Overview

- Last lecture
  - Logic simplification
    - Boolean cubes
    - Karnaugh maps
- Today
  - Incompletely specified functions
  - Design examples
  - k-maps for POS minimization

Incompletely specified functions

- Functions of n inputs have $2^n$ possible configurations
  - Some combinations may be unused
  - Call unused combinations “don’t cares”
  - Exploit don’t cares during logic minimization
  - Don’t care $\not=0$ no output
- Example: A BCD increment-by-1
  - Function $F$ computes the next number in a BCD sequence
    - If the input is 00102, the output is 00112
    - BCD encodes decimal digits 0–9 as 00002–10012
    - Don’t care about binary numbers 10102–11112

Truth table for a BCD increment-by-1

<table>
<thead>
<tr>
<th>INPUTS A B C D</th>
<th>OUTPUTS W X Y Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 0 0 0</td>
</tr>
</tbody>
</table>

- Off-set for W: m0–m6, m9
- On-set for W: m7 and m8
- Don’t care set for W: We don’t care about the output values

Notation

- Don’t cares in canonical forms
  - Three distinct logical sets
    - {on}, {off}, {don’t care}
- Canonical representations of a BCD increment-by-1
  - Minterm expansion
    - $W = m7+m8+d10+d11+d12+d13+d14+d15$
  - Maxterm expansion
Karnaugh maps and don't cares

- Can treat don't cares as 0s or 1s
  - Depending on which is more advantageous
- \(F(A,B,C,D) = \Sigma m(1, 3, 5, 7, 9) + d(6, 12, 13)\)
  - Example: Minimize \(F\) with and without don't cares

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Design example: A two-bit comparator

Need a 4-variable Karnaugh map for each of the 3 output functions

Two-bit comparator (con't)
Two-bit comparator (con’t)

LT = A'B'D + A'C + B'CD
GT = BC'D' + AC' + ABD'

EQ = A'B'C'D' + A'BCD + ABCD + AB'CD'
= (A xnor C) • (B xnor D)

Two-bit comparator design

EQ
LT
GT

Design example: Two-bit multiplier

Need a 4-variable Karnaugh map for each of the 4 output functions
Two-bit multiplier design

- Draw the circuit schematic
  - $P_0 = A_1A_2B_2B_1$
  - $P_1 = A_1A_2B_2'$ + $A_1A_2B_1$
  - $P_2 = A_1'A_0B_2 + A_1B_2B_1' + A_1B_2B_1' + A_0A_1B_1$
  - $P_3 = A_1B_1$

Design example: BCD increment by 1

Need a 4-variable Karnaugh map for each of the 4 output functions
BCD increment by 1 (con’t)

We greatly simplify the logic by using the don’t cares.

BCD increment by 1 design

- Draw the circuit schematic
  - \( O_8 = I_1I_1I_1 + I_1I_1' \)
  - \( O_4 = I_1I_1' + I_1'I_1 + I_1I_1 \)
  - \( O_2 = I_1'I_1'I_1 + I_1'I_1' \)
  - \( O_1 = I_1'I_1 \)

Loose end: POS minimization using k-maps

- Using k-maps for POS minimization
  - Encircle the zeros in the map
  - Interpret indices complementary to SOP form

\[
\begin{array}{c|c|c|c}
AB & 00 & 01 & 11 \\
\hline
CD & 00 & 01 & 11 \\
\hline
 00 & 1 & 0 & 0 \\
01 & 0 & 1 & 0 \\
11 & 1 & 1 & 1 \\
\end{array}
\]

\[
F = (B'+C+D)(B+C+D')(A'+B'+C)
\]

Check using de Morgan’s on SOP

\[
F' = BCD' + B'C'D + ABC'
\]

\[
(F')' = (BC'D' + B'C'D + ABC')'
\]

\[
F = (BC'D')(B'C'D')(ABC')'
\]

\[
F = (B'+C+D)(B+C+D')(A'+B'+C)
\]