean function

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

Example



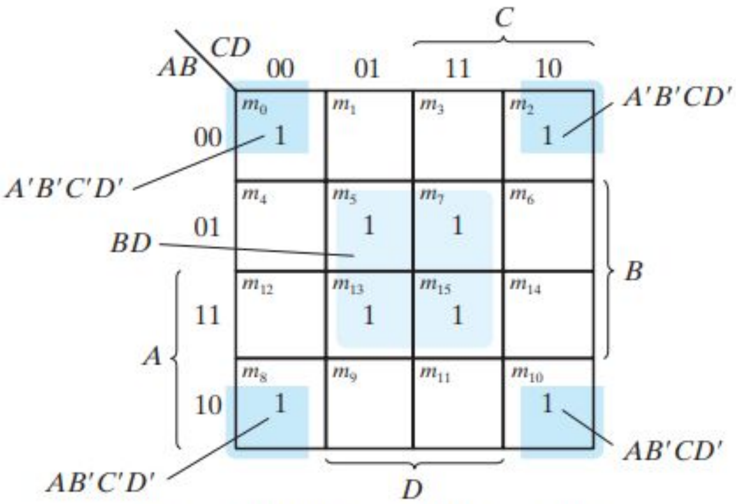
Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

FIGURE 3.10
Map for Example 3.6, $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

Simplification using Prime Implicants

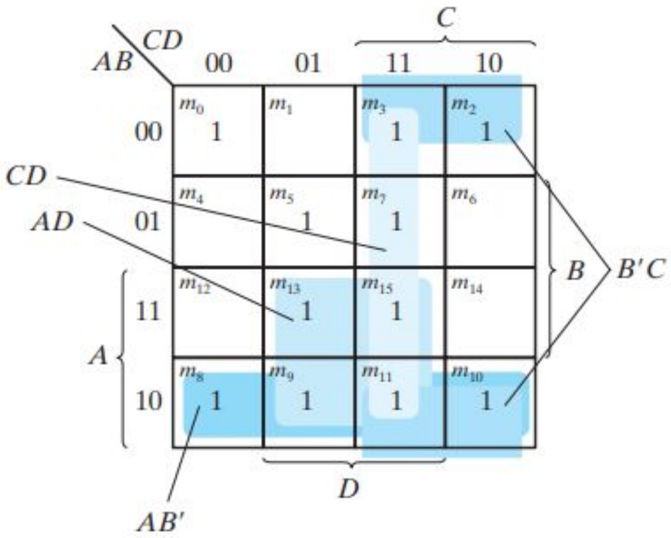
(a) $F(w, x, y, z) = \Sigma(0, 2, 5, 7, 8, 10, 13, 15)$

(b) $F(w, x, y, z) = \Sigma(2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$

(a) Essential prime implicants BD and $B'D'$



(b) Prime implicants CD, $B'C$, AD, and AB'

Another K-Map Example

K-Map Minimization Example

a	b	y
0	0	1
0	1	0
1	0	1
1	1	1

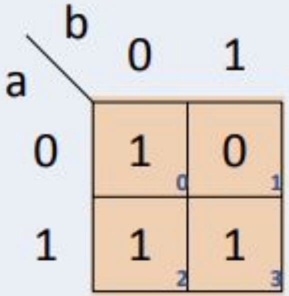
Two Input Truth Table

Another K-Map Example

K-Map Example (two inputs)

a	*b*	*y*
0	0	1
0	1	0
1	0	1
1	1	1

Two Input Truth Table



Another K-Map Example

K-Map Example (two inputs)

a	*b*	*y*
0	0	1
0	1	0
1	0	1
1	1	1

Two Input Truth Table

a \ b	0	1
0	1	0
1	1	1

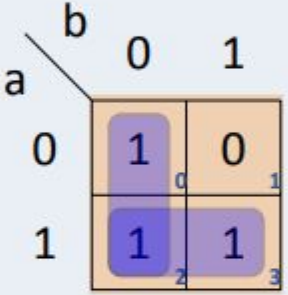
- Two prime implicants

Another K-Map Example

K-Map Example (two inputs)

a	*b*	*y*
0	0	1
0	1	0
1	0	1
1	1	1

Two Input Truth Table



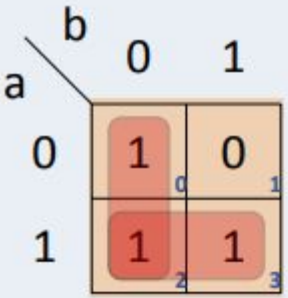
- Two prime implicants
- Two essential prime implicants

