Combinational Logic Design Principles. Combinational Circuits Synthesis Using Karnaugh Maps

Doru Todinca

Department of Computers
Politehnica University of Timisoara

Outline

Combinational-Circuit Synthesis

- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- “Don’t Care” Input Combinations
- Simplifying Product of Sums

Timing Hazards

- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Outline

Combinational-Circuit Synthesis
- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- "Don't Care" Input Combinations
- Simplifying Product of Sums

Timing Hazards
- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Outline

Combinational-Circuit Synthesis
- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- “Don’t Care” Input Combinations
- Simplifying Product of Sums

Timing Hazards
- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Karnaugh Maps

- A *Karnaugh map* is a graphical representation of the truth table of a logic function.
- Figure 1 presents Karnaugh maps for functions of two (a), three (b) and four variables (c).
- The Karnaugh map of an n-input logic function is an array containing 2^n cells, one cell for each input combination (minterm).
- The rows and columns of a Karnaugh map are labeled so that the input combination for each cell can be determined from the row and column headings of that cell.
- The small number inside each cell is the corresponding row number (minterm number) in the truth table, assuming that the truth table inputs are labeled alphabetically from left to right (e.g. W, X, Y, Z), and that the rows are numbered in binary ascending order.
  - For example, in a 4-variable map (fig 1 (c)), cell number 13 corresponds to the table row in which $WXYZ = 1101$, cell 7 to the table row $WXYZ = 0111$ and so on

Karnaugh Maps

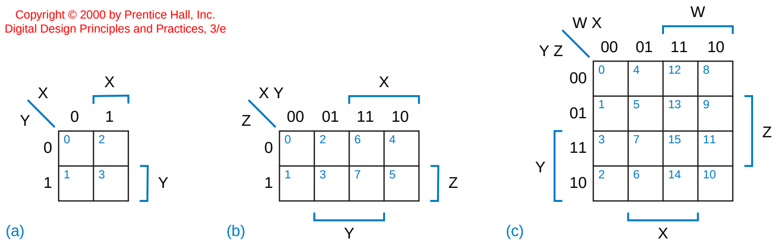
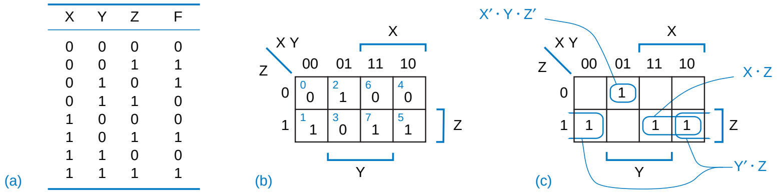


Figure 1 : Karnaug maps: (a) 2-variable; (b) 3-variable; (c) 4-variable

Karnaugh Maps

- Each cell from the map contains the output of the function that corresponds to that input combination (to that table row)
- We use two redundant labeling for rows and columns (see for example the 4-variable map from figure 1 (c)):
  1. The columns are numbered with the four possible combinations of W and X, and the rows with the four combinations of Y and Z: 00, 01, 11 and 10.
  2. We use also brackets to associate four regions of the map, one region for each variable, indicating where the variable is 1.
- The columns and rows numbering give all information we need, and so do the brackets, but we use both of them.
- To represent a logic function on a Karnaugh map we copy the 1s and 0s from the truth table (or one of the other equivalent representations of a truth table)
- Figure 2 shows the truth table and the Karnaugh map for the 3-variable function given in table 4.7.
- In figure 2 (c) we copied only the 1s from the truth table.
- In general we will copy either the 1s, or the 0s, not both.

Karnaugh Maps for Logic Functions: Example 1



X	Y	Z	F
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	0
1	1	1	1

Figure 2 : $F = \sum_{X,Y,Z}(1,2,5,7)$; (a) truth table; (b) Karnaugh map; (c) combining adjacent 1-cells

Minimizing with Karnaugh maps

- Ordering of rows and columns in Karnaugh maps is quite unusual: 00, 01, **11**, 10 (not 00, 01, **10**, 11)
- This is because "each cell corresponds to an input combination that differs from each of its immediately adjacent neighbors in **only** one variable"
- Note that cells from the left/right and top/bottom borders (edges) are also neighbors.
- Examples: cells 1 and 5 from figure 2 are neighbors that differ only in variable X, cells 7 and 5 differ only in variable Y.
- Minimization is based on theorem T10 generalized: $term \cdot Y + term \cdot Y' = term$:
- From two adjacent cells that both contain 1 values we can eliminate the variable that changes value.

Minimizing with Karnaugh maps

Example 1:

- From neighboring cells 1 and 5, which both contain 1, we can write: $X' \cdot Y' \cdot Z + X \cdot Y' \cdot Z = Y' \cdot Z(X + X') = Y' \cdot Z$
- From neighboring cells 7 and 5 that both contain 1, we have: $X \cdot Y \cdot Z + X \cdot Y' \cdot Z = X \cdot Z$
- The 1 from cell number 2 has no other 1s as neighbors. Its expression is $X' \cdot Y \cdot Z'$
- It results that the function F can be expressed as: $F = \sum_{X,Y,Z}(1,2,5,7) = X \cdot Z + Y' \cdot Z + X' \cdot Y \cdot Z'$, and its realization is given in figure 3

