



Karnaugh map covering



Paolo PRINETTO
 Politecnico di Torino (Italy)
 University of Illinois at Chicago, IL (USA)

Paolo.Prinetto@polito.it
 Prinetto@uic.edu
 www.testgroup.polito.it

Goal

- This lecture presents the basic fundamentals of purely manual synthesis of Combinational networks, based on Karnaugh map covering
- Students are then guided through the solution of simple problems
- Algorithmic approaches are briefly introduced
- The lecture eventually focuses on peculiar aspects, such as covering of *chessboard-like* maps and of multiple-output functions.

6.2

2

Prerequisites

- Lectures 6.1 and 3.3

6.2

3

Homework

- Students are warmly encouraged to solve the proposed exercises

6.2

4

Further readings

- Students interested in making a reference to text books on the arguments covered in this lecture can refer, for instance, to:
 - M. Morris Mano, C.R.Kime: “*Logic and Computer Design Fundamentals*,” 2nd edition updated Prentice Hall, Upple Saddle River, NJ (USA), 2001, (pp. 45-75)
 or to

6.2

5

Further readings (cont'd)

- E.J.McCluskey: “*Logic design principles with emphasis on testable semicustom circuits*” Prentice-Hall, Englewood Cliffs, NJ, USA, 1986, (pp. 191-250)

6.2

6

Further readings (cont'd)

- In particular, students interested in *logic minimization algorithms* are recommended to refer to
 - G. De Micheli:
 “*Synthesis and Optimization of digital circuits*,”
 McGraw-Hill, Inc, New York, NJ, USA,
 1994, (chapters 7-8, pp. 269-440)

6.2

7

Outline

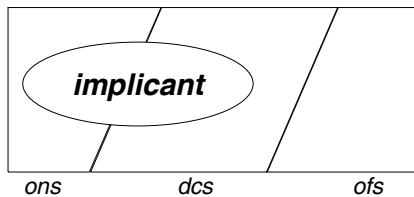
- Implicants
- Covers
- Algorithmic approaches
- Chessboard map covering
- Map composition
- Multiple-output functions.

6.2

8

Implicants

An *implicant* of a function f is a k -cube not including any vertex of the off-set of f .



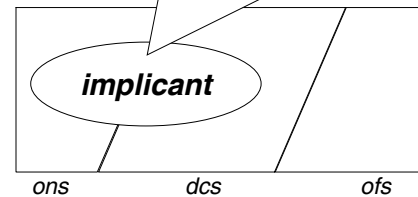
6.2

9

Implicants

An *implicant* including a

- They are usually represented by:
- cubic notation
 - algebraic notation by product terms



6.2

10

Prime implicants

An implicant is *prime* iff

- is not included in any other implicant of f and
- is not completely included in the don't-care set of f .

6.2

11

Note

Since prime implicants only will be of interest in manual designing,, for sake of simplicity, the adjective “prime” will be mostly omitted hereinafter.

6.2

12

Essential implicant

A prime implicant of a function f is *essential* iff it includes at least one vertex of the on-set of f which is not included in any other prime implicant of f .

6.2

13

Implicants on Karnaugh maps

On the Karnaugh map of a function f , any k -cube whose 2^k cells all correspond to vertices *not* belonging to the off-set of f is an implicant.

6.2

14

Complete sum

A *complete sum* is a representation of a function as the sum of all its prime implicant.

6.2

15

Example

	ab			
cd	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

Find *all* the prime implicants and the essential implicants.

6.2

16

Solution Prime implicants

	ab			
cd	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

6.2

17

Solution Prime implicants

	ab			
cd	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

$a'c'd'$

6.2

18

Solution
Prime implicants

<i>cd</i> \ <i>ab</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

6.2

19

Solution
Prime implicants

<i>cd</i> \ <i>ab</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

$a' b d'$

6.2

20

Solution
Prime implicants

<i>cd</i> \ <i>ab</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

6.2

21

Solution
Prime implicants

<i>cd</i> \ <i>ab</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

$b' c' d'$

6.2

22

Solution
Prime implicants

<i>cd</i> \ <i>ab</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

6.2

23

Solution
Prime implicants

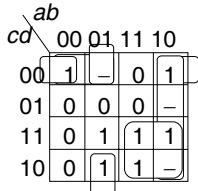
<i>cd</i> \ <i>ab</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

