ENGIN 112

Intro to Electrical and Computer Engineering

Lecture 9

More Karnaugh Maps and Don’t Cares
Overview

° Karnaugh maps with four inputs
  • Same basic rules as three input K-maps

° Understanding prime implicants
  • Related to minterms

° Covering all implicants

° Using Don’t Cares to simplify functions
  • Don’t care outputs are undefined

° Summarizing Karnaugh maps
Karnaugh Maps for Four Input Functions

- Represent functions of 4 inputs with 16 minterms
- Use same rules developed for 3-input functions
- Note bracketed sections shown in example.

![Karnaugh Map Diagram](image)

Fig. 3-8 Four-variable Map
Karnaugh map: 4-variable example

\[ F(A,B,C,D) = \Sigma m(0,2,3,5,6,7,8,10,11,14,15) \]

\[ F = \]

\[ C + A'B'D + B'D' \]

Solution set can be considered as a coordinate System!
Design examples

LT = $A' B' D' + A' C + B' C D$

EQ = $A'B'C'D' + A'BC'D + ABCD + AB'CD'$

GT = $B C' D' + A C' + A B D'$

Can you draw the truth table for these examples?
Physical Implementation

- Step 1: Truth table
- Step 2: K-map
- Step 3: Minimized sum-of-products
- Step 4: Physical implementation with gates

K-map for EQ

A
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

B
C
D

SEQ

ENGIN112  L9: More Karnaugh Maps
September 22, 2003
Karnaugh Maps

° Four variable maps.

\[
\begin{array}{c|cccc}
C & D & 00 & 01 & 11 & 10 \\
\hline
A & B & 00 & 01 & 11 & 10 \\
\hline
00 & 0 & 0 & 0 & 1 & \\
01 & 1 & 1 & 0 & 1 & \\
11 & 1 & 1 & 1 & 1 & \\
10 & 1 & 0 & 1 & 1 & \\
\end{array}
\]

\[F = A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot \overline{C} + A \cdot B \cdot C + A \cdot B \cdot \overline{C}
\]

\[F = \overline{B} \cdot C + D + \overline{A} \cdot C + A \cdot D
\]

° Need to make sure all 1’s are covered

° Try to minimize total product terms.

° Design could be implemented using NANDs and NORs
Karnaugh maps: Don’t cares

- In some cases, outputs are undefined
- We “don’t care” if the logic produces a 0 or a 1
- This knowledge can be used to simplify functions.

- Treat X’s like either 1’s or 0’s
- Very useful
- OK to leave some X’s uncovered

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Karnaugh maps: Don’t cares

\[ f(A,B,C,D) = \Sigma m(1,3,5,7,9) + d(6,12,13) \]

- without don't cares

\[ f = \]

\[ A'D + C'D \]

\[ \begin{array}{cccc|c}
 A & B & C & D & f \\
 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 1 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 \\
 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 1 \\
 0 & 1 & 1 & 0 & X \\
 0 & 1 & 1 & 1 & 1 \\
 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 1 & 1 \\
 1 & 0 & 1 & 0 & 0 \\
 1 & 0 & 1 & 1 & 0 \\
 1 & 1 & 0 & 0 & X \\
 1 & 1 & 0 & 1 & X \\
 1 & 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 & 0 \\
\end{array} \]
Don’t Care Conditions

° In some situations, we don’t care about the value of a function for certain combinations of the variables.
  • these combinations may be impossible in certain contexts
  • or the value of the function may not matter in when the combinations occur

° In such situations we say the function is *incompletely specified* and there are multiple (completely specified) logic functions that can be used in the design.
  • so we can select a function that gives the simplest circuit

° When constructing the terms in the simplification procedure, we can choose to either cover or not cover the don’t care conditions.
Map Simplification with Don’t Cares

\[ F = A \bar{C} \bar{D} + B + A \bar{C} \]

**Alternative covering.**

\[ F = A \bar{B} \bar{C} \bar{D} + A B C + B C + A \bar{C} \]
Karnaugh maps: don’t cares (cont’d)

- \(f(A,B,C,D) = \sum m(1,3,5,7,9) + d(6,12,13)\)
  - \(f = A'D + B'C'D\) without don't cares
  - \(f = A'D + C'D\) with don't cares

Don’t cares can be treated as 1s or 0s depending on which is more advantageous.

By using don’t care as a "1," a 2-cube can be formed rather than a 1-cube to cover this node.
Definition of terms for two-level simplification

° Implicant
  • Single product term of the ON-set (terms that create a logic 1)

° Prime implicant
  • Implicant that can't be combined with another to form an implicant with fewer literals.

° Essential prime implicant
  • Prime implicant is essential if it alone covers a minterm in the K-map
  • Remember that all squares marked with 1 must be covered

° Objective:
  • Grow implicant into prime implicants (minimize literals per term)
  • Cover the K-map with as few prime implicants as possible (minimize number of product terms)
Examples to illustrate terms

6 prime implicants:
- $A'B'D$, $BC'$, $AC$, $A'C'D$, $AB$, $B'CD$

minimum cover: $AC + BC' + A'B'D$

5 prime implicants:
- $BD$, $ABC'$, $ACD$, $A'BC$, $A'C'D$

essential

minimum cover: 4 essential implicants
Prime Implicants

Any single 1 or group of 1s in the Karnaugh map of a function $F$ is an implicant of $F$.

A product term is called a prime implicant of $F$ if it cannot be combined with another term to eliminate a variable.

Example:

If a function $F$ is represented by this Karnaugh Map. Which of the following terms are implicants of $F$, and which ones are prime implicants of $F$?

(a) $AC'D'$
(b) $BD$
(c) $A'B'C'D'$
(d) $AC'$
(e) $B'C'D'$

Implicants: (a),(c),(d),(e)

Prime Implicants: (d),(e)
Essential Prime Implicants

A product term is an essential prime implicant if there is a minterm that is only covered by that prime implicant.

- The minimal sum-of-products form of \( F \) must include all the essential prime implicants of \( F \).

Fig. 3-11  Simplification Using Prime Implicants
Summary

- K-maps of four literals considered
  - Larger examples exist
- Don’t care conditions help minimize functions
  - Output for don’t cares are undefined
- Result of minimization is minimal sum-of-products
- Result contains prime implicants
- Essential prime implicants are required in the implementation