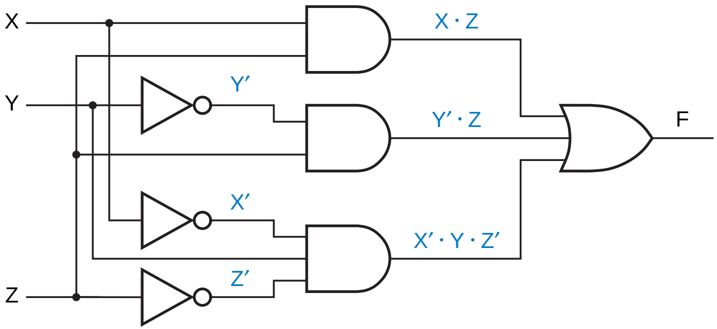


Figure 3 : Minimized AND-OR circuit

Outline

Combinational-Circuit Synthesis
Karnaugh Maps
Minimizing Sums of Products
Karnaugh Maps for Five and Six Variables Logic Functions
“Don’t Care” Input Combinations
Simplifying Product of Sums

Timing Hazards
Static Hazards
Finding Static Hazards Using Maps
Dynamic Hazards

- In general we can simplify a logic function by first combining pairs of adjacent cells that contain a 1 inside (we call them *1-cells*)
- We circle the pair of 1-cells to indicate that the corresponding minterms are added, resulting a single product term
- We want to select a set of product terms that cover all 1s from the Karnaugh map.
- In example 1 (figure 2) we circled only pairs of 1s, but we can extend the procedure, i.e., to combine more than 2 minterms in a product term
- Next step would be to combine (if possible) two neighboring product terms that have been obtained by combining pairs of minterms
- Hence we have a set of 4 minterms
- If we continue the procedure, it results that the number of cells (minterms) combined will be always a power of 2

Example 2

- In example 2 from figure 4, the function to minimize is $F = \sum_{X,Y,Z}(0,1,4,5,6)$
- If we work the function algebraically we have: $F = X' \cdot Y' \cdot Z' + X' \cdot Y' \cdot Z + X \cdot Y' \cdot Z' + X \cdot Y' \cdot Z + X \cdot Y \cdot Z' = (X' \cdot Y' \cdot Z' + X' \cdot Y' \cdot Z) + (X \cdot Y' \cdot Z' + X \cdot Y' \cdot Z) + X \cdot Y \cdot Z' = (X' \cdot Y' \cdot (Z' + Z)) + (X \cdot Y' \cdot (Z' + Z)) + X \cdot Y \cdot Z' = X' \cdot Y' + X \cdot Y' + X \cdot Y \cdot Z' = (X' \cdot Y' + X \cdot Y') + X \cdot Y \cdot Z' = Y' \cdot (X' + X) + X \cdot Y \cdot Z' = Y' + X \cdot Y \cdot Z'$

Example 2

- Using Karnaugh maps, we first combine pairs of minterms:
  1. from cells 0 and 1, i.e. minterms : $X' \cdot Y' \cdot Z'$ and $X' \cdot Y' \cdot Z$, which give:
     $X' \cdot Y' \cdot Z' + X' \cdot Y' \cdot Z = X' \cdot Y' \cdot (Z + Z') = X' \cdot Y'$
  2. from cells 4 and 5, we have: $X \cdot Y' \cdot Z' + X \cdot Y' \cdot Z = X \cdot Y'$
  3. from cells 4 and 6 we have:
     $X \cdot Y' \cdot Z' + X \cdot Y \cdot Z' = X \cdot Z' \cdot (Y + Y') = X \cdot Z'$
- In step 2 we combine product terms 1 and 2 obtained in step 1 (which incorporate now the cells 0, 1, 4 and 5) and we have: $X' \cdot Y' + X \cdot Y' = (X + X') \cdot Y' = Y'$
- We cannot combine product term 3 with other product terms.
- It follows that the minimized expression of function $F = \sum_{X,Y,Z}(0, 1, 4, 5, 6)$ is $F = Y' + X \cdot Z'$
- Compared with algebraic processing, the function obtained using Karnaugh map has one less literal, which means a cheaper implementation
- We added the minterm 4 twice, but this is ok, because $1 + 1 = 1$

Example 2

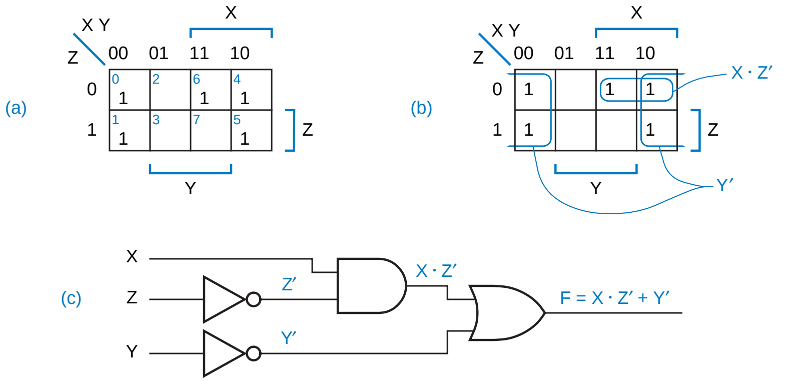


Figure 4 : $F = \sum_{X,Y,Z}(0,1,4,5,6)$; (a) initial Karnaugh map; (b) Karnaugh map with circled product terms; (c) AND/OR circuit

Minimizing with Karnaugh Maps: Rules

Rule for combining 1-cells and forming product term: A set of 2^i cells containing 1 may be combined if there are i variable of the logic function that take all the 2^i possible combinations in the set, and the remaining $n - i$ variables do not change values within the set. The resulting product term has $n - i$ literals. A variable will appear complemented in the resulting product term if it takes the value 0 in all cells within the set, and uncomplemented if it takes the value 1 in all the cells from the set.