Another K-Map Example

K-Map Example (two inputs)

a	*b*	*y*
0	0	1
0	1	0
1	0	1
1	1	1

Two Input Truth Table



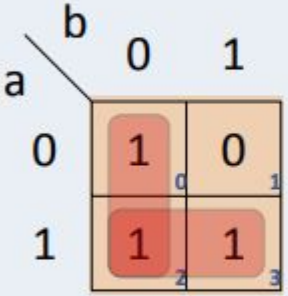
- Two prime implicants
- Two essential prime implicants
- Two required prime implicants

Another K-Map Example

K-Map Example (two inputs)

a	b	y
0	0	1
0	1	0
1	0	1
1	1	1

Two Input Truth Table



- Two prime implicants
- Two essential prime implicants
- Two required prime implicants
- Minimized Boolean formula: $f(a, b, c, d) = \overline{b} + a$

Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

Another K-Map Example

K-Map Minimization Example

a	*b*	*c*	*y*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0_{0}	1_{1}	1_{3}	0_{2}
1	1_{4}	1_{5}	1_{7}	1_{6}

Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0 $_{0}$	1 $_{1}$	1 $_{3}$	0 $_{2}$
1	1 $_{4}$	1 $_{5}$	1 $_{7}$	1 $_{6}$

- Two prime implicants

Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0_0	1_1	1_3	0_2
1	1_4	1_5	1_7	1_6

- Two prime implicants
- Two essential prime implicants

Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0_{0}	1_{1}	1_{3}	0_{2}
1	1_{4}	1_{5}	1_{7}	1_{6}

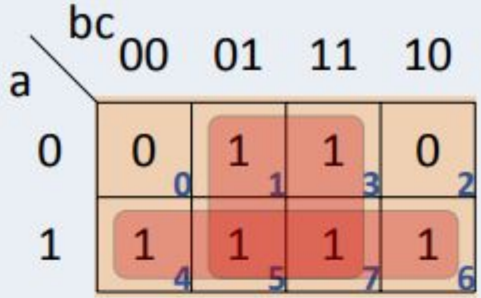
- Two prime implicants
- Two essential prime implicants
- Two required prime implicants: *c*, *a*

Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table



- Two prime implicants
- Two essential prime implicants
- Two required prime implicants: c, a
- Minimal Boolean formula: $f(a, b, c) = c + a$

Yet Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

Yet Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

Yet Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

- Four prime implicants

Yet Another K-Map Example

K-Map Minimization Example

a	*b*	*c*	*y*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

- Four prime implicants
- Two essential prime implicants

Yet Another K-Map Example

K-Map Minimization Example

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

- Four prime implicants
- Two essential prime implicants
- Three required prime implicants: $\overline{b}c$, $b\overline{c}$ along with either ca or ab
- Minimal Boolean formulas:
  $f(a,b,c) = \overline{b}c + b\overline{c} + ca$
  $f(a,b,c) = \overline{b}c + b\overline{c} + ab$

Yet Another K-Map Example

K-Map Minimization Example

a	b	c	d	y
0	0	0	0	1
0	0	0	0	0
0	0	1	0	1
0	0	1	0	0
0	1	0	0	0
0	1	0	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	0
1	1	0	1	0
1	1	0	1	1
1	1	1	1	0
1	1	1	1	1

Four Input Truth Table

Yet Another K-Map Example

K-Map Example (four inputs)

a	b	c	d	y
0	0	0	0	1
0	0	0	0	0
0	0	1	0	1
0	0	1	0	0
0	1	0	0	0
0	1	0	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	0
1	1	0	1	0
1	1	0	1	1
1	1	1	1	0
1	1	1	1	1