$a b'$

6.2

24

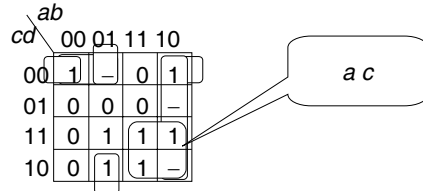
Solution
Prime implicants



6.2

25

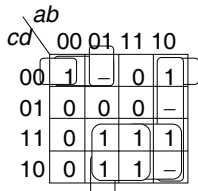
Solution
Prime implicants



6.2

26

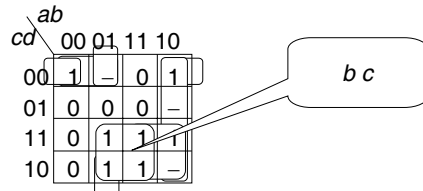
Solution
Prime implicants



6.2

27

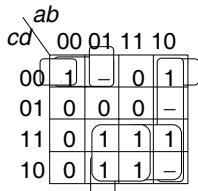
Solution
Prime implicants



6.2

28

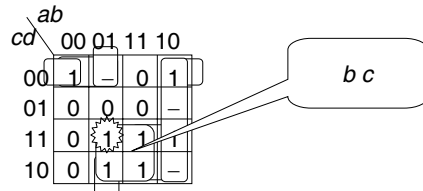
Solution
Essential implicants



6.2

29

Solution
Essential implicants



6.2

30

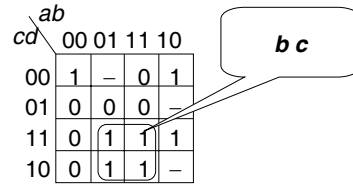
Property

If you erase a variable from a prime implicant of a function f , the resulting product term is no longer an implicant of f .

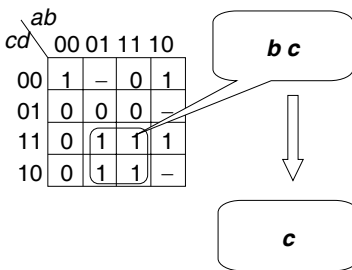
6.2

31

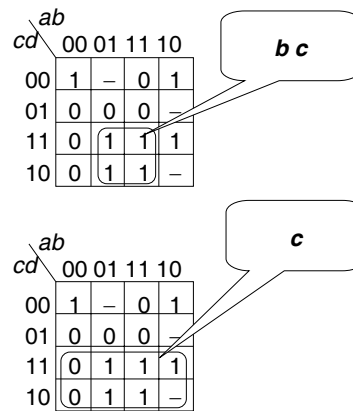
Example



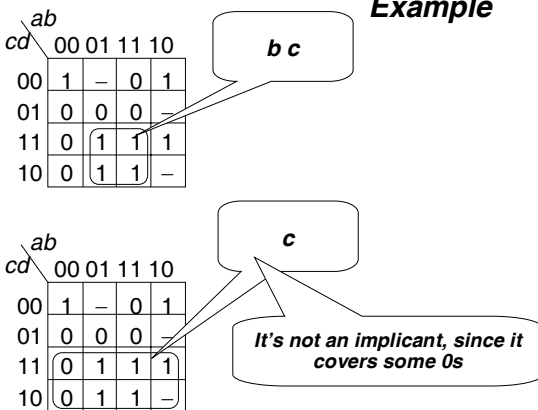
Example



Example



Example



Outline

- Implicants
- ⇒ Covers
- Algorithmic approaches
- Chessboard map covering
- Map composition
- Multiple-output functions

6.2

36

Function Covers

A *cover* of a function f is a function C including all the vertices of the on-set and no vertex of the off-set of f :

$$\text{ons}(f) \subseteq C \subseteq \text{ons}(f) \cup \text{dcs}(f)$$

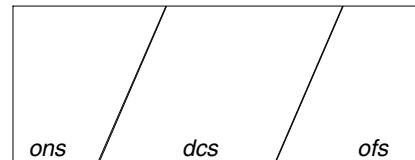
6.2

37

Function Covers

A *cover* of a function f is a function C including all the vertices of the on-set and no vertex of the off-set of f :

