Combinational logic optimization

- Alternate representations of Boolean functions
  - cubes
  - karnaugh maps
- Simplification
  - two-level simplification
  - exploiting don’t cares
  - algorithm for simplification

Simplification of two-level combinational logic

- Finding a minimal sum of products or product of sums realization
  - exploit don’t care information in the process
- Algebraic simplification
  - not an algorithmic/systematic procedure
  - how do you know when the minimum realization has been found?
- Computer-aided design tools
  - precise solutions require very long computation times, especially for functions with many inputs (> 10)
  - heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- Hand methods still relevant
  - to understand automatic tools and their strengths and weaknesses
  - ability to check results (on small examples)
The uniting theorem

- **Key tool to simplification:** $A (B' + B) = A$
- **Essence of simplification of two-level logic:**
  - find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

$$F = A'B' + AB' = (A' + A)B' = B'$$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- B has the same value in both on-set rows
  - B remains
- A has a different value in the two rows
  - A is eliminated

Boolean cubes

- **Visual technique for identifying when the uniting theorem can be applied**
- **n input variables = n-dimensional "cube"**

- 1-cube
  - $01$
  - $11$
  - 00
  - 10

- 2-cube
  - 00
  - 11

- 3-cube
  - 000
  - 111

- 4-cube
  - 0000
  - 1111
Mapping truth tables onto Boolean cubes

Uniting theorem combines two "faces" of a cube into a larger "face"

Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ON-set = solid nodes
OFF-set = empty nodes
DC-set = $\times$d nodes

Two faces of size 0 (nodes) combine into a face of size 1 (line)

A varies within face, B does not
This face represents the literal B'

Three variable example

Binary full-adder carry-out logic

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that "111" is covered three times

Cout = BCin + AB + ACin
Higher dimensional cubes

- Sub-cubes of higher dimension than 2

\[ F(A,B,C) = \Sigma m(4,5,6,7) \]

on-set forms a square
i.e., a cube of dimension 2

represents an expression in one variable
i.e., 3 dimensions – 2 dimensions

A is asserted (true) and unchanged
B and C vary

This subcube represents the literal A

m-dimensional cubes in a n-dimensional Boolean space

- In a 3-cube (three variables):
  - a 0-cube, i.e., a single node, yields a term in 3 literals
  - a 1-cube, i.e., a line of two nodes, yields a term in 2 literals
  - a 2-cube, i.e., a plane of four nodes, yields a term in 1 literal
  - a 3-cube, i.e., a cube of eight nodes, yields a constant term "1"

- In general,
  - an m-subcube within an n-cube (m < n) yields a term with n – m literals
Karnaugh maps

- Flat