Karnaugh Map (K-Map) Outline

- SOP and POS Forms
- Terminology
- Circuit Optimization
  - Literal cost
  - Gate input cost
- Karnaugh Maps
- 4-variable Examples
- K-Map with don't care
- K-Map POS forms
- 5-variable Examples
- K-Map : multiple-output cases

minterms and Maxterms

- Minterm
  - A product term which contains each of the *n* variables as factors in either complemented or uncomplemented form is called a *minterm*
  - Example for 3 variables: ab'c is a *minterm; ab' is not*
- Maxterm
  - A sum term which contains each of the *n variables* as factors in either complemented or uncomplemented form is called a *maxterm*
  - For 3 variables: a'+b+c' is a *maxterm; a'+b is not*

Minterms and Maxterms

- Examples
  - Three-variable example:

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1x_2x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1x_2x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Note that $(m_i)' = M_i$ *and* $(M_i)' = m_i$

Sum-of-Products (SOP) Form

- Canonical Sum-of-Products (or Disjunctive Normal) Form
  - *The sum of all minterms derived from those rows for which the value of the function is 1 takes on the value 1 or 0 according to the value assumed by f. Therefore this sum is in fact an algebraic representation of f. An expression of this type is called a canonical sum of products, or a disjunctive normal expression.*

SOP Example

- $f = ab'c + a'b + a'bc' + b'c'$

		a	b	c	f
m_0	M_0	0	0	0	1 = a_0
m_1	M_1	0	0	1	0 = a_1
m_2	M_2	0	1	0	1 = a_2
m_3	M_3	0	1	1	1 = a_3
m_4	M_4	1	0	0	1 = a_4
m_5	M_5	1	0	1	1 = a_5
m_6	M_6	1	1	0	0 = a_6
m_7	M_7	1	1	1	0 = a_7

SOP Form

General Form : $$f = \sum_{k=0}^{N} a_k m_k$$

Example: *f = ab'c + a'b + a'bc' + b'c'*

$$f = 1m_0 + 0m_1 + 1m_2 + 1m_3 +$$
$$1m_4 + 1m_5 + 0m_6 + 0m_7$$
$$= m_0 + m_2 + m_3 + m_4 + m_5$$
$$= \sum m(0,2,3,4,5)$$

Product-of-Sums (POS) Form

- Canonical Product-of-Sums (or Conjunctive Normal) Form
  - *An expression formed of the product of all maxterms for which the function takes on the value 0 is called a canonical product of sums, or a conjunctive normal expression.*

POS Form

General Form : $f = \prod_{k=0}^{N} (a_k + M_k)$

Example: $f = ab'c + a'b + a'bc' + b'c'$

$$f = (1 + M_0)(0 + M_1)(1 + M_2)(1 + M_3)$$
$$\times (1 + M_4)(1 + M_5)(0 + M_6)(0 + M_7)$$
$$= M_1 M_6 M_7$$
$$= \prod M(1,6,7)$$

Terminology

- Literal : a variable, either uncomplemented or complemented
- Implicant : A product term that indicates the input valuation(s) for which a given function is equal to 1, e.g.,

$f = a'b'c' + a'bc' + a'b'c + a'bc + abc$

Implicants:

5 minterms: $a'b'c'$, $a'bc'$, $a'b'c$, $a'bc$, abc

Combined minterms: $a'b'$, $a'b$, $a'c'$, $a'c$, bc, a'

Terminology

- **Prime implicant** : An implicant is called a <u>*prime implicant*</u> *if it cannot be combined into another implicant* that has fewer literals, e.g., *a', bc in previous slide.*
- ***Essential*** *prime implicant* :*a prime implicant that includes at* least one 1 that is not included in any other prime implicant.
- **Cover** : A collection of implicants that account for all valuations for which a given function is equal to 1.

A set of all minterms, a set of all prime implicants

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
- Cost criteria:
  - 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 + AB'C + AC'D' (L=8)
  - F = BD + AB'C + AB'D' + ABC' (L=11)
  - F = (A + B)(A + D)(B + C + D')(B' + C' + D) (L=10)
  - Which 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 found from the equation(s) by finding the sum of:
  - all literal appearances
  - the number of terms excluding terms consisting only of a single literal,(G) and
  - optionally, the number of distinct complemented single literals (GN).
- Example:
  - F = BD + AB'C + AC'D' — G=11,GN=14
  - F = BD + AB'C + AB'D' + ABC' — G=15,GN=18
  - F = (A + B)(A + D)(B + C + D')(B' + C' + D) — G=14,GN=17
  - Which solution is best?

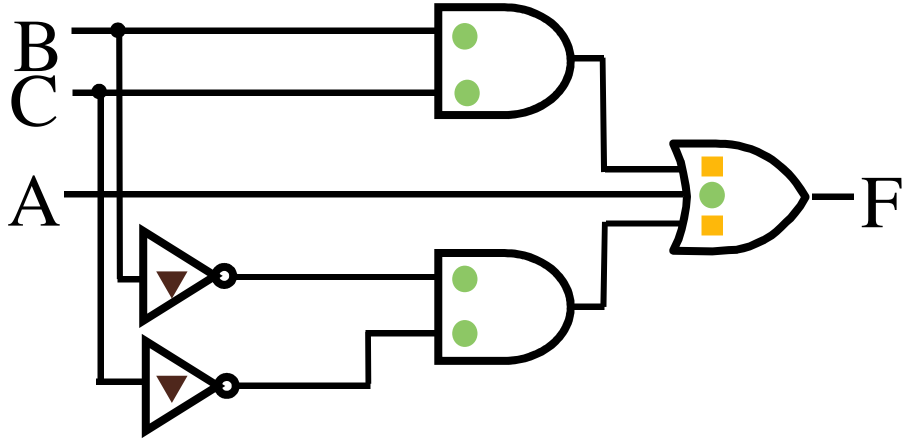
Cost Criteria (continued)

- Example 1:
- F= A + B C + B'C'