$$\text{ons}(f) \subseteq C \subseteq \text{ons}(f) \cup \text{dcs}(f)$$



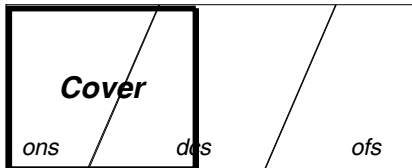
6.2

38

Function Covers

A *cover* of a function f is a function C including all the vertices of the on-set and no vertex of the off-set of f :

$$\text{ons}(f) \subseteq C \subseteq \text{ons}(f) \cup \text{dcs}(f)$$



6.2

39

Irredundant cover

A cover C is *irredundant* iff erasing any of its terms would result in a set of cubes which is not a cover of f .

6.2

40

Finding irredundant covers

Finding an irredundant cover of a function relies on the following theorem, proved by Quine in '52.

6.2

41

Theorem

A minimal sum must always consist of a sum of prime implicants if any definition of cost is used in which the addition of a single literal to any expression increases the cost of the expression.

6.2

42

Corollary

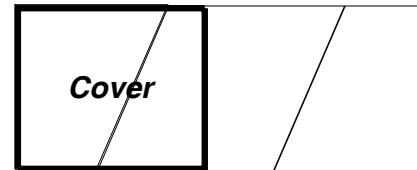
If an s-o-p function f is minimal w.r.t. either the # of logic gates or the # gate inputs, then each of its product term corresponds to a prime implicant of f .

6.2

43

Cover by s-o-p expressions

A possible cover of a function f is given by a set of prime implicants covering any "1" of f , at least once, and, if needed, some of the "-" of f .



6.2

44

Covering incompletely specified functions

When dealing with incompletely specified functions, the vertices of dcs can be freely covered or not, in dependence of designer's convenience.

6.2

45

Example

	<i>ab</i>			
<i>cd</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

Find an irredundant cover.