Minimizing with Karnaugh Maps: Rules

Graphic representation of the rule:

- Circle rectangular sets of 2^i 1-cells, including rectangles that wraparound at the edges of the map.
- For each variable use the following rules:
    - If the circle covers an area where the variable takes only the value 0, then the variable appears complemented in the product term
    - If the circle covers an area where the variable takes only the value 1, then the variable will be uncomplemented in the product term
    - If the circle covers an area where the variable takes both the value 0 and the value 1, then the variable will not appear in the product term
- **A sum of products expression for a function must contain product terms that cover all the cells where the function takes the value 1 and none of the cells where the function takes the value 0.**

Definitions

Definition

A *minimal sum* of a logic function $F(X_1, \ldots, X_n)$ is a sum of products expression for F such that no sum-of-products expression for F has fewer product terms, and any sum-of-products expression with the same number of product terms has at least as many literals.

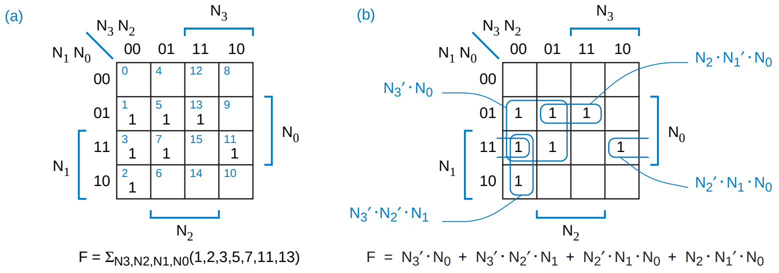
According to the criteria for combinational-circuit minimization, a minimal sum will realize a minimal circuit because:

1. The minimal sum has the fewest possible product terms, which means fewest possible first-level gates and second level-gates inputs
2. Within previous condition (fewest possible product terms), it has the fewest possible literals, meaning the fewest possible first-level gate inputs

Example 3: Prime-number detector

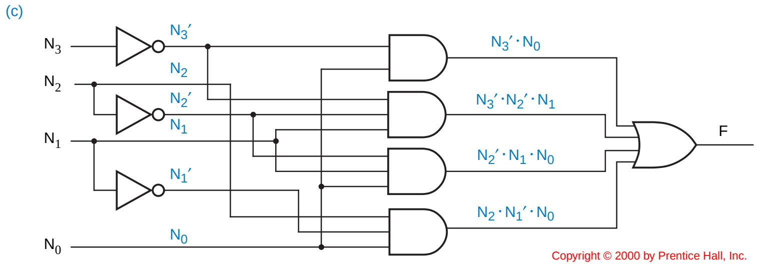
In figure 5 we minimze the prime-number detector using Karnaugh maps.

The resulting circuit is the only circuit, among those obtained for the 4-bits prime-number detector problem, that realizes a minimal sum.



(a)
$N_3 N_2$
$N_1 N_0$
N_3
N_0
N_1
N_2
$F = \Sigma_{N3,N2,N1,N0}(1,2,3,5,7,11,13)$
(b)
$N_3' \cdot N_0$
$N_2 \cdot N_1' \cdot N_0$
$N_2' \cdot N_1 \cdot N_0$
$N_3' \cdot N_2' \cdot N_1$
$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0 + N_2 \cdot N_1' \cdot N_0$

$N_1 N_0 \backslash N_3 N_2$	00	01	11	10
00	0	4	12	8
01	1: 1	5: 1	13: 1	9
11	3: 1	7: 1	15	11: 1
10	2: 1	6	14	10



(c)
N_3
N_2
N_1
N_0
N_3'
N_2
N_2'
N_1
N_1'
N_0
$N_3' \cdot N_0$
$N_3' \cdot N_2' \cdot N_1$
$N_2' \cdot N_1 \cdot N_0$
$N_2 \cdot N_1' \cdot N_0$
F

Figure 5 : Prime-number detector; (a) initial Karnaugh map; (b) circled product terms; (c) minimized circuit

More Definitions

Definition

A logic function $P(X_1, \ldots, X_n)$ *implies* a logic function $F(X_1, \ldots, X_n)$ if for every input combination such that $P = 1$, then $F = 1$ also.

In other words, if P implies F, then $F = 1$ for every input combination for which $P = 1$ and possible for more input combinations.

Notation: $P \Rightarrow F$.

We may say also that "F includes P" or "F covers P"

More Definitions: Prime Implicant

Definition

A *prime implicant* of a logic function $F(X_1, \ldots, X_n)$ is a normal product term $P(X_1, \ldots, X_n)$ that implies F, such that if any variable is removed from P, then the resulting product term does not imply F.

In terms of Karnaugh maps, a prime implicant is a circled set of 1-cells satisfying the combining rule, that cannot be extended (i.e., make it cover twice as many cells) without covering one or more 0s. In the definition, removing a variable from the prime implicant means trying to make the prime implicant larger (to double the number of covered cells).

Prime-Implicant Theorem

Tells us when to stop when we want to find a minimal sum of a logic function.

Theorem

Prime-Implicant Theorem: A minimal sum is a sum of prime implicants.

It means that, when trying to find a minimal sum, we do not need to consider product terms that are not prime implicants.
Proof of the Prime-Implicant Theorem: by contradiction.

- Suppose that a product term P in a "minimal" sum is not a prime implicant.
- According to the definition of a prime implicant, if P is not 1, we can remove some literals from P and obtain a product term P^* that still implies F.
- Replacing P with P^* in the "minimal" sum, the resulting sum has one fewer literal and still equals F.
- Which means that the "minimal" sum was not minimal.