GN = G + 2 = 9

L = 5

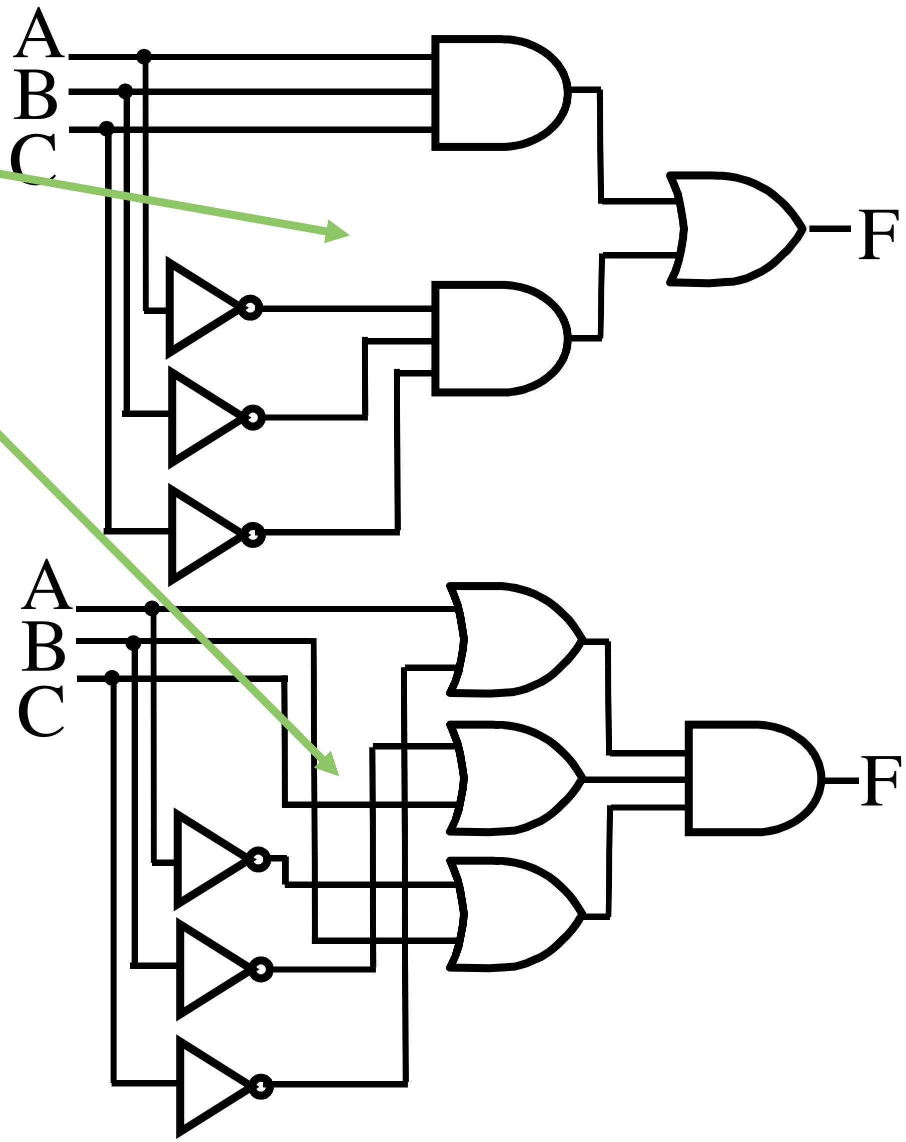
G = L + 2 = 7



- 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

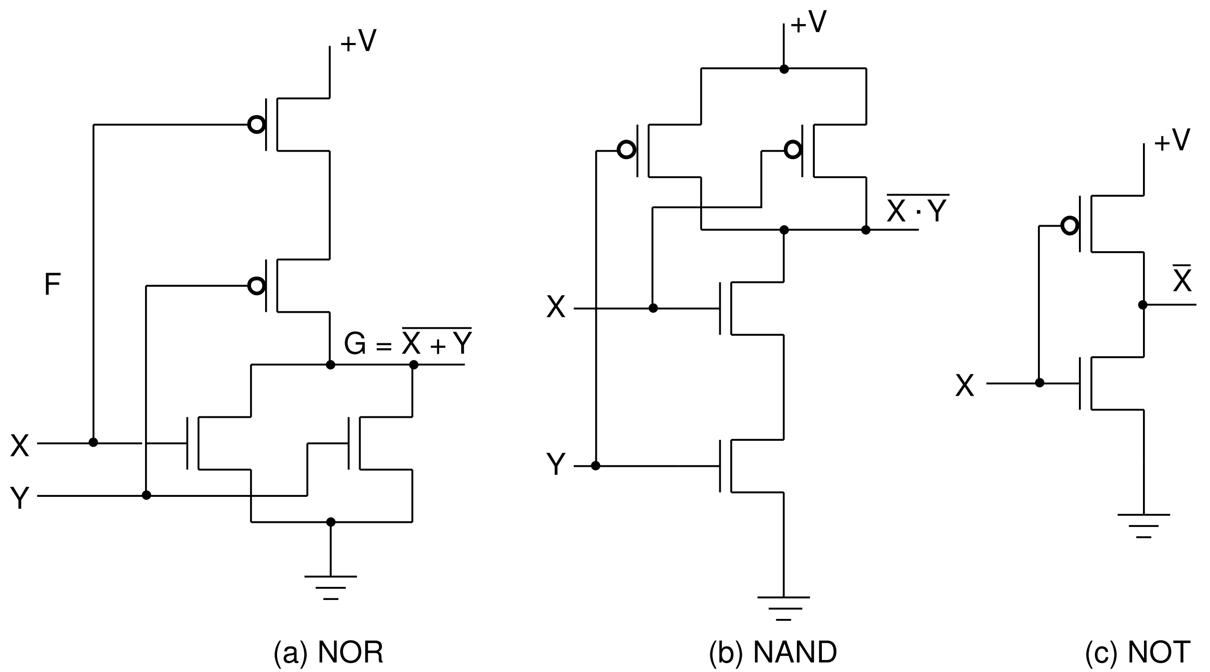
Cost Criteria (continued)

- Example 2:
- F = ABC + A'B'C'
- L = 6 G = 8 GN = 11
- F = (A+C')(B'+C)(A'+B)
- L = 6 G = 9 GN = 12
- Same function and same literal cost
- But first circuit has better gate input count and better gate input count with NOTs
- Select it!



Why Use Gate Input Counts?

- **CMOS logic gates:**



- **Each input adds:**
  - **P-type transistor to pull-up network**
  - **N-type transistor to pull-down network**

Boolean Function Optimization

- **Minimizing the gate inputs reduces circuit cost.**
- **Some important questions:**
  - **When do we stop trying to reduce the cost?**
  - **Do we know when we have a minimum cost?**
- **Two-level SOP & POS optimum or near-optimum functions**
- **Karnaugh maps (K-maps)**
  - **Graphical technique useful for up to 5 or 6 inputs**

Minimization Procedure

1. Generate all prime implicants.
2. Find the set "essential" prime implicants.
3. If this set covers all 1's, then it is the desired cover. If not, determine other prime implicants needed to form a complete minimum-cost cover.

Two Variable K-Maps

	y = 0	y = 1
x = 0	$m_0 =$ $x'y'$	$m_1 =$ $x'y$
x = 1	$m_2 =$ xy'	$m_3 =$ xy

- A 2-variable Karnaugh Map:
  - Similar to Gray Code
  - Adjacent minterms differ by one variable

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

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

Some Uses of K-Maps

- Finding optimum or near optimum
  - SOP and POS standard forms, and
  - two-level AND/OR and OR/AND circuit implementations

  for functions with small numbers of variables
- Demonstrate concepts used by computer-aided design programs to simplify large circuits

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 as:

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

K-Map Function Representation

- Example: $G(x,y) = x + y$

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

- For $G(x,y)$, two pairs of adjacent cells containing 1's can be combined as:

Duplicate xy

$$G(x,y) = x'y + xy' + xy = xy' + xy + x'y + xy$$
$$= x(y'+y) + y(x'+x) = x+y$$

Three Variable Maps

- A three-variable K-map:

	xy=00	xy=01	xy=11	xy=10
z=0	m_0	m_2	m_6	m_4
z=1	m_1	m_3	m_7	m_5

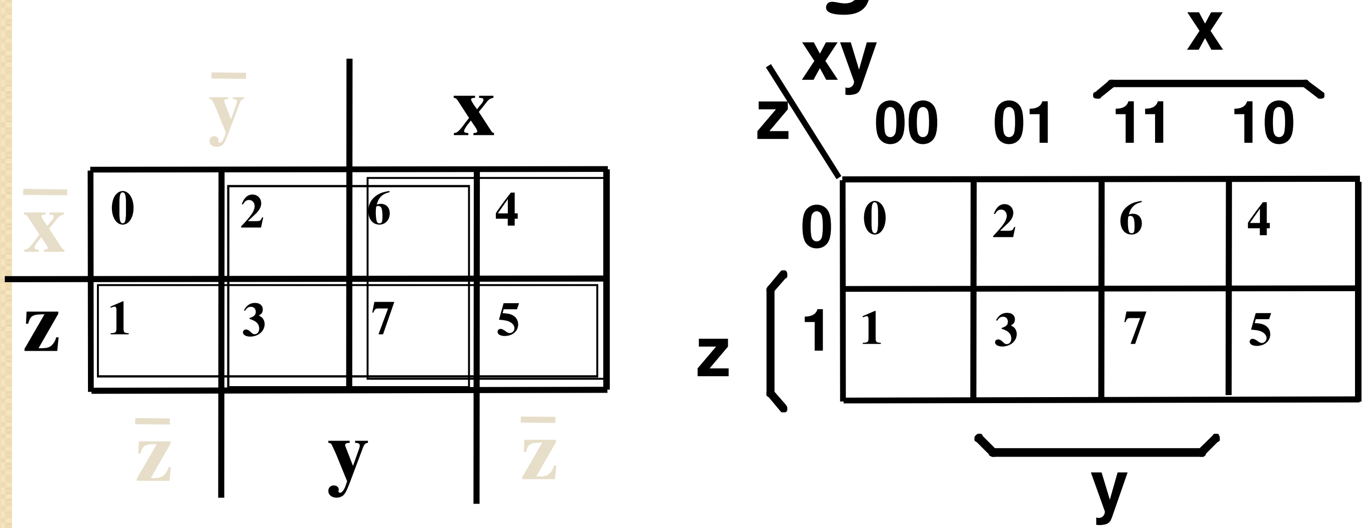
- Where each minterm corresponds to the product terms:

	xy=00	xy=01	xy=11	xy=10
z=0	x'y'z'	x'yz'	xyz'	xy'z'
z=1	x'y'z	x'yz	xyz	xy'z

- Note that if the binary value for an <u>index</u> 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 Functions

- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of F' blank
- Example:

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

			x	x
	0	2 1	6	4 1
z	1	3 1	7	5 1
		y	y	

- Example:

$$G(x,y,z)=\sum m(3,4,6,7)$$

			x	x
	0	2	6 1	4 1
z	1	3 1	7 1	5
		y	y	

- <u>Learn</u> the locations of the 8 indices based on the variable order shown (x, most significant and z, least significant) on the map boundaries

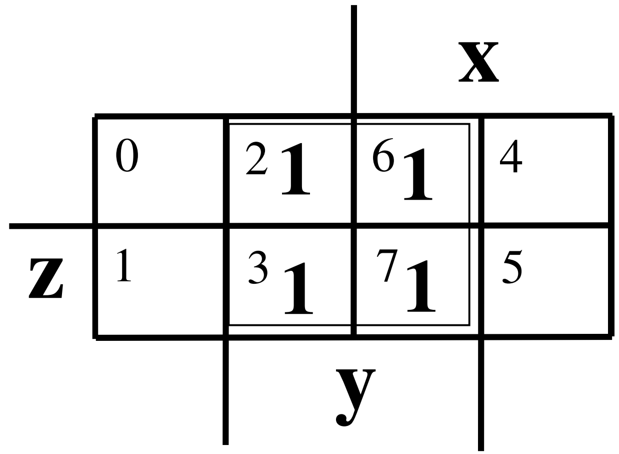
Combining Squares

- 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 product term with two variables
  - Four "adjacent" terms represent a product term with one variable
  - Eight "adjacent" terms is the function of all ones (no variables) = 1.