Four Input Truth Table

ab \ cd	00	01	11	10
00	1 (0)	0 (1)	0 (3)	1 (2)
01	0 (4)	1 (5)	1 (7)	0 (6)
11	0 (12)	1 (13)	1 (15)	0 (14)
10	1 (8)	0 (9)	0 (11)	1 (10)

Yet Another K-Map Example

K-Map Example (four inputs)

a	b	c	d	y
0	0	0	0	1
0	0	0	0	0
0	0	1	0	1
0	0	1	0	0
0	1	0	0	0
0	1	0	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	0
1	1	0	1	0
1	1	0	1	1
1	1	1	1	0
1	1	1	1	1

Four Input Truth Table

ab \ cd	00	01	11	10
00	1_{0}	0_{1}	0_{3}	1_{2}
01	0_{4}	1_{5}	1_{7}	0_{6}
11	0_{12}	1_{13}	1_{15}	0_{14}
10	1_{8}	0_{9}	0_{11}	1_{10}

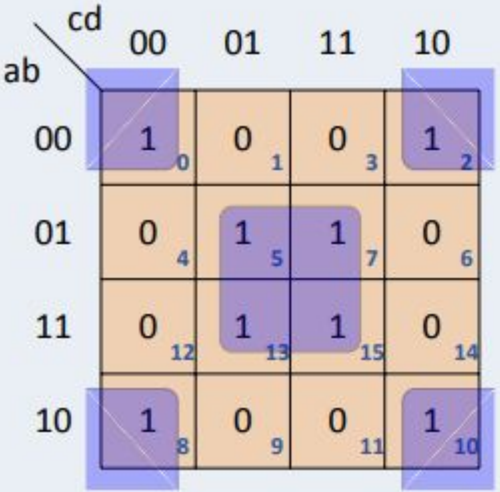
- Two prime implicants

Yet Another K-Map Example

K-Map Example (four inputs)

a	b	c	d	y
0	0	0	0	1
0	0	0	0	0
0	0	1	0	1
0	0	1	0	0
0	1	0	0	0
0	1	0	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	0
1	1	0	1	0
1	1	0	1	1
1	1	1	1	0
1	1	1	1	1

Four Input Truth Table



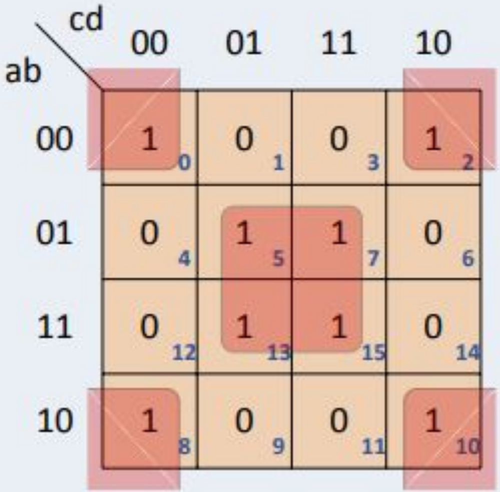
- Two prime implicants
- Two essential prime implicants

Yet Another K-Map Example

K-Map Example (four inputs)

a	b	c	d	y
0	0	0	0	1
0	0	0	0	0
0	0	1	0	1
0	0	1	0	0
0	1	0	0	0
0	1	0	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	0
1	1	0	1	0
1	1	0	1	1
1	1	1	1	0
1	1	1	1	1

Four Input Truth Table



- Two prime implicants
- Two essential prime implicants
- Two required prime implicants: bd, $\overline{bd}$

Yet Another K-Map Example

K-Map Example (four inputs)

a	*b*	*c*	*d*	*y*
0	0	0	0	1
0	0	0	0	0
0	0	1	0	1
0	0	1	0	0
0	1	0	0	0
0	1	0	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	0
1	1	0	1	0
1	1	0	1	1
1	1	1	1	0
1	1	1	1	1

Four Input Truth Table

ab \ cd	00	01	11	10
00	1 (0)	0 (1)	0 (3)	1 (2)
01	0 (4)	1 (5)	1 (7)	0 (6)
11	0 (12)	1 (13)	1 (15)	0 (14)
10	1 (8)	0 (9)	0 (11)	1 (10)

- Two prime implicants
- Two essential prime implicants
- Two required prime implicants: bd, $\overline{bd}$
- Minimized Boolean formula: $f(a, b, c, d) = bd + \overline{b}\,\overline{d}$

Gate-Level Minimization and Combinational logic Don’t-care conditions.