Example 4

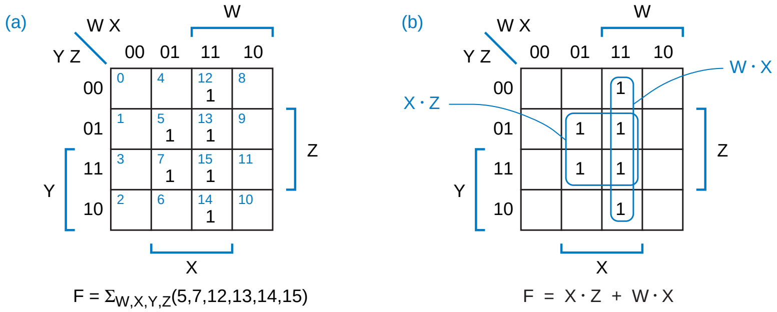


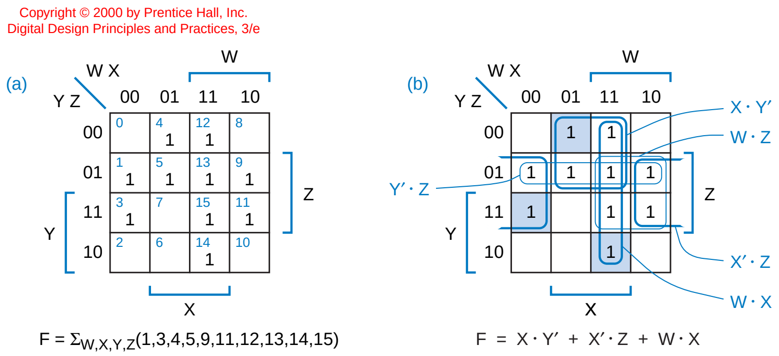
Figure 6 : $F = \sum_{W,X,Y,Z}(5, 7, 12, 13, 14, 15)$ (a) Karnaugh map; (b) prime implicants

Another example of minimization. There are only 2 prime implicants, the minimal sum includes both of them in order to cover all 1-cells.

Example 5: Complete Sum and Minimal Sum

- The sum of all prime implicants is called the *complete sum*.
- The complete sum is a realization of a function, but it is not necessarily a minimal sum.
- The logic function from example 5, figure 7 has 5 prime implicants: $X \cdot Y'$, $X' \cdot Z$, $W \cdot X$, $W \cdot Z$, $Y' \cdot Z$
- The minimal sum includes only the first three of them: $F = X \cdot Y' + X' \cdot Z + W \cdot X$
- We need a systematic way to determine which prime implicants to include in the minimal sum and which to leave out.
- First, we need two more definitions: distinguished 1-cells and essential prime implicants.

Example 5



Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e
(a)
W X
Y Z
W
00 01 11 10
Z
Y
X
F = Σ_{W,X,Y,Z}(1,3,4,5,9,11,12,13,14,15)
(b)
X · Y′
W · Z
Y′ · Z
X′ · Z
W · X
F = X · Y′ + X′ · Z + W · X

Figure 7 : $F = \sum_{W,X,Y,Z}(1,3,4,5,9,11,12,13,14,15)$ (a) Karnaugh map; (b) prime implicants and distinguished 1-cells

Essential Prime Implicants: Definitions

Definition

A *distinguished 1-cell* of a logic function is an input combination that is covered by only one prime imlicant.

Definition

An *essential prime implicant* is a prime implicant that covers one or more distinguished 1-cells.

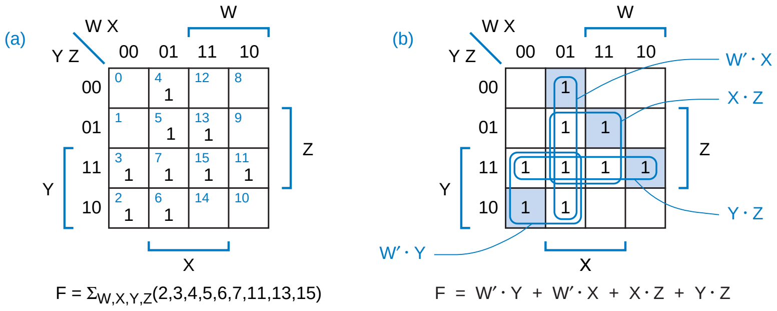
In figure 7 the distinguished 1-cells are shadowed and the essential prime implicants are circled with heavier lines.

Essential Prime Implicants

Procedure for selecting the prime implicants for a minimal sum:

- Since an essential prime implicant is the only prime implicant that covers some cells, it must be included in the minimal sum.
- Step 1: Identify all distinguished 1-cells and the corresponding essential prime implicants and include them in the minimal sum.
- Step 2: Cover the remaining 1-cells (if any), that were not covered by essential prime implicants.
- In example 5 from figure 7 all 1-cells are covered by essential prime implicants, hence we stop here.

Example 6



(a)
W X
Y Z
W
00
01
11
10
Z
Y
X
0
4
12
8
1
5
13
9
3
7
15
11
2
6
14
10
1
F = Σ_{W,X,Y,Z}(2,3,4,5,6,7,11,13,15)
(b)
W′ · X
X · Z
Y · Z
W′ · Y
F = W′ · Y + W′ · X + X · Z + Y · Z

Figure 8 : $F = \sum_{W,X,Y,Z}(2,3,4,5,6,7,11,13,15)$ (a) Karnaugh map; (b) prime implicants and distinguished 1-cells

In figure 8 it is another example where all of the 1-cells are covered by essential prime implicants.

Example 7

- A logic function where not all the 1-cells are covered by essential prime implicants is given in example 7, figure 9
- After removing the essential prime implicants and the 1-cells covered by them, we obtain a *reduced map*.
- In this example the reduced map contains only one 1-cell, that is covered by two prime-implicants: $W' \cdot Z$ and $X \cdot Y \cdot Z$
- We chose $W' \cdot Z$ because it has a lower cost (fewer inputs).
- In general, for more complex cases, we need one more definition.