Example: Combining Squares

- **Example: Let**

$$F = \sum m(2,3,6,7)$$



- **Using the Boolean algebra operations:**

$$F = x'yz' + x'yz + xyz' + xyz$$

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

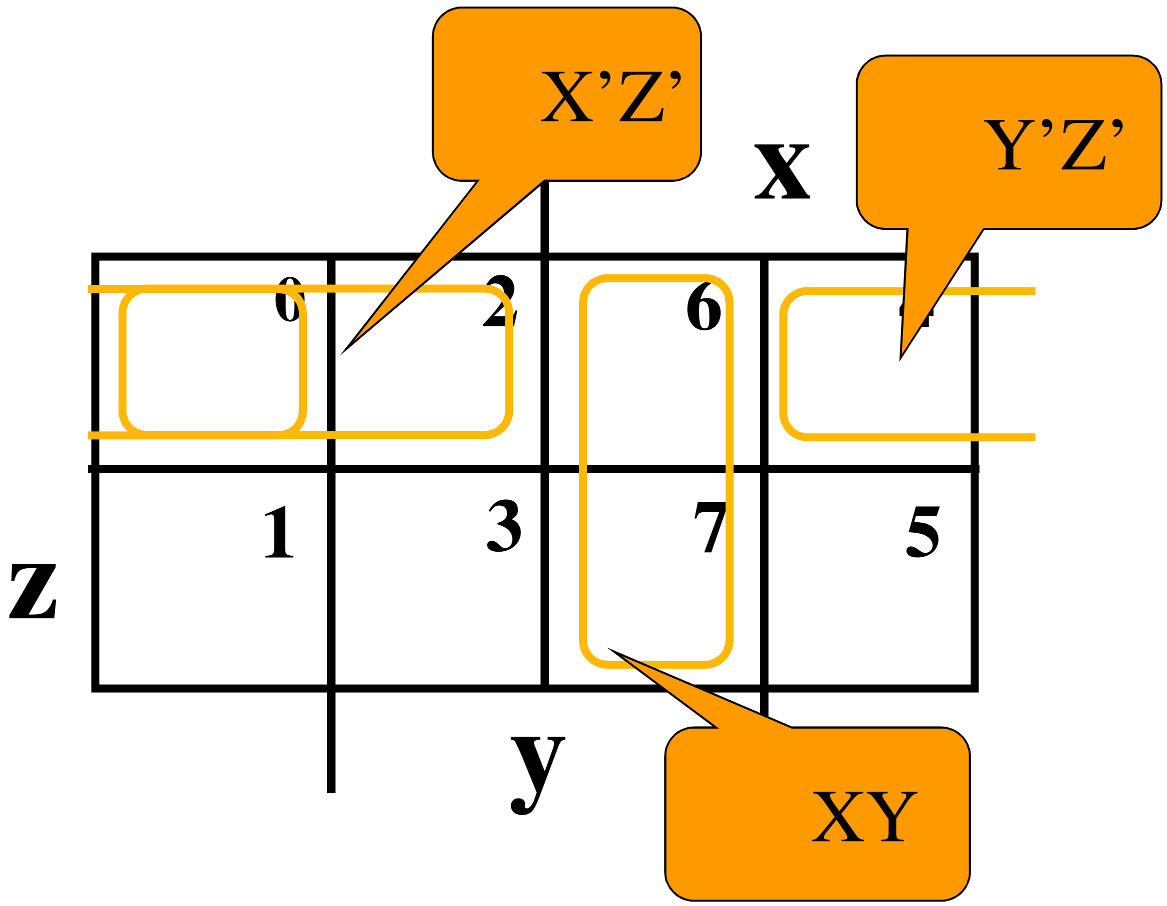
- **Thus the four terms that form a 2 × 2 square correspond to the term "y".**

Three-Variable Maps

- Reduced literal product terms for SOP standard forms correspond to <u>rectangles</u> on 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.
- Rectangles can contain non-adjacent cells due to wrap-around at edges

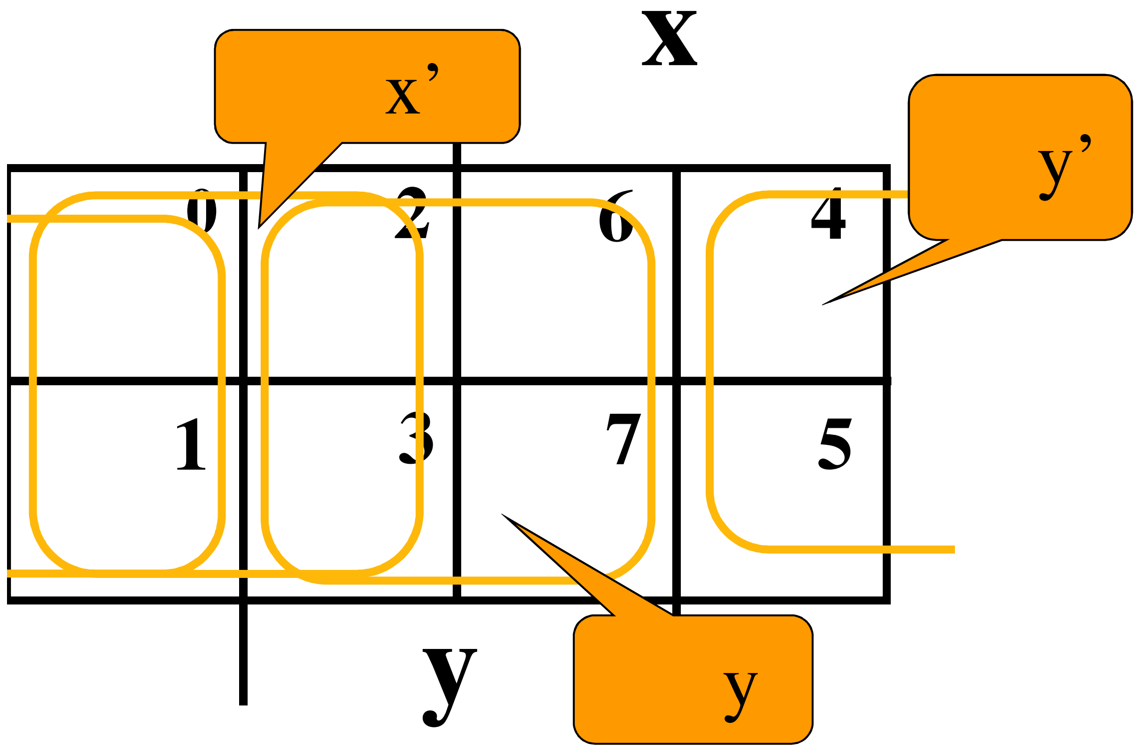
Three-Variable Maps

- **Example Shapes of 2-cell Rectangles:**



Three-Variable Maps

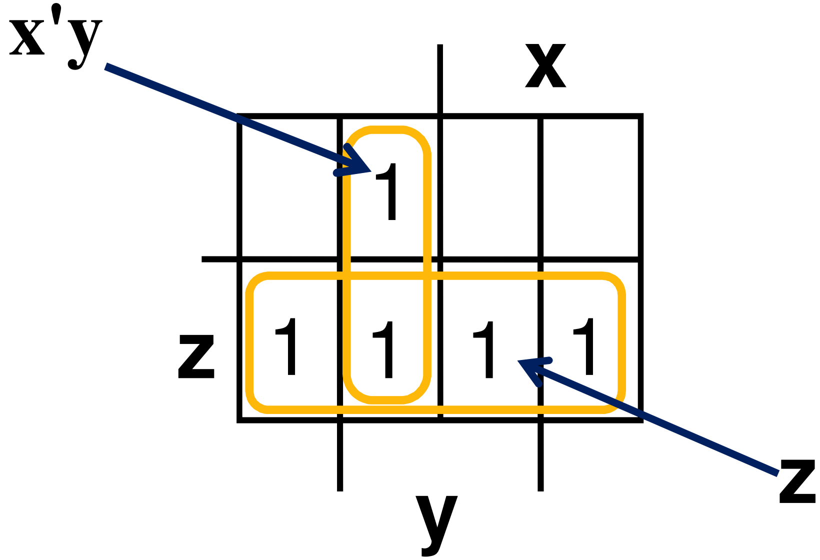
- Example Shapes of 4-cell Rectangles:



- Read off the product terms for the rectangles shown

Three Variable Maps

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



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

Karnaugh Map Method I

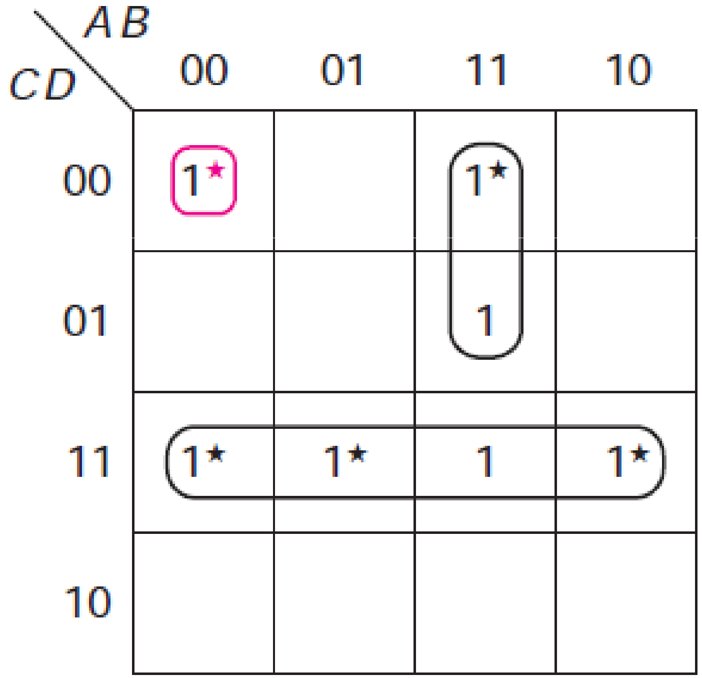
1. Find all essential implicants, circle them, and mark the minterm(s) that makes them essential with *.
2. Find enough other prime implicants to cover the function using 2 criteria:
   1. Choose a prime implicant that covers as many 1's as possible.
   2. Avoid leaving uncovered 1's isolated.

Karnaugh Map Method 2

1. Circle all prime implicants.
2. Select all essential prime implicants; they are easily identified by finding 1's that have only been circled once.
3. Then choose enough of the other prime implicants to cover all 1's.