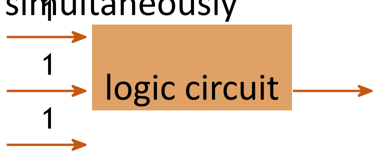
- Functions that have **unspecified outputs** for some input combinations are called incompletely specified functions .
- In most applications, we simply don’t care what value is assumed by the function for the unspecified minterms.
- For this reason, it is customary to call the unspecified minterms of a function don’t-care conditions

.

K-MAPS

Don’t Cares

- Suppose we are asked to design a three input logic circuit all whose inputs can never be 1 simultaneously



guaranteed can’t happen

a	*b*	*c*	*y*
0	0	0	1
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	*X*

Truth table

a \ bc	00	01	11	10
0	1	1	0	0
1	1	1	X	1

- Case 0: $y = \bar{b} + \bar{c}a$
- Case 1: $y = \bar{b} + a$
- Either 0 or 1 can result in smaller formula
- So initially write X in truth table and K-Map

Don’t Care

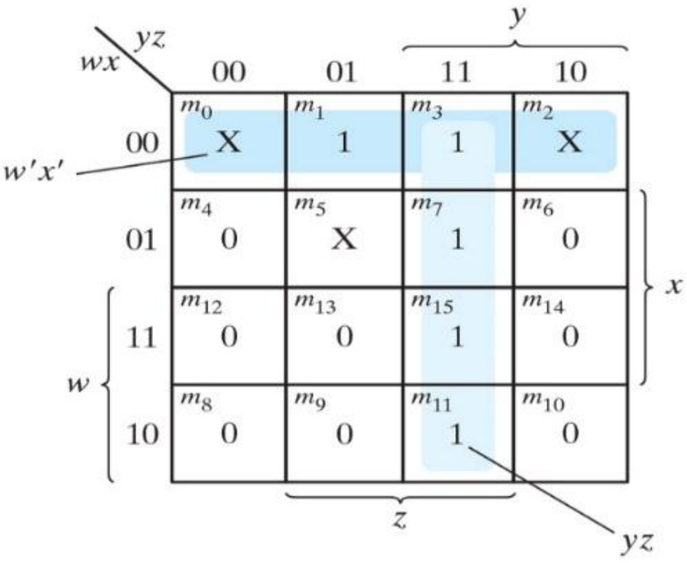
Output(s) we **don’t care** about are denoted by X , can be treated as either 0 or 1

- Don’t Care / X has other use cases, discussed later

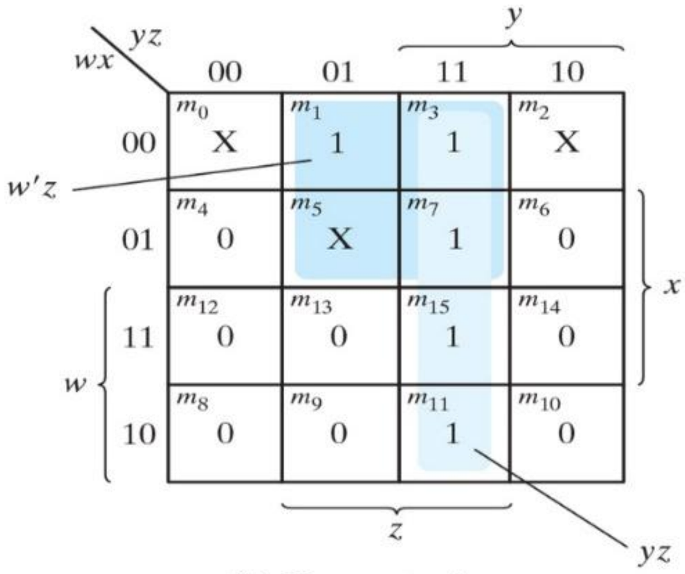
Gate-Level Minimization and Combinational logic Don’t-care conditions.