6.2

46

**Solution
Irredundant cover**

	<i>ab</i>			
<i>cd</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

$f =$

6.2

47

**Solution
Irredundant cover**

	<i>ab</i>			
<i>cd</i>	00	01	11	10
00	1	-	0	1
01	0	0	0	-
11	0	1	1	1
10	0	1	1	-

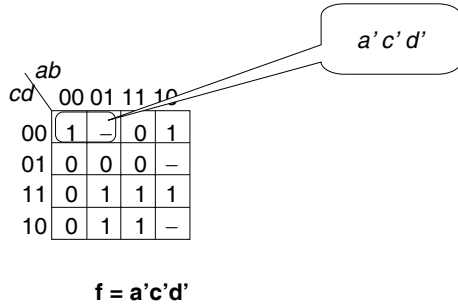
$f =$

$a'c'd'$

6.2

48

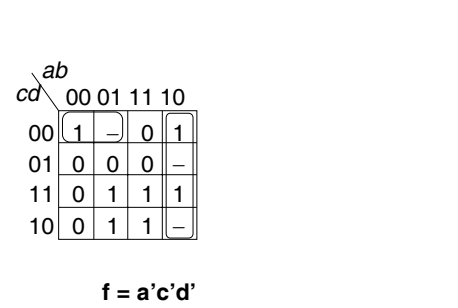
**Solution
Irredundant cover**



6.2

49

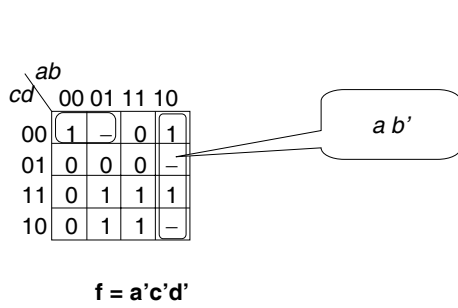
**Solution
Irredundant cover**



6.2

50

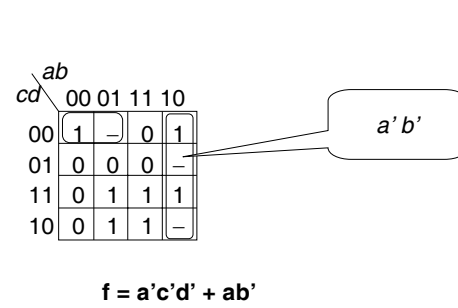
**Solution
Irredundant cover**



6.2

51

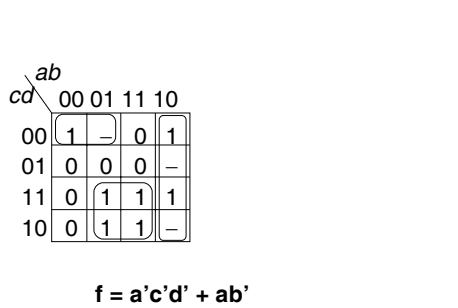
**Solution
Irredundant cover**



6.2

52

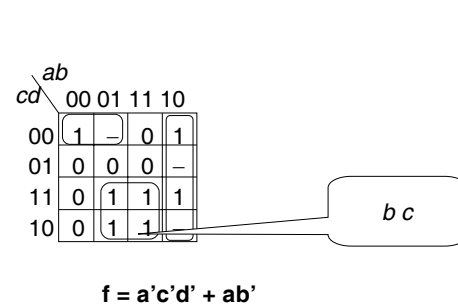
**Solution
Irredundant cover**



6.2

53

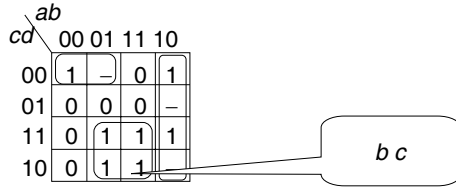
**Solution
Irredundant cover**



6.2

54

Solution
Irredundant cover



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

6.2

55

Outline

- Implicants
- Covers
- ⇒ *Algorithmic approaches*
- Chessboard map covering
- Map composition
- Multiple-output functions.

6.2

56

Algorithmic approaches

- Several algorithmic approaches have been proposed to generate minimal covers.
- They can be clustered as:
 - *Exact logic minimization algorithms*
 - *Heuristic logic minimization algorithms*

6.2

57

Note

A detailed presentation of these algorithms is outside the scope of this course. In the following we shall just introduce the basic ideas behind some of them.

6.2

58

Exact approaches

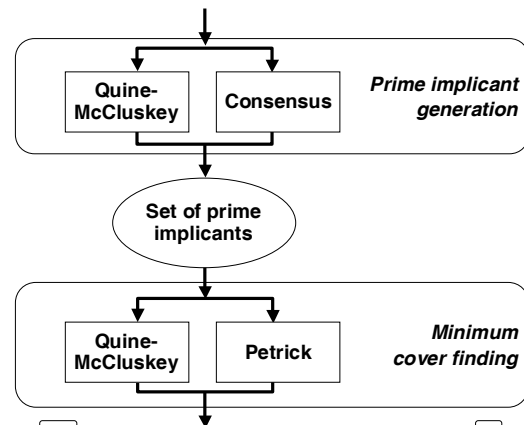
Exact approaches mostly rely on two steps:

- Generating all the prime implicants of the function $ons(f) \cup dcs(f)$
- Selecting a subset of these prime implicants that is an irredundant minimum cost cover of $ons(f)$.

Each step can be performed in several ways.

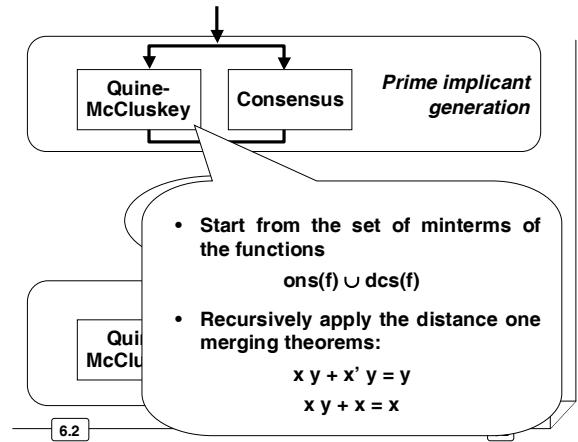
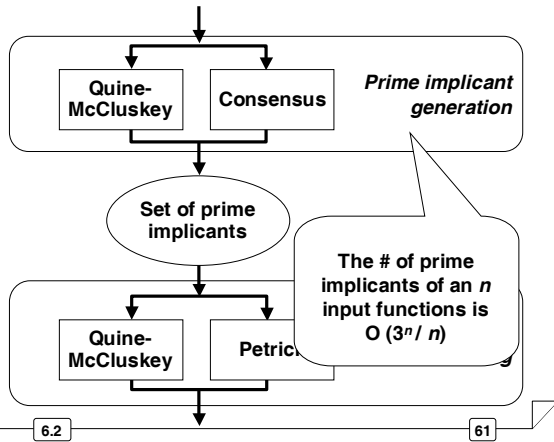
6.2