K-map Example 1

$$f(A,B,C,D)=\sum m(0,3,7,11,12,13,15)$$

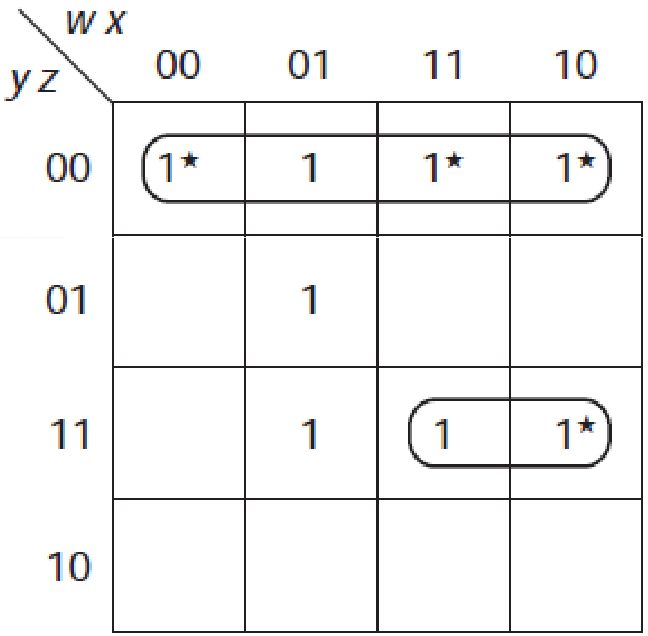


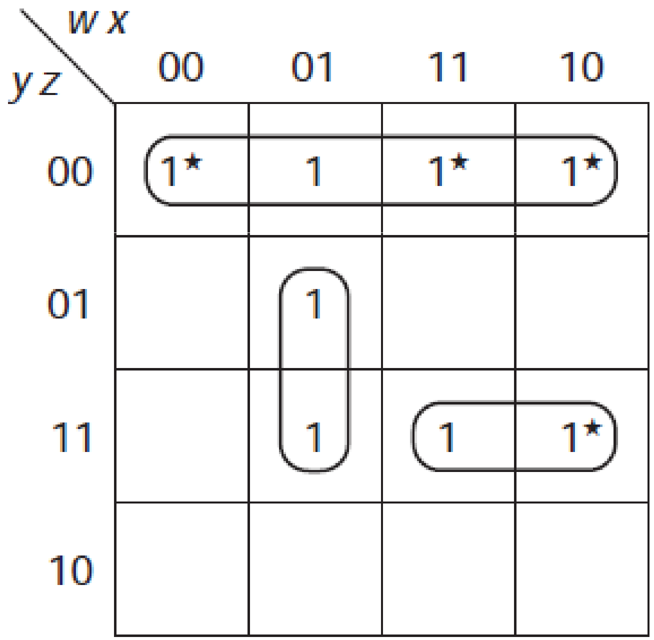
$$f = CD + ABC' + A'B'C'D'$$

From Marcovitz’s Introduction to Logic Design

K-map Example 2

$$f(w,x,y,z)=\sum m(0,4,5,7,8,11,12,15)$$





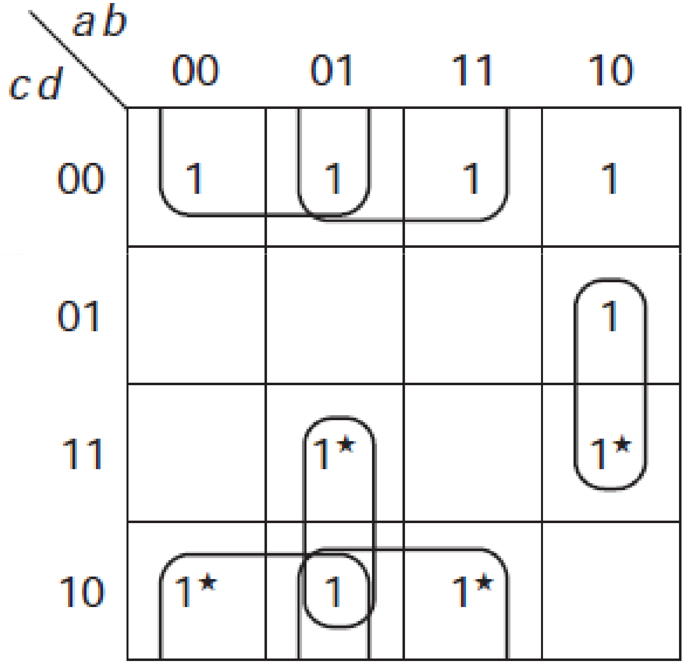
$$f = y'z' + wyz + w'xz$$

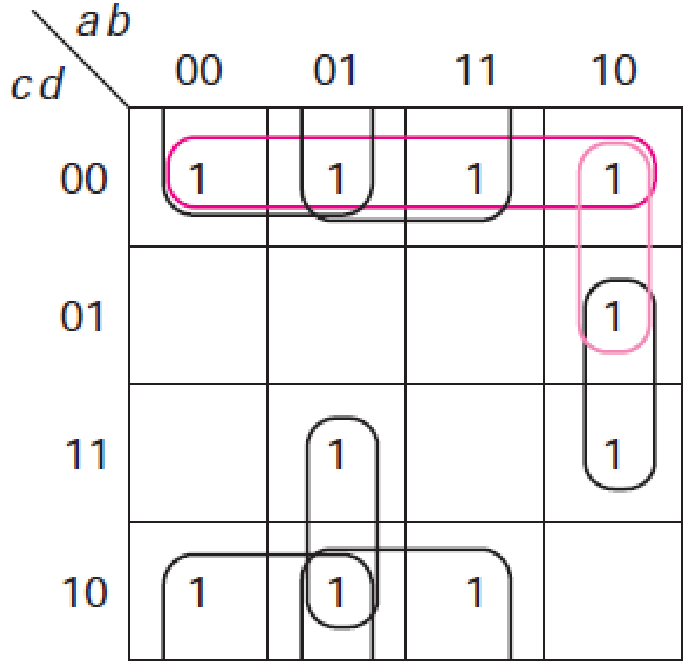
Note: *w'xy', xyz are redundant*

From Marcovitz's Introduction to Logic Design

K-map Example 3

$$f(a,b,c,d) = \sum m(0,2,4,6,7,8,9,11,12,14)$$



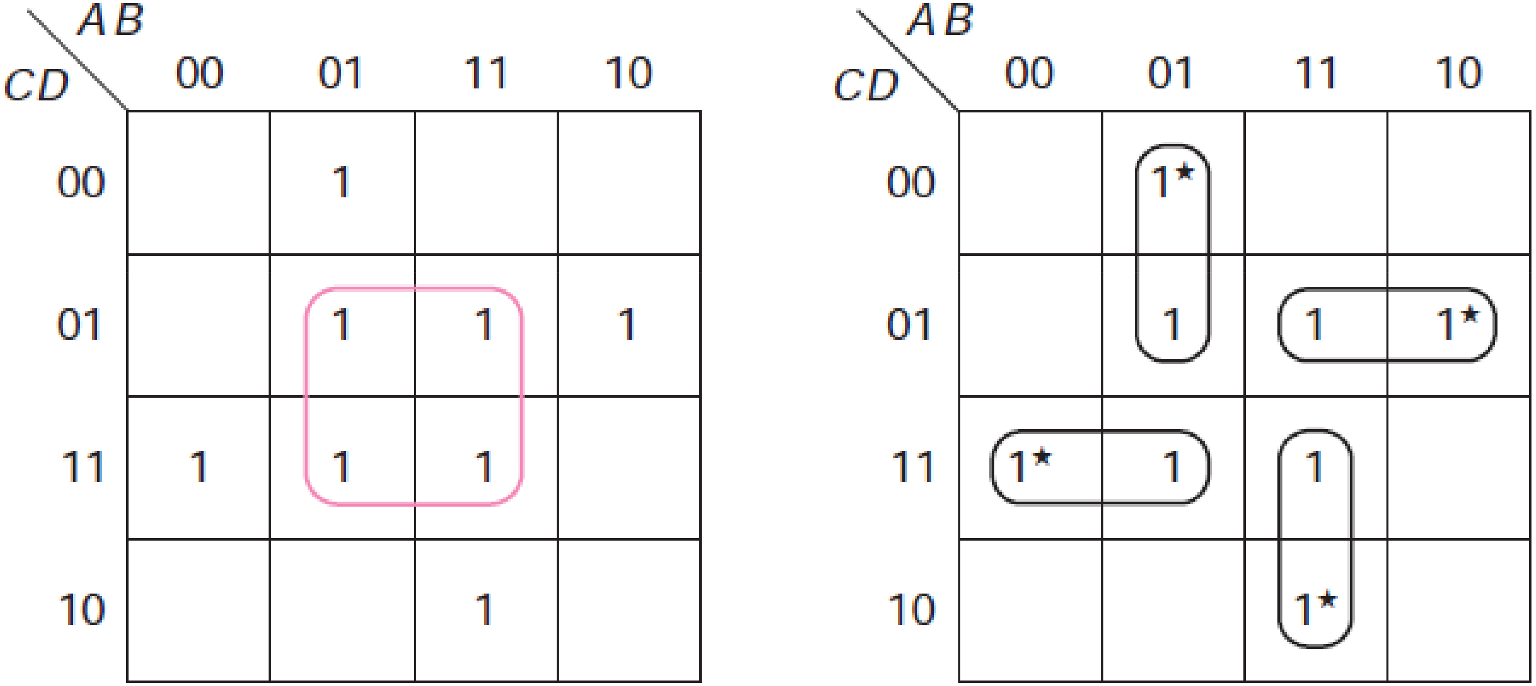


$$f = a'd' + bd' + ab'd + a'bc + c'd'$$

From Marcovitz's Introduction to Logic Design

K-map Example 4

"Don't be greedy" Example



$$f = A'BC' + AC'D + A'CD + ABC$$

From Marcovitz's Introduction to Logic Design

K-map Example 5

cd \ ab	00	01	11	10
00	1	1		1
01		1	1	1
11		1	1	
10	1	1		1

$$f = bd + b'd' + \left\{ \begin{matrix} a'd' \\ a'b \end{matrix} \right\} + \left\{ \begin{matrix} ab'c' \\ ac'd \end{matrix} \right\}$$