Example 7

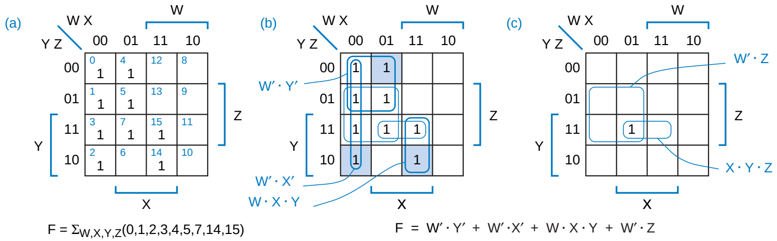


Figure 9 : $F = \sum_{W,X,Y,Z}(0, 1, 2, 3, 4, 5, 7, 14, 15)$ (a) Karnaugh map; (b) prime implicants and distinguished 1-cells; (c) reduced map after removal of essential prime implicants and covered 1-cells

Definitions for Reduced Maps

Definition

Given two prime implicants P and Q in a reduced map, P is said to *eclipse* Q (written $P \supseteq Q$) if P covers at least all the 1-cells covered by Q.

If P costs no more than Q and eclipses Q, then P is at least as good as Q and we can remove Q from consideration from finding a minimal sum

An example of eclipsing is given in example 8 from figure 10:

- After removing the essential prime implicants, two 1-cells remain.
- Each of them is covered by two prime implicants.
- The prime implicant $X \cdot Y \cdot Z$ eclipses the other two prime implicants, which can then be removed from consideration
- We must include the *secondary essential prime implicant* $X \cdot Y \cdot Z$ in the minimal sum, because it covers both 1-cells remained in the reduced map.

Example 8

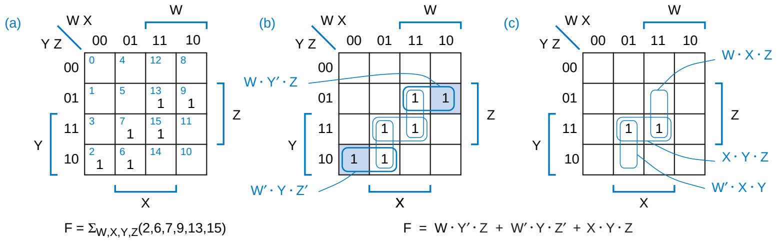
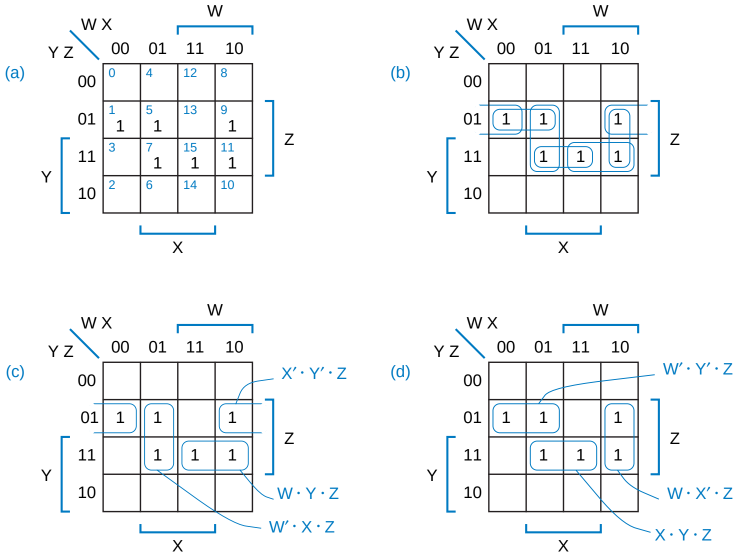


Figure 10 : $F = \sum_{W,X,Y,Z}(2,6,7,9,13,15)$ (a) Karnaugh map; (b) prime implicants and distinguished 1-cells; (c) reduced map after removal of essential prime implicants and covered 1-cells

Example 9

- A more difficult case is shown in example 9, figure 11.
- The logic function $F = \sum_{W,X,Y,Z}(1,5,7,9,11,15)$ has no essential prime implicants.
- By trial and error we can find two different minimal sums for this function:
  1. $F = X' \cdot Y' \cdot Z + W' \cdot X \cdot Z + W \cdot Y \cdot Z$
  2. $F = W' \cdot Y' \cdot Z + W \cdot X' \cdot Z + X \cdot Y \cdot Z$



F = W′ · X · Z + W · Y · Z + X′ · Y′ · Z

F = X · Y · Z + W · X′ · Z + W′ · Y′ · Z

Figure 11 : Example 9: $F = \sum_{W,X,Y,Z}(1, 5, 7, 9, 11, 15)$ (a) Karnaugh map; (b) prime implicants; (c) a minimal sum; (d) another minimal sum

Outline

Combinational-Circuit Synthesis
Karnaugh Maps
Minimizing Sums of Products
Karnaugh Maps for Five and Six Variables Logic Functions
“Don’t Care” Input Combinations
Simplifying Product of Sums

Timing Hazards
Static Hazards
Finding Static Hazards Using Maps
Dynamic Hazards

Karnaugh Maps for Five and Six-Variable Logic Functions

- A Karnaugh map for a 5-variable logic function can be drawn like in figure 12
- Cells that occupy the same relative position in the $V = 0$ and $V = 1$ submaps are considered to be adjacent
- A Karnaugh map for a 6-variable logic function can be drawn like in figure 13
- Cells that occupy the same relative positions in adjacent submaps are considered to be adjacent
- There are also other graphical representations of 5- and 6-variable Karnaugh maps
- For more than 6 variable functions (and even for 5- and 6-variable functions) it is difficult to visualize the prime implicants.