59



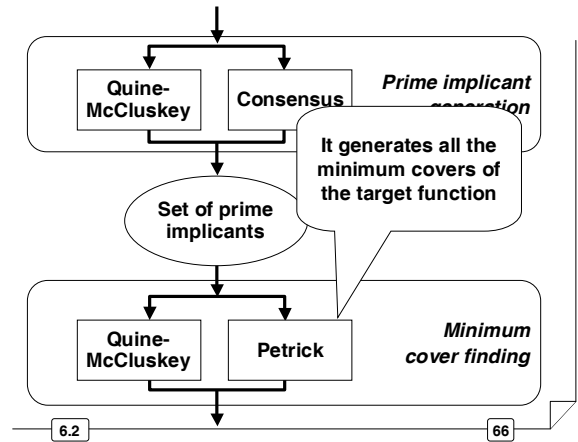
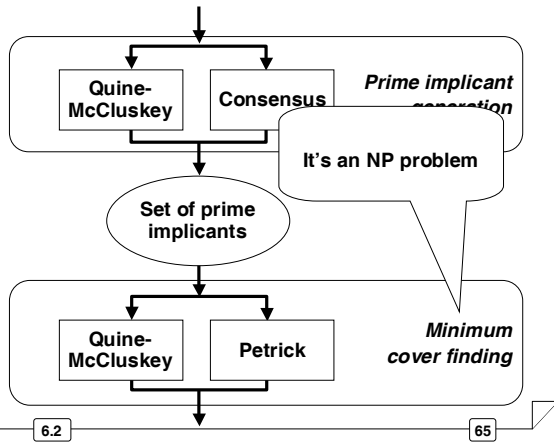
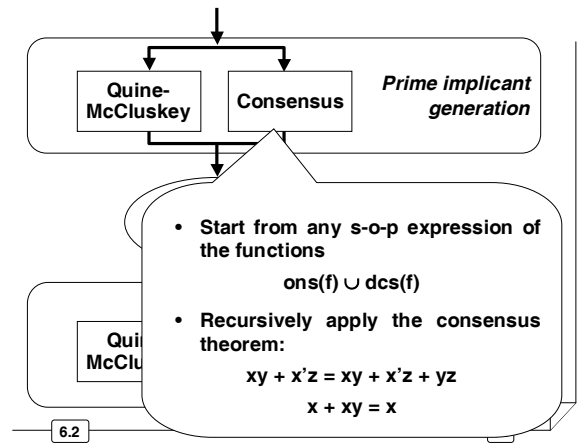
6.2

60



Example of merging

0	1	0	1	1
0	1	1	1	1
0	1	-	1	1



Minimization tools

- Several minimization tools are available.
- They will be covered in Lecture 6.4.

6.2

67

Outline

- Implicants
- Covers
- Algorithmic approaches
 - ⇒ *Chessboard map covering*
- Map composition
- Multiple-output functions

6.2

68

Chessboard-like maps

When dealing with chessboard (or chessboard-like) maps, it may be convenient to resort to exor functions.

6.2

69

Example

Design the sum output bit S_i of a 1-bit *full-adder*.

6.2

70

Example

Design the sum output bit S_i of a 1-bit *full-adder*.

	$a_i b_i$			
c_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0
	S_i			

6.2

71

Example

Design the sum output bit S_i of a 1-bit *full-adder*.

	$a_i b_i$			
c_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0
	S_i			

$$S_i = a_i \oplus b_i \oplus c_i$$

6.2

72

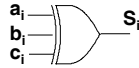
Example

Design the sum output bit S_i of a 1-bit *full-adder*.

	a_i, b_i			
c_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

S_i

$S_i = a_i \oplus b_i \oplus c_i$



6.2

73

Outline

- Implicants
- Covers
- Algorithmic approaches
- Chessboard map covering
⇒ *Map composition*
- Multiple-output functions.