From Marcovitz's Introduction to Logic Design

Don’t care conditions

- *Don’t care* means “the value of function not specified”; such functions are called *Incompletely specified functions.*
- Example:

a	b	f
0	0	0
0	1	1
1	0	1
1	1	X

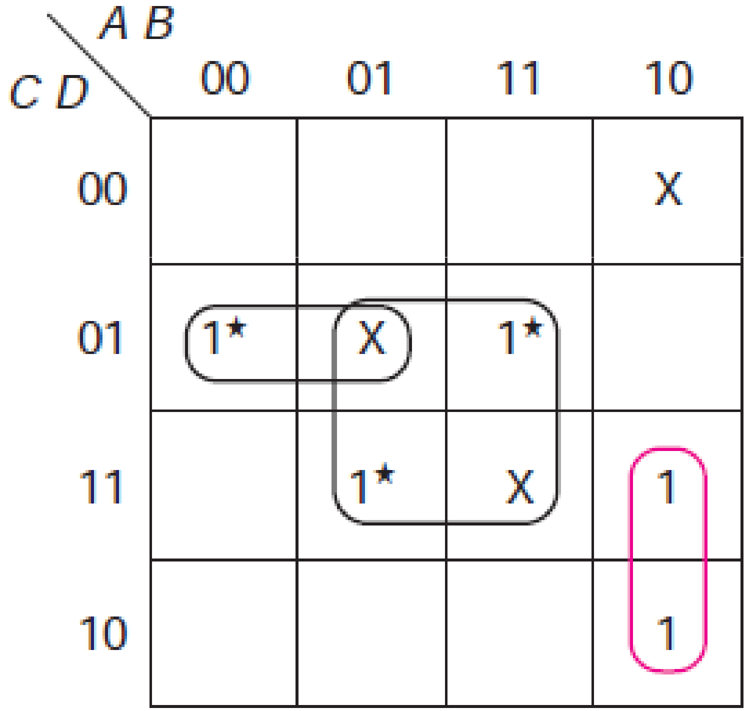
a	b	f_1	f_2
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

- Circuit for 10-digit display: 4-bit input from 0-9, (10-15 don't care!)

K-map Example with don't care 1

$$f(A,B,C,D)=\sum m(1,7,10,11,13)+\sum d(5,8,15)$$

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

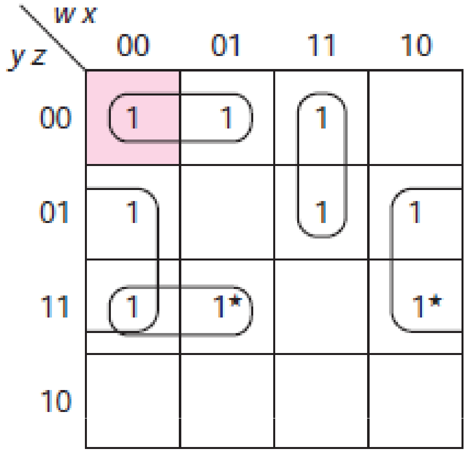


$$f = BD + A'C'D + AB'C$$

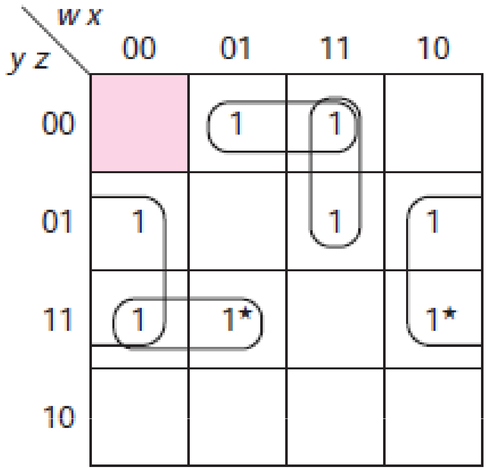
From Marcovitz's Introduction to Logic Design

K-map Example with don't care 2

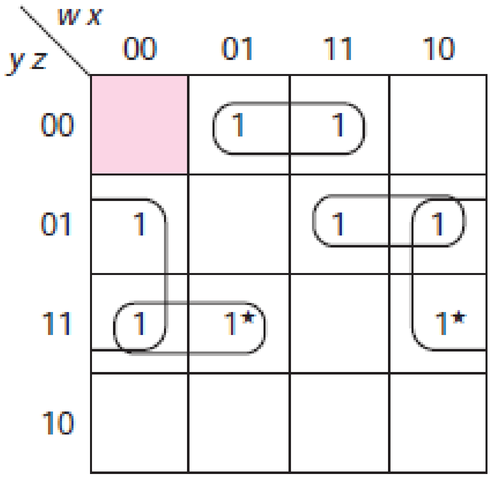
yz \ wx	00	01	11	10
00	X	1	1	
01	X		1	1
11	X	1		1
10	X			



g_1



g_2



g_3

$$f = x'z + w'yz + \begin{Bmatrix} w'y'z' \\ xy'z' \\ xy'z' \end{Bmatrix} + \begin{Bmatrix} wxy' \\ wxy' \\ wy'z \end{Bmatrix}$$

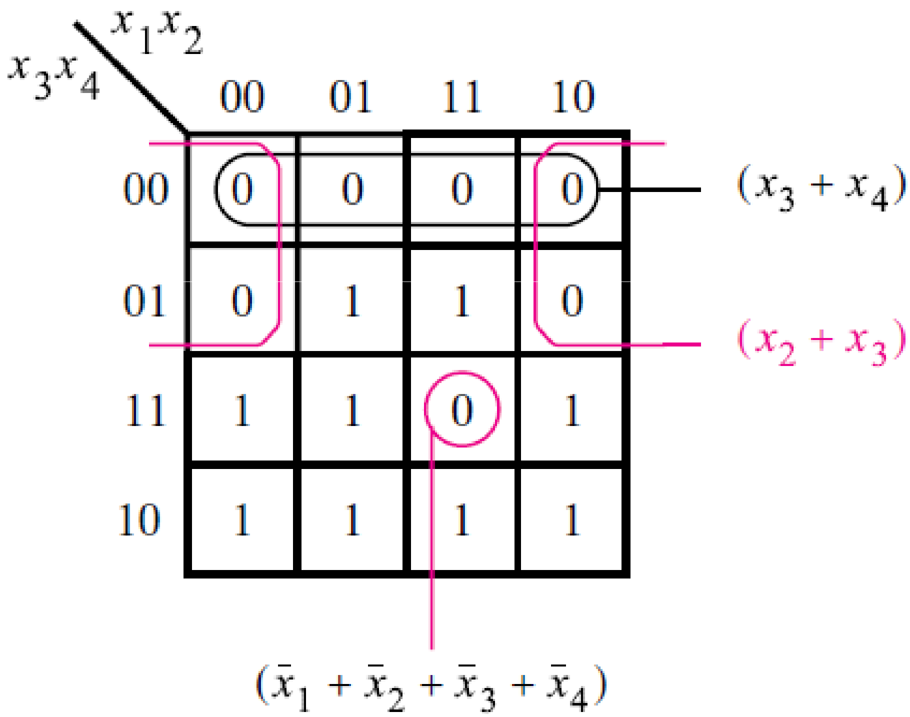
From Marcovitz's Introduction to Logic Design

K-Map & POS Form

- **Direct method : construct K-map and find the minimum cover, then find the POS form.**
- **Find the SOP for *f'*, then obtain the POS using the DeMorgan's law.**

K-map Example POS 1

$$f(x_1, x_2, x_3, x_4) = \sum m(2,3,5,6,7,10,11,13,14) = \prod M(0,1,4,8,9,12,15)$$

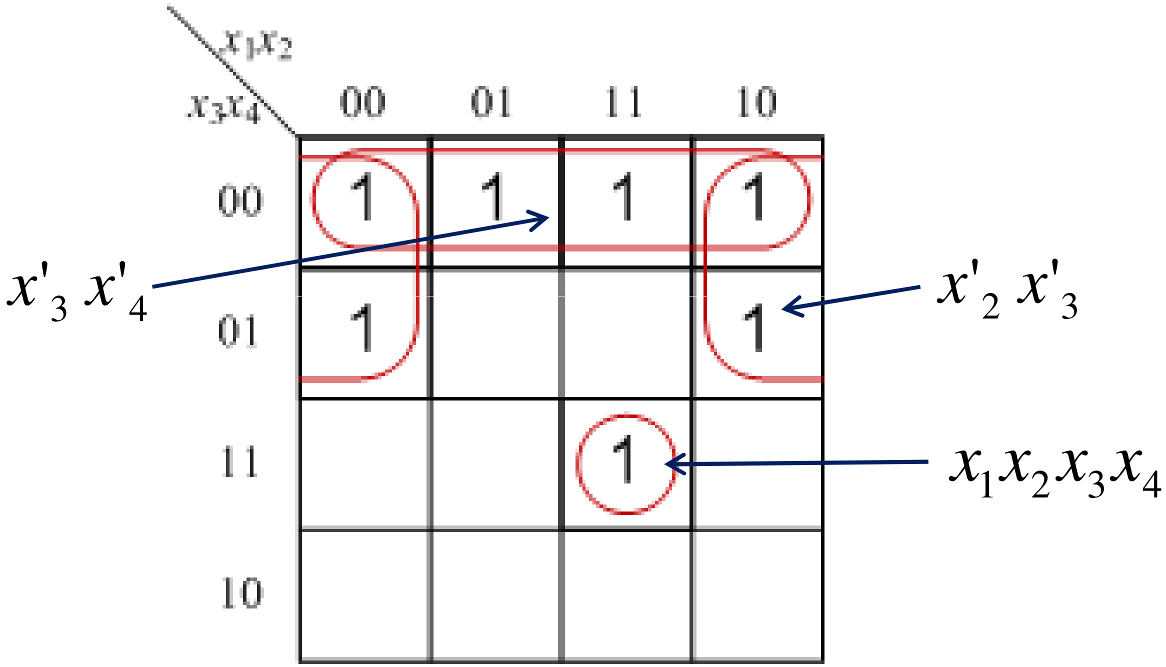


$$f = (x_3 + x_4)(x_2 + x_3)(x'_1 + x'_2 + x'_3 + x'_4)$$

From Brown's Fundamentals of digital logic

K-map Example POS 1 (2)