(a)F(w, x, y, z)=∑m(1,3,7,11,15) + d(1,2,5)

(b)F(w, x, y, z)=∑m(1,3,7,11,15) + d(0,2,5)



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

.

Gate-Level Minimization and Combinational logic Product of sums simplification.

- The minimized Boolean functions derived from the map in all previous examples were expressed in sum-of-products form.
- With a minor modification, the product-of-sums form can be obtained.
- The procedure for obtaining a minimized function in product-of-sums form follows from the basic properties of Boolean functions.

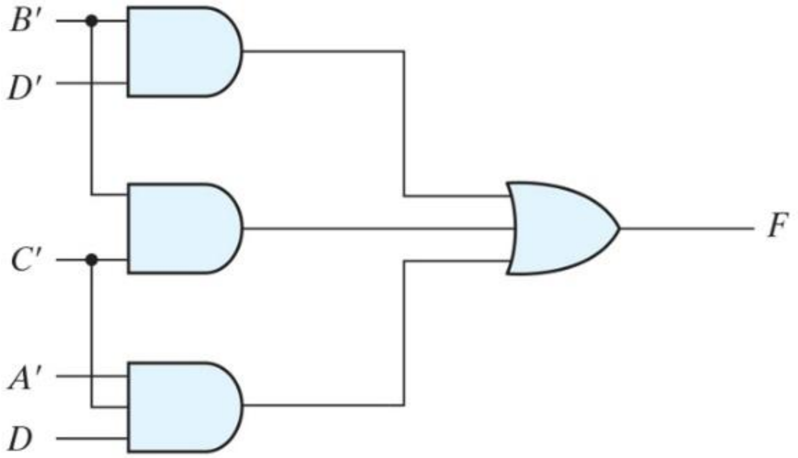
Gate-Level Minimization and Combinational logic Product of sums simplification.

- The 1's placed in the squares of the map represent the minterms of the function.
- The minterms not included in the standard sum-of-products form of a function denote the complement of the function.
- From this observation, we see that the complement of a function is represented in the map by the squares not marked by 1's
- If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified sum-of-products expression of the complement of the function (i.e., of F).
- The complement of F gives us back the function F in product-of-sums form (a consequence of DeMorgan's theorem)

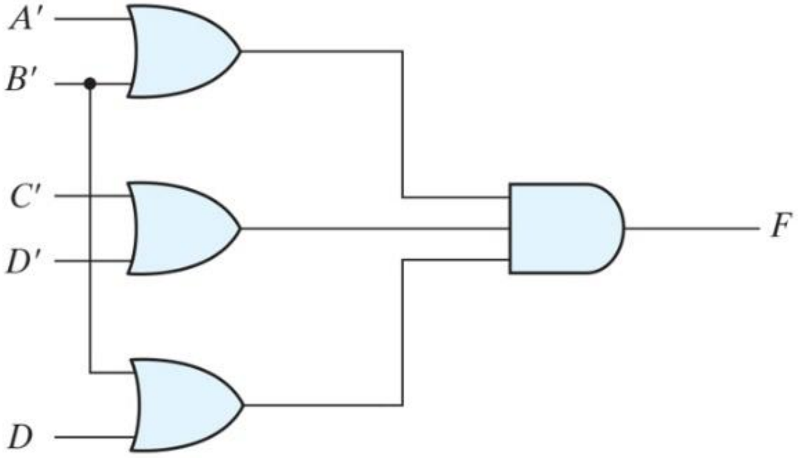
Gate-Level Minimization and Combinational logic

Product of sums simplification.