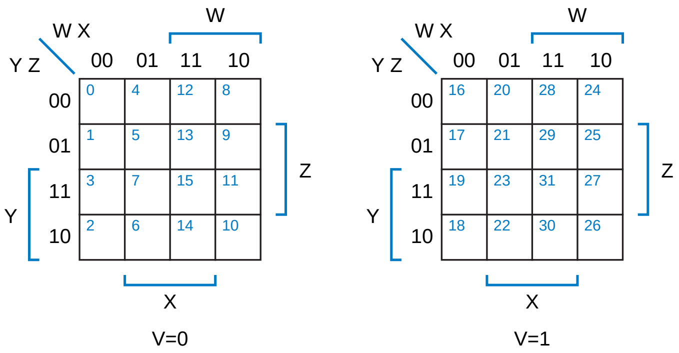


Figure 12 : Karnaugh map for 5-variable logic functions

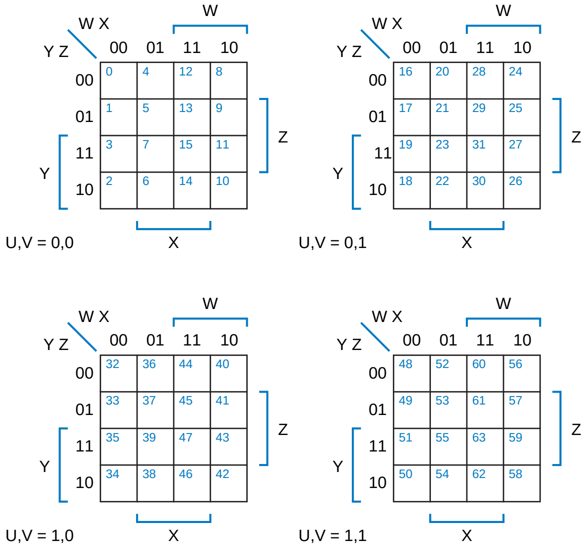


Figure 13 : Karnaugh map for 6-variable logic functions

Outline

Combinational-Circuit Synthesis

Karnaugh Maps

Minimizing Sums of Products

Karnaugh Maps for Five and Six Variables Logic Functions

“Don’t Care” Input Combinations

Simplifying Product of Sums

Timing Hazards

Static Hazards

Finding Static Hazards Using Maps

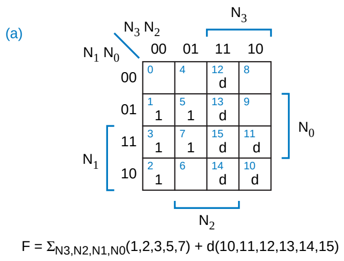
Dynamic Hazards

“Don’t Care” Input Combinations

- Sometimes in the specifications of a circuit the output doesn’t matter for certain input combinations, called don’t cares
- This is because these input combinations never occur or because the output really does’n matter for certain input combinations
- In the function that describes the circuit there is a *d-list*, or *d-set*, the list of don’t care input combinations
- In the Karnaugh map we put a *d* value on these positions.
- In the minimization procedure it is not necessary to include the *d*’s in the product terms, but we can include them if it helps reducing the cost.
- An example is given in figure 14: the prime-number detector for BCD (binary codded decimally) numbers.

“Don’t Care” Input Combinations

- BCD means that the combinations corresponding to decimal numbers between 10 and 15 (or hexadecimal numbers between A and F) never occur.
  - For example, if we count in base 16, we have the sequence $0, 1, 2, \ldots, 8, 9, A, B, \ldots, E, F$, which corresponds to the binary numbers $0000, 0001, 0010, \ldots, 1000, 1001, 1010, 1011, \ldots, 1110, 1111$
  - If we count in BCD, we have the sequence $0, 1, 2, \ldots, 8, 9, 10, 11, \ldots, 14, 15$, which corresponds to the binary numbers $0000, 0001, 0010, \ldots 1000, 1001,$ $0001\ 0000, 0001\ 0001, \ldots, 0001\ 0100, 0001\ 0101$
- In the Karnaugh map we put d’s on the cells 10-15.
- The procedure for circling prime implicants changes as follows:
  1. Allow d’s to be included when we circle 1’s, in order to increase the size of the product term (and hence to reduce the number of variables in the corresponding prime implicants)
  2. Do not circle sets that contain only d’s, this would only increase the cost by adding product terms
  3. Do not circle any 0 !



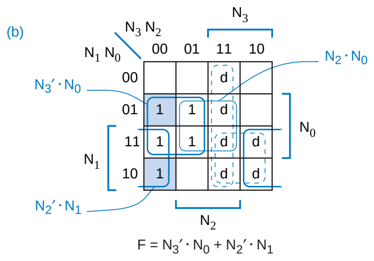


Figure 14 : Prime BCD detector (a) initial Karnaugh map; (b) Karnaugh map with prime implicants and distinguished 1-cells

Outline

Combinational-Circuit Synthesis

- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- “Don’t Care” Input Combinations
- Simplifying Product of Sums

Timing Hazards

- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Simplifying Product of Sums

We use principle of duality. There are two methods

1. First method:
   1.1 We look at 0s in Karnaugh map, since each 0 corresponds to a maxterm in the canonical product of the function
   1.2 Follow the procedure learned for minimizing sum of products in a dual way !
2. The second method is easier:
   2.1 First complement the logic function F and obtain F'.
   2.2 Find a minimal sum for F' using the known method
   2.3 Complement the minimal sum of F' and we will obtain the minimal product of F, using DeMorgan's theorems

Other Topics Concerning Combinational Circuits Minimization

- The Karnaugh maps are good for minimizing “by hand”
- There are programmed minimization methods, that are more suitable for programming on computers
- One such programmed method is the Quine-McCluskey method, guaranteed to find “minimal” solution
- Works well for up to 8-12 variables, but for more variables its complexity blows up
- In real-world problems are used heuristics, that find a “good enough” (“almost minimal”) solution in a reasonable time using well-educated guesses
- Another issue is the minimization of the multiple-output functions: there are formal methods to perform multiple-output minimization using Karnaugh maps, but of high complexity (some methods use multivalued logic)