$$f'(x_1, x_2, x_3, x_4) = \sum m(0,1,4,8,9,12,15)$$



$$f' = x'_3\, x'_4 + x'_2\, x'_3 + x_1x_2x_3x_4$$

$$f = (x_3 + x_4)(x_2 + x_3)(x'_1 + x'_2 + x'_3 + x'_4)$$

From Brown's Fundamentals of digital logic

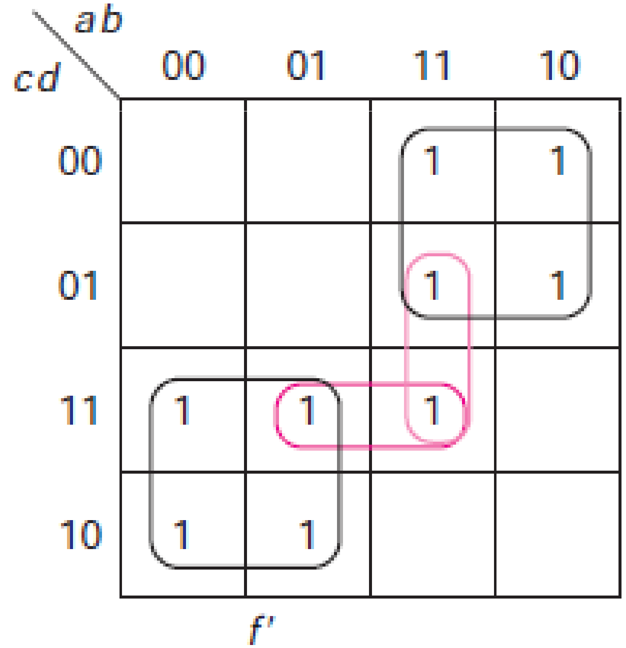
K-map Example POS 2

$$f(a,b,c,d) = \sum m(0,1,4,5,10,11,14)$$

$$f'(a,b,c,d) = \sum m(2,3,6,7,8,9,12,13,15)$$

$$f' = ac' + a'c + \begin{Bmatrix} abd \\ bcd \end{Bmatrix}$$

$$f = (a'+c)(a+c')\begin{Bmatrix} (a'+b'+d') \\ (b'+c'+d') \end{Bmatrix}$$



From Marcovitz's Introduction to Logic Design

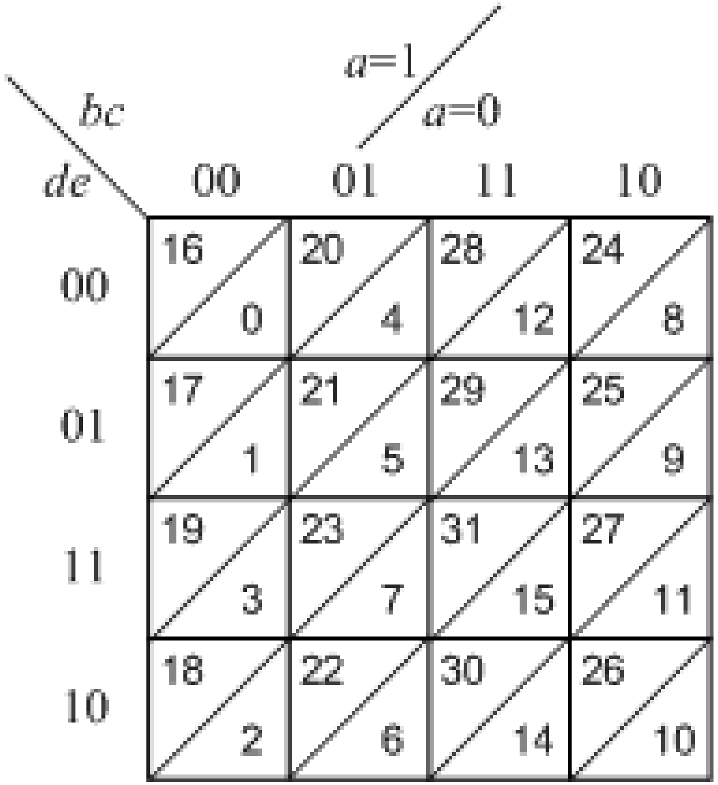
5-variable K-Map

A = 0

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

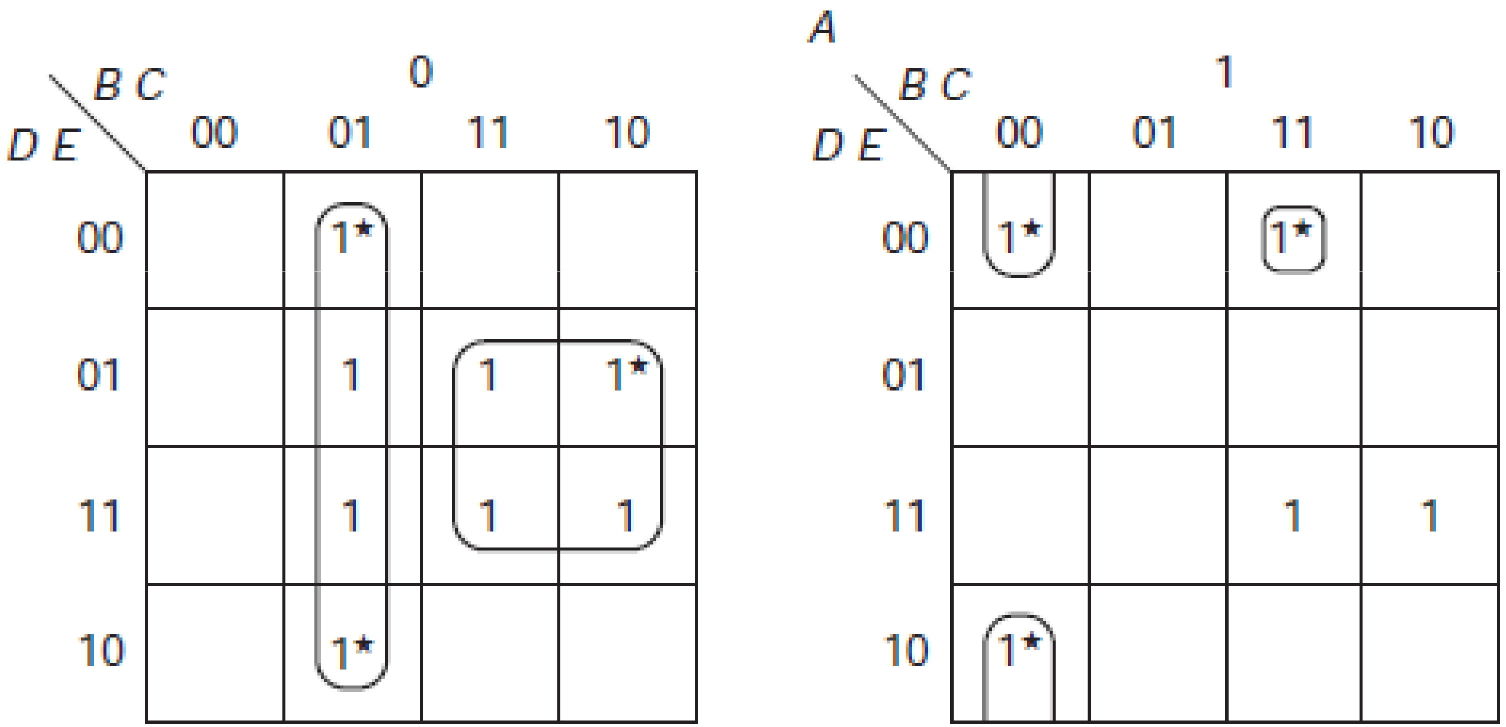
A = 1

16	20	28	24
17	21	29	25
19	23	31	27
18	22	30	26



K-map Example 5-variable 1

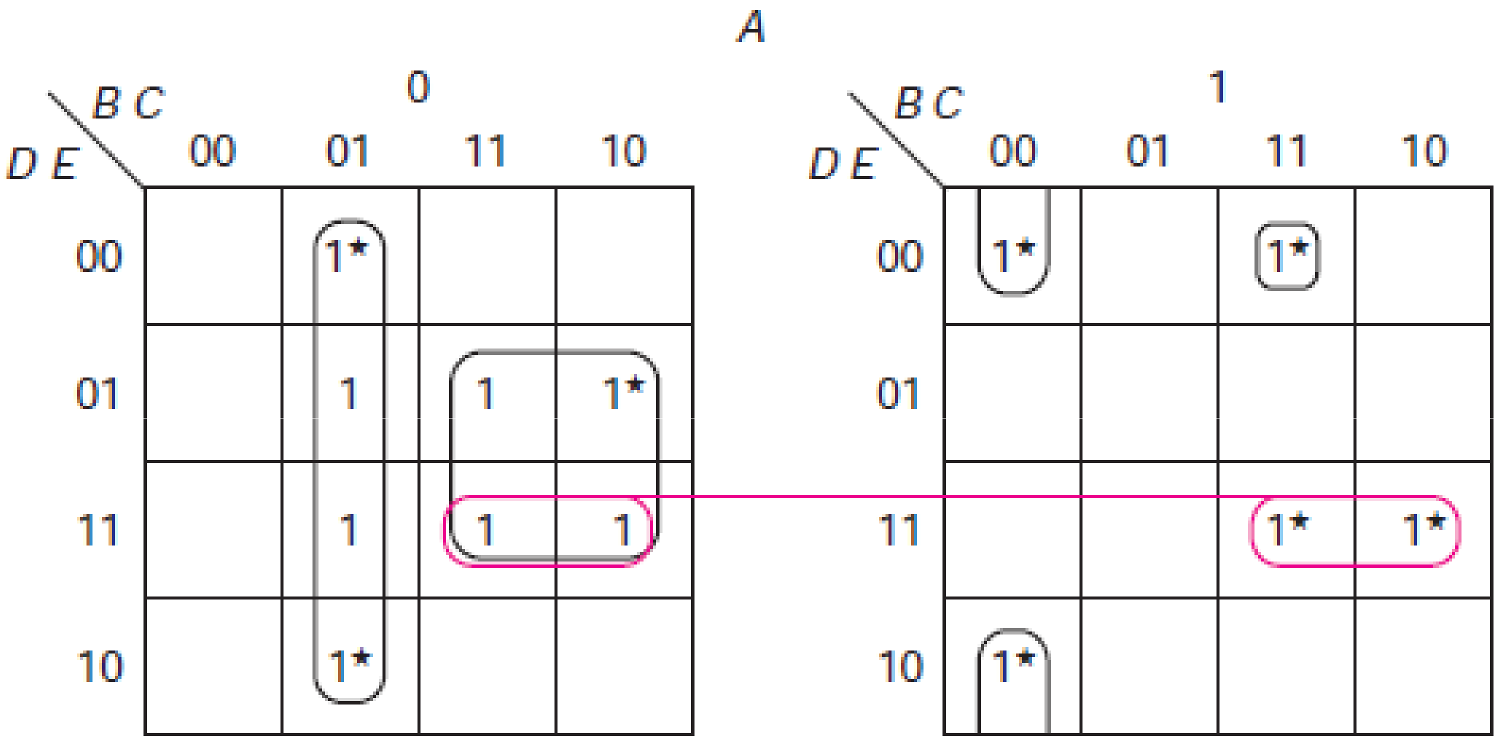
$$f(A,B,C,D,E)=\sum m(4,5,6,7,9,11,13,15,16,18,27,28,31)$$