6.2

74

Map composition

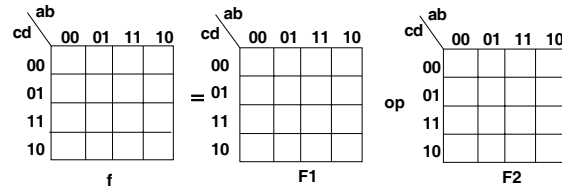
- It can sometimes be convenient to get the cover of a function by resorting to a set of proper *composing functions*:

6.2

75

Map composition

- It can sometimes be convenient to get the cover of a function by resorting to a set of proper *composing functions*:

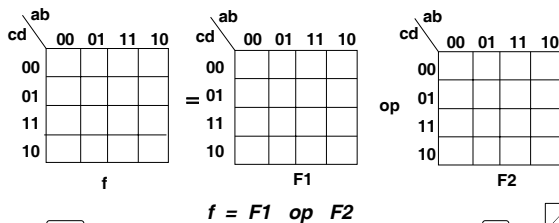


6.2

76

Map composition

- It can sometimes be convenient to get the cover of a function by resorting to a set of proper *composing functions*:



6.2

77

**A typical example:
covering chessboard-like maps**

A typical example is provided by covering maps that have either some additional 0's w.r.t. the chessboard or some additional 1's, or both.

6.2

78

Maps with additional 0's

Implement the target function f as:

$$f = E \cdot R$$

where:

- E is an exor gate having as many inputs as f
- R is a masking function (having as many inputs as f), in charge of masking those 1's of E for which f must be 0.

f	E	R
0	0	-
0	1	0
1	0	not possible
1	1	1

6.2

79

Example #1

ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	0
10	1	0	0	0

f

6.2

80

Example #1

ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	0
10	1	0	0	0

f

Additional 0's

6.2

81

Solution

ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

f

=

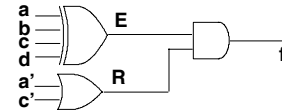
ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

E

^

ab \ cd	00	01	11	10
00	-	1	-	1
01	1	-	1	-
11	-	1	-	0
10	1	-	0	-

R



6.2

82

Example #2

ab \ cd	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

U

6.2

83

Solution

ab \ cd	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

U

$U = 1$ when $ab=cd$, i.e., when $(a=c) \wedge (b=d)$, and thus:

$$U = (a \oplus c)' (b \oplus d)'$$

6.2

84

Solution

One can get the same result on the basis of the following reasoning:

6.2

85

Solution

	ab	00	01	11	10
cd	00	1	0	0	0
	01	0	1	0	0
	11	0	0	1	0
	10	0	0	0	1

 $=$

	ab	00	01	11	10
cd	00	1	1	0	0
	01	1	1	0	0
	11	0	0	1	1
	10	0	0	1	1

 \wedge

	ab	00	01	11	10
cd	00	1	0	-	-
	01	0	1	-	-
	11	-	-	1	0
	10	-	-	0	1

$U = F G$

6.2

86

Solution

	ab	00	01	11	10
cd	00	1	0	0	0
	01	0	1	0	0
	11	0	0	1	0
	10	0	0	0	1

 $=$

	ab	00	01	11	10
cd	00	1	1	0	0
	01	1	1	0	0
	11	0	0	1	1
	10	0	0	1	1

 \wedge

	ab	00	01	11	10
cd	00	1	0	-	-
	01	0	1	-	-
	11	-	-	1	0
	10	-	-	0	1

$F = (a \oplus c)'$
 $G = (b \oplus d)'$
 $U = F G = (a \oplus c)' (b \oplus d)'$

6.2

87

Example #3

	ab	00	01	11	10
cd	00	0	-	0	-
	01	-	0	-	0
	11	0	-	0	-
	10	1	0	1	0

U

6.2

88

Solution

	ab	00	01	11	10
cd	00	0	-	0	-
	01	-	0	-	0
	11	0	-	0	-
	10	1	0	1	0

U

One can get alternative solutions, according to the values assigned to “-”