Outline

Combinational-Circuit Synthesis
Karnaugh Maps
Minimizing Sums of Products
Karnaugh Maps for Five and Six Variables Logic Functions
"Don't Care" Input Combinations
Simplifying Product of Sums

Timing Hazards
Static Hazards
Finding Static Hazards Using Maps
Dynamic Hazards

Introduction

- The circuit analysis that we performed earlier predicts only *steady-state behaviour* of combinational circuits.
- It means that it predicts the output of the circuit as a function of its inputs under the assumption that the inputs have been stable for a long period.
- Also, we ignored circuits' delay.
- *Transient behaviour*: we consider the changes (transitions) that take place in the circuit
- The predictions of the transient behaviour can differ from these of the steady state analysis
- The output of a circuit can produce a short impulse, called *glitch*, when steady-state analysis predicts that the output of the circuit should not change.
- A *hazard* is said to exist when the circuit has the possibility of producing a glitch.

Introduction

- It does not mean that the glitch will appear for sure, it depends on the circuit's delays and on other electrical characteristics of the circuit.
- Those factors are difficult, if not impossible to control
- Maybe a glitch will occur in the worst case combination of logical and electrical conditions
- We want to eliminate the hazards, i.e. the *possibility* of apparition of glitches
- Depending on how the output of the circuit is used, the glitches can negatively impact the functioning of the system that contains the circuit.
- Hazards are not harmful if we use properly designed synchronous circuits, i.e., if we store the outputs of a combinational circuit only after is become stable.

Outline

Combinational-Circuit Synthesis

- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- “Don’t Care” Input Combinations
- Simplifying Product of Sums

Timing Hazards

- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Static Hazards. Static-1 Hazard

Definition

A static-1 hazard is a pair of input combination that:

1. differ in only one input variable and
2. both give a 1 output

such that it is possible for a momentary 0 output to occur during a transition in the differing input variable.

Example: circuit from figure 15 (a)

- Suppose that $X = Y = 1$ and that Z changes from 1 to 0
- We assume that the propagation delay on each gate (or inverter) is one time unit
- Figure 15 (b) shows the timing diagram
- Static analysis predicts that circuit's output F is 1 for both input combinations: $XYZ = 111$ and $XYZ = 110$
- However, $F = 0$ for one time unit during the $1 \to 0$ transition on Z, because of the delay on the inverter gate (with input Z and output ZP)

Static Hazards. Static-1 Hazard

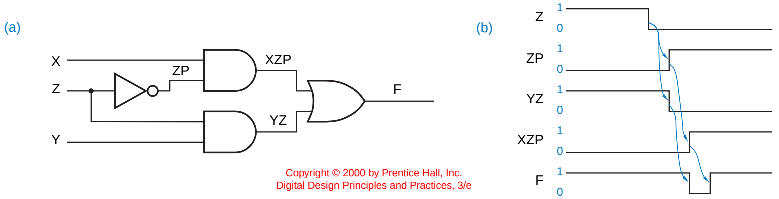

(a)
X
Z
Y
ZP
XZP
YZ
F
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e
(b)
Z
ZP
YZ
XZP
F
1
0

Figure 15 : Circuit with a static-1 hazard (a) logic diagram; (b) timing diagram

Static Hazards. Static-0 Hazard

Definition

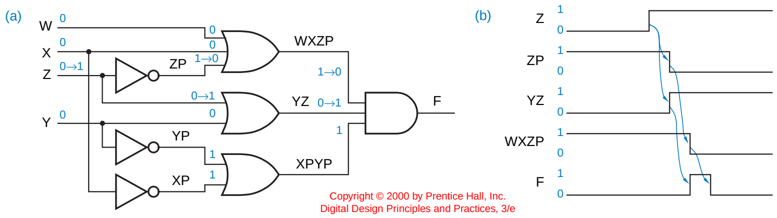
A static-0 hazard is a pair of input combination that:

1. differ in only one input variable and
2. both give a 0 output

such that it is possible for a momentary 1 output to occur during a transition in the differing input variable.

Static-0 hazard is the dual of static-1 hazard, hence the dual of the circuit from figure 15 would have a static-0 hazard.
Figure 16 (a) presents a circuit with static-0 hazards. One of the hazards is shown in figure 16 (b), which gives the timing diagram for the case when $WXY = 000$ and Z changes from 0 to 1.

Static Hazards. Static-0 Hazard



(a)
W
X
Z
Y
0
0
0→1
0
ZP
YP
XP
1→0
0→1
WXZP
YZ
XPYP
1→0
0→1
1
F
(b)
Z
ZP
YZ
WXZP
F
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

Figure 16 : Circuit with static-0 hazards (a) logic diagram; (b) timing diagram

Outline

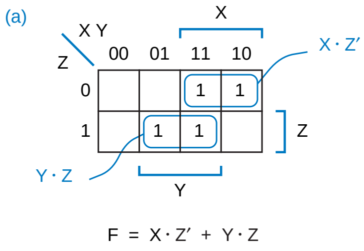
Combinational-Circuit Synthesis
- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- “Don’t Care” Input Combinations
- Simplifying Product of Sums

Timing Hazards
- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Finding Static Hazards Using Maps