Essential Prime Implicants on one layer

From Marcovitz's Introduction to Logic Design

K-map Example 5-variable 1 (2)

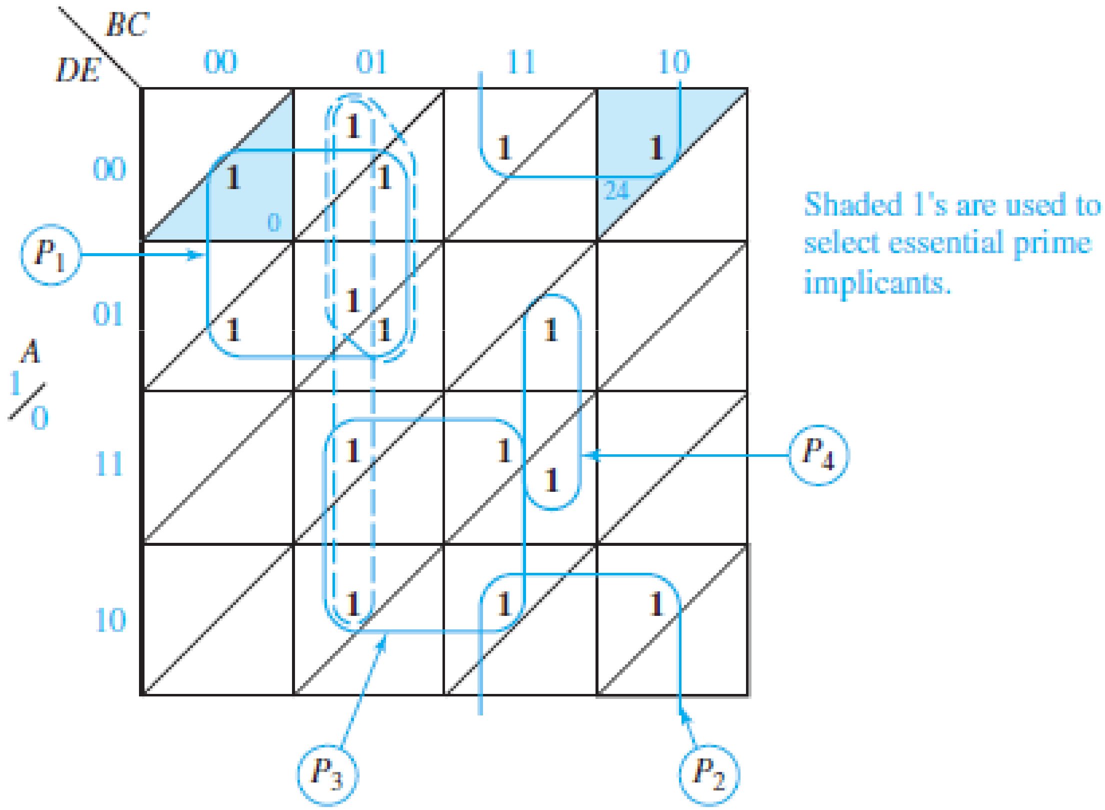


$$f = A'B'C + A'BE + AB'C'E' + ABCD'E' + BDE$$

From Marcovitz's Introduction to Logic Design

K-map Example 5-variable 2

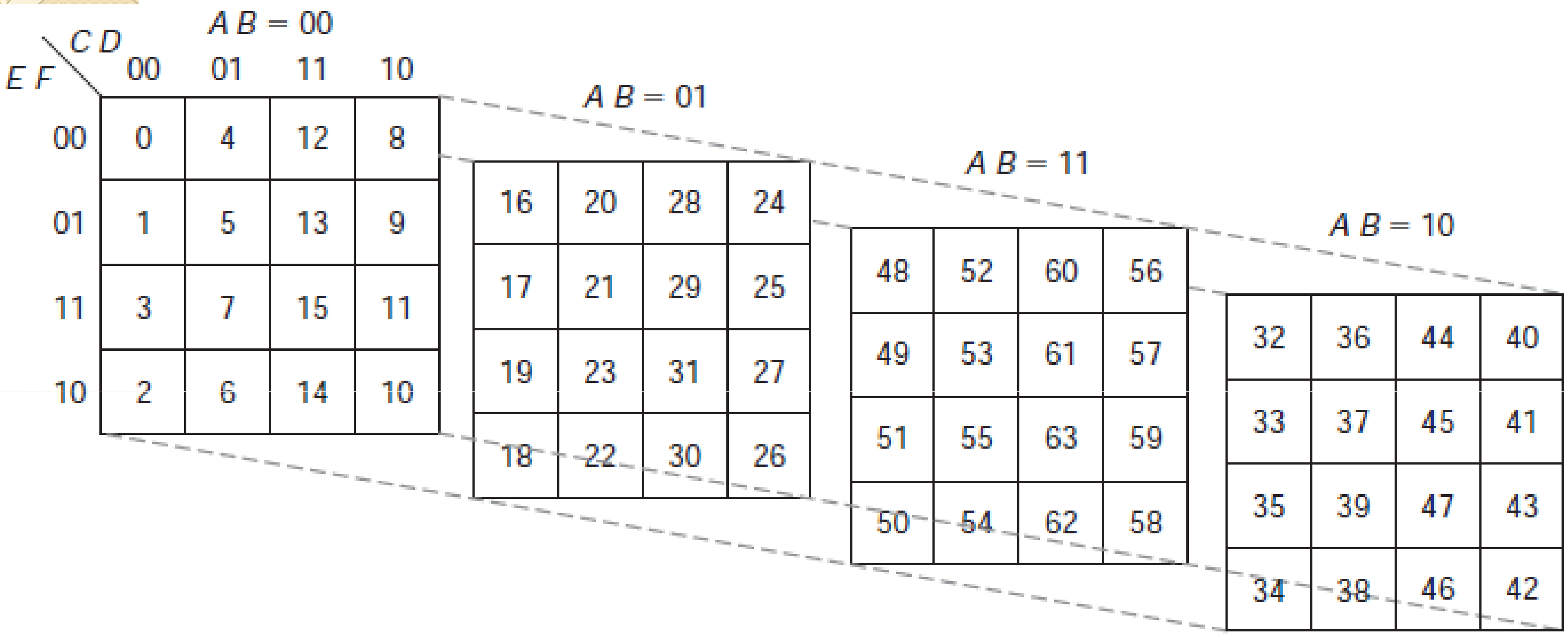
$$f(A,B,C,D,E)=\sum m(0,1,4,5,13,15,20,21,22,23,24,26,28,30,31)$$



$$F = \underset{P_1}{A'B'D'} + \underset{P_2}{ABE'} + \underset{P_3}{ACD} + \underset{P_4}{A'BCE} + \begin{Bmatrix} AB'C \\ \text{or} \\ B'CD' \end{Bmatrix}$$

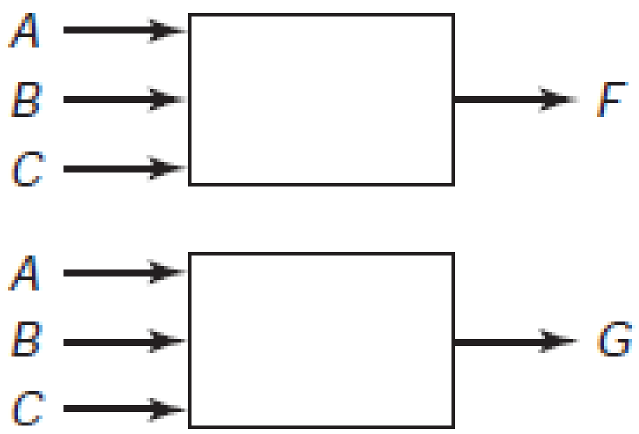
From Roth's Fundamentals of Logic Design

6-variable K-Map

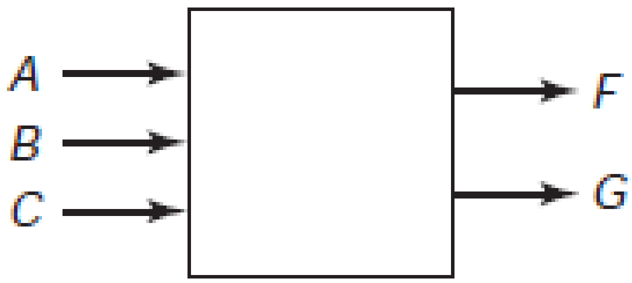


Multiple-Output Problems

- Can design two separate systems for *F* and *G*.
- Or design one system with 2 outputs: *F* and *G*, which may be simpler and more efficient.