6.2

89

Solution #1

	ab	00	01	11	10
cd	00	0	-	0	-
	01	-	0	-	0
	11	0	-	0	-
	10	1	0	1	0

 \Rightarrow

	ab	00	01	11	10
cd	00	0	1	0	1
	01	1	0	1	0
	11	0	1	0	1
	10	1	0	1	0

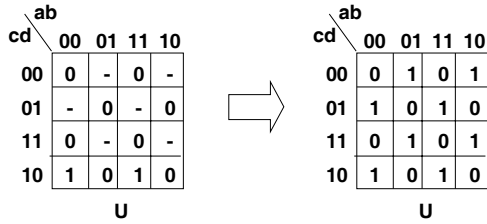
U

U

6.2

90

Solution #1

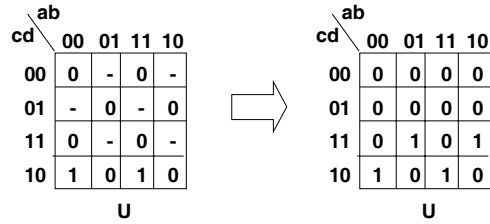


$U = a \oplus b \oplus c \oplus d$

6.2

91

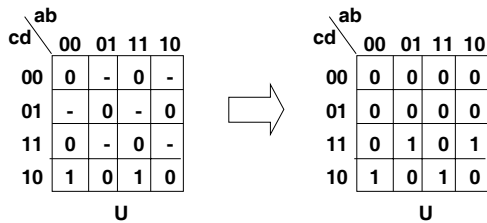
Solution #2



6.2

92

Solution #2



$U = (a \oplus b \oplus d) c'$

6.2

93

Outline

- Implicants
 - Covers
 - Algorithmic approaches
 - Chessboard map covering
 - Map composition
- ⇒ *Multiple-output functions*

6.2

94

Multiple outputs minimization

Methods used so far are cannot guarantee an optimal covering when dealing with multiple outputs functions.

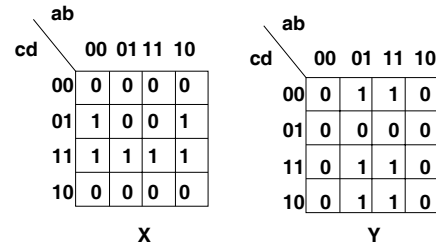
In most cases a minimal solution must resort to sharing some implicants among two or more functions.

6.2

95

Example

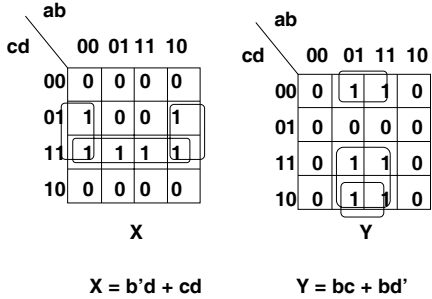
Minimize the expression of the outputs X and Y of a given circuits, defined as follows:



6.2

96

Independent covers



6.2

97

1st implementation

The solution *looks optimal*, since no shareable implicants exist:

- $X = b'd + cd$
- $Y = bc + bd'$

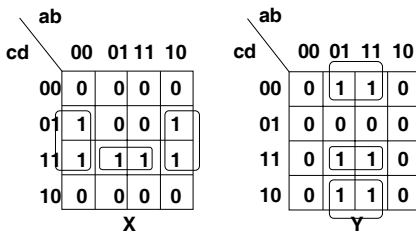
Cost:

- 4 *and* functions (\Rightarrow 4 *and* gates)
- 2 *or* functions (\Rightarrow 2 *or* gates)

6.2

98

Better implementation



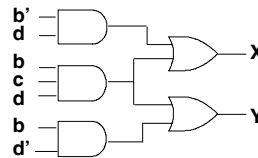
The two covers share an implicant which is not prime:

- $X = b'd + bcd$
- $Y = bd' + bcd$

6.2

99

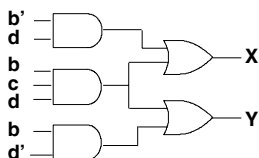
- $X = b'd + bcd$
- $Y = bd' + bcd$



6.2

100

- $X = b'd + bcd$
- $Y = bd' + bcd$



Cost:

- 3 *and* functions (\Rightarrow 3 *and* gates)
- 2 *or* functions (\Rightarrow 2 *or* gates)

6.2

101

Note

A systematic approach to the manual optimization of multiple output functions is outside the scope of the present course.

6.2

102