Gate implementations of the function



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



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

SOP POS

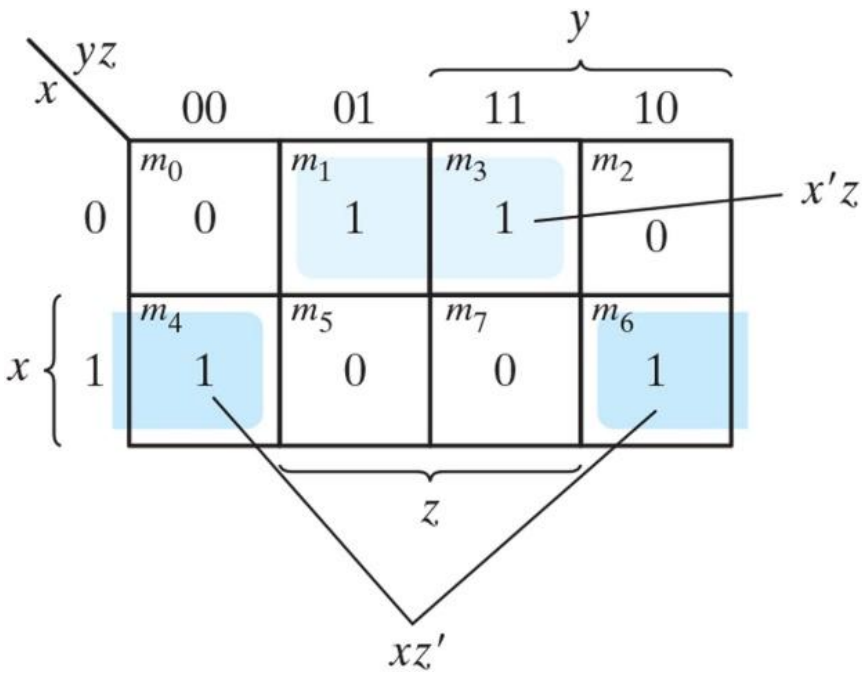
Gate-Level Minimization and Combinational logic
Product of sums simplification.

Truth Table of Function F(x, y, z) =∑m(1,3,4,6).

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

Gate-Level Minimization and Combinational logic

Product of sums simplification.



Map for the function **F(x, y, z)=∑m(1,3,4,6).**

Real-Life Applications of K-Maps

Digital Circuits: Karnaugh maps are widely used in the design of digital circuits. The simplified expressions obtained from K-Maps can be easily translated into logic gates, making it easier to design and implement the circuit.
Computer memory: K-Maps are used in the design of computer memory. The simplified expressions obtained from K-Maps help in reducing the size and complexity of the memory circuit.
Communication systems: K-Maps are used in the design of communication systems. The simplified expressions obtained from K-Maps help in reducing the complexity and improving the efficiency of the communication system.

Real-Life Applications of K-Maps

Consumer electronics: K-Maps are used in the design of consumer electronics such as televisions, radios, and other electronic devices. The simplified expressions obtained from K-Maps help in reducing the size and complexity of electronic devices.

Automotive electronics: K-Maps are used in the design of automotive electronics such as engine control units, braking systems, and other electronic systems. The simplified expressions obtained from K-Maps help in reducing the size and complexity of the electronic systems, making them more efficient and reliable.

Used in error detection like Parity Checkers, Cyclic Redundancy Check (CRC) etc.

Try it Out

Minimize using K-Maps

- $f(a, b, c) = \Sigma(0, 3, 5)$
- $f(a, b, c) = \Sigma(0, 1, 2, 4, 5, 6)$
- $f(a, b, c) = \Sigma(0, 2, 3, 4, 6, 7)$
- $f(a, b) = \Sigma(0, 1, 2, 3)$
- $f(a, b, c, d) = \Sigma(0, 1, 5, 7, 15, 14, 10)$
- $g(w,x,y,z) = \Sigma(1,3,7,11,15)+d(0,2,5)$
- $s(w,x,y,z) = \Sigma(0,1,2,4,5,7,9)+d(03,8,15)$

THANK YOU

Team DDCO
Department of Computer Science and Engineering