Two Separate Systems



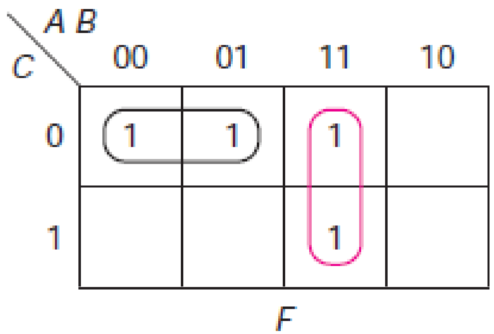
One System

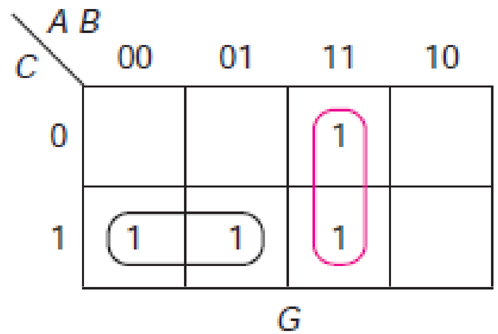
K-map Example 2-output 1

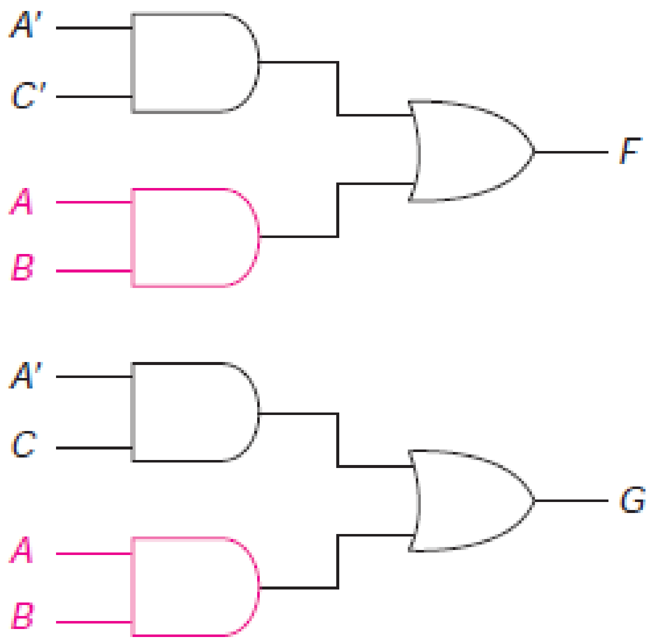
$$F(A,B,C)=\sum m(0,2,6,7);G(A,B,C)=\sum m(1,3,6,7)$$

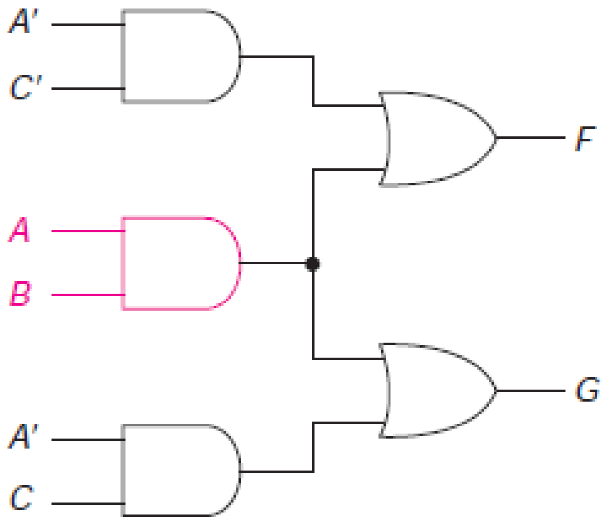
$$F = A'C' + \boxed{AB}$$

$$G = A'C + \boxed{AB}$$









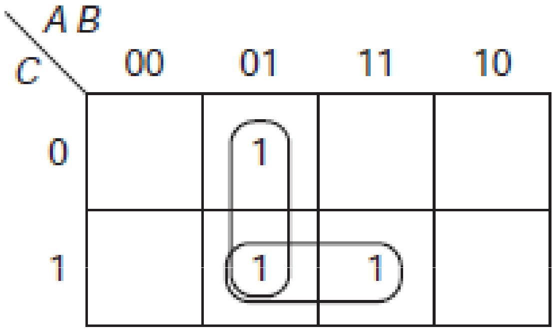
From Marcovitz's Introduction to Logic Design

K-map Example 2-output 2

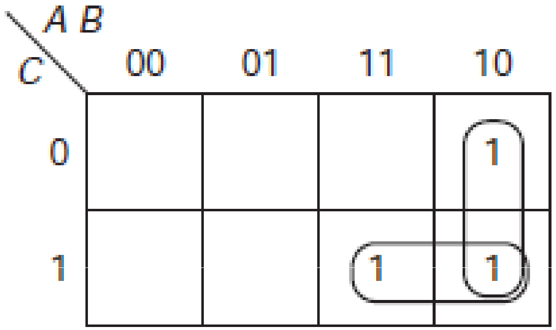
$$F(A,B,C)=\sum m(2,3,7);G(A,B,C)=\sum m(4,5,7)$$

$$F = A'B + BC$$

$$G = AB' + AC$$



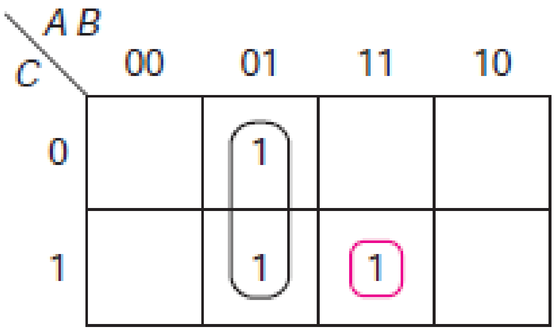
F



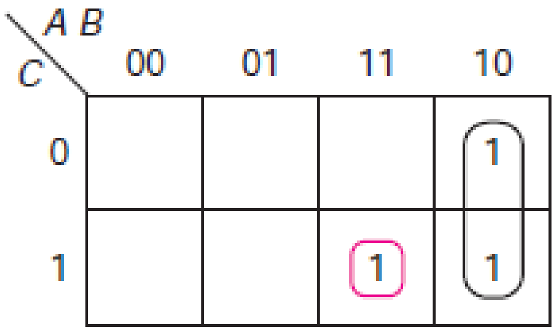
G

$$F = A'B + ABC$$

$$G = AB' + ABC$$



F



G

From Marcovitz's Introduction to Logic Design

K-map Example 2-output 3

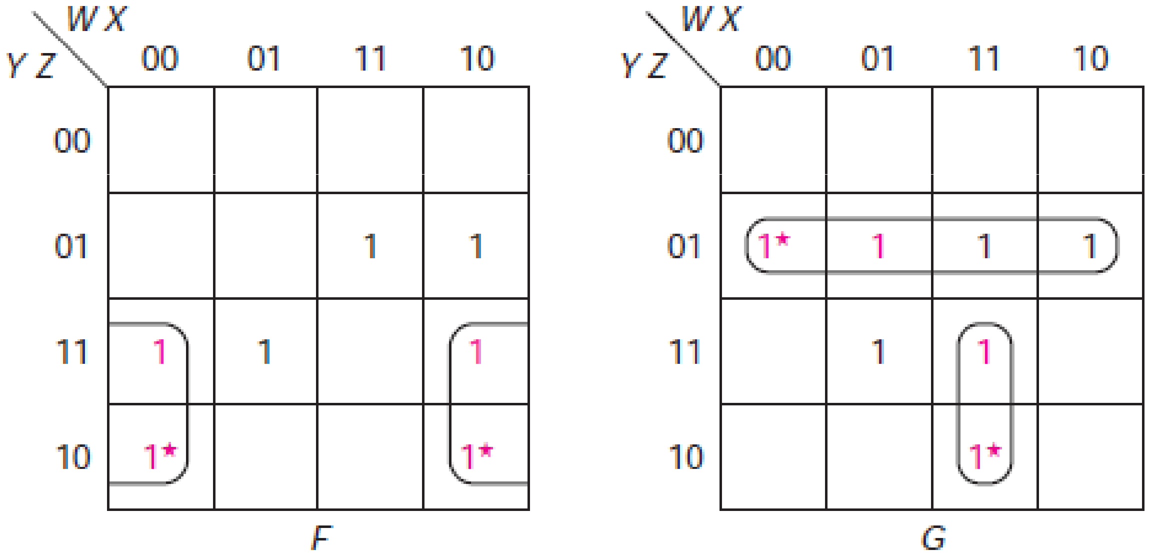
$$F(W,X,Y,Z)=\sum m(2,3,7,9,10,11,13);$$

$$G(W,X,Y,Z)=\sum m(1,5,7,9,13,14,15)$$

$$F = X'Y + WY'Z + \boxed{W'XYZ}$$

$$G = Y'Z + WXY + \boxed{W'XYZ}$$

Total : 20 Inputs, 7 Gates



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

$$G = Y'Z + WXY + XZ$$

Separated System Total : 21 Inputs, 8 Gates

From Marcovitz's Introduction to Logic Design

Summary

- Circuit Optimization
  - Literal cost
  - Gate input cost
- (2,3,4,5)-Variable Karnaugh Maps
- K-Maps with don't care
- K-Maps POS forms
- K-Maps Multiple-output problems