- Karnaugh maps can be used to detect and remove hazards
- Static-0 hazard can appear in a circuit that implements a sum of products only if an AND gate contains a variable and its complement, but this would be nonsense
- Hence, a properly designed two-level AND-OR circuit cannot have static-0 hazards, but it can have static-1 hazards
- If we look at the Karnaugh map that from figure 17 (a), that corresponds to the circuit from figure 15, we can observe that:
  - There is no product term that covers both input combinations $XYZ = 111$ and $XYZ = 110$
  - Then, intuitively it is possible for the output to "glitch" to 0 momentarily if the AND gate that covers one of the two input combination goes to 0 before the output gate that covers the other combination goes to 1
  - Also intuitively, the solution would be to add the extra product term (an AND gate) that covers the combination $XY = 11$
  - Figure 17 (b) gives the solution, and the resulted circuit is in figure 18

Finding Static Hazards Using Maps



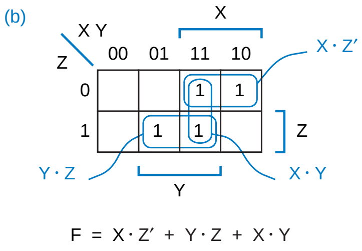


Figure 17 : Karnaugh map for the circuit of figure 15: (a) as originally designed; (b) with static-1 hazard eliminated

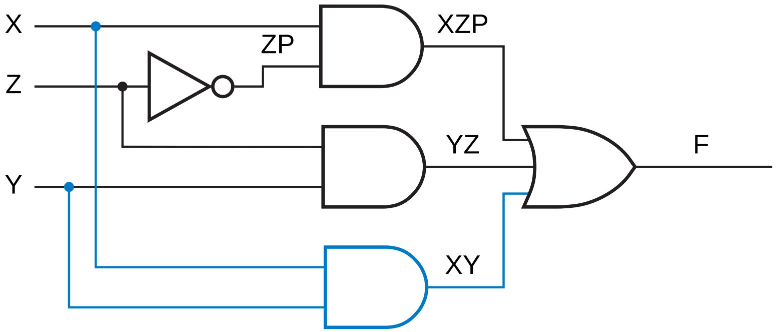
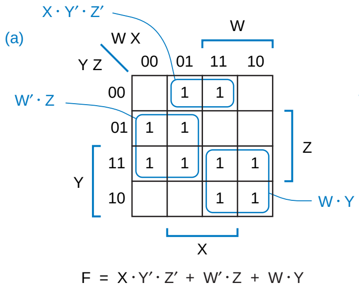


Figure 18 : Circuit with static-1 hazard eliminated

Finding Static Hazards Using Maps

- The extra product term is the *consensus* of the two original terms
- In general, we must add consensus terms to eliminate hazards
- Figure 19 shows an example where three product terms must be added to eliminate hazard
- A properly designed two-level OR-AND circuit (product of sums) has no static-1 hazards, but it can have static-0 hazards
- These static-0 hazards can be detected and eliminated if we study the adjacent 0s in Karnaugh maps, in a dual way to what was presented for static 1 hazards.



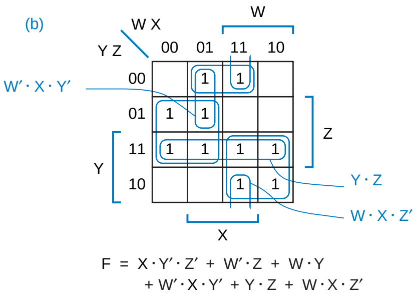


Figure 19 : Karnaugh map for another sum-of-products circuit: (a) as originally designed; (b) with extra product terms to cover static-1 hazards

Outline

Combinational-Circuit Synthesis
- Karnaugh Maps
- Minimizing Sums of Products
- Karnaugh Maps for Five and Six Variables Logic Functions
- “Don’t Care” Input Combinations
- Simplifying Product of Sums

Timing Hazards
- Static Hazards
- Finding Static Hazards Using Maps
- Dynamic Hazards

Dynamic Hazards

Definition
A *dynamic hazard* is the possibility of an output to change more than once as the result of a single input transition.

Multiple output transitions occur if there are multiple paths with different delays from the changing input to the changing output.

Example: circuit from figure 20, where there are three different paths from input X to the output F:

1. the first path goes through a slow OR gate,
2. the second path goes through an even slower OR gate
3. the third path goes through very fast gates.

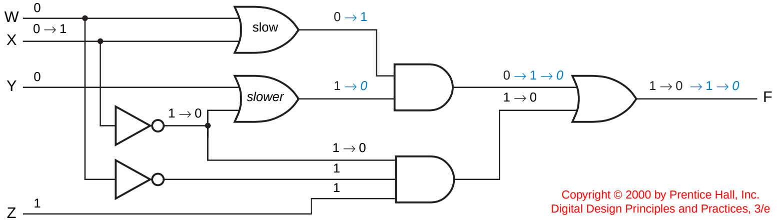
Except the two OR gate (slow and slower), we consider that all other gates are very fast.

Dynamic Hazards

1. We consider that the input $WXYZ = 0001$, which means that the output F will be 1.
2. We change the input X to 1.
3. The transitions through the very fast gates (shown in black) take place next and the output goes to 0
4. Then the output of the slow gate changes from 0 to 1, followed by the output of the next AND gate and then by the output F (all transitions in non-italic blue)
5. After that the slower gate changes from 1 to 0, followed by the output of the next AND gate and then by the output F, that reaches the final 0 value (all these transitions in italic blue)

Dynamic hazards do not occur in properly designed two-level AND-OR or OR-AND circuits, they can appear only in multilevel circuits.
A properly designed circuit is a circuit where no variable and its complement are connected to the same first-level gate.

Dynamic Hazards



W
0
X
0 → 1
Y
0
Z
1
slow
slower
0 → 1
1 → 0
1 → 0
1 → 0
1
1
0 → 1 → 0
1 → 0
1 → 0 → 1 → 0
F
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

Figure 20 : Circuit with a dynamic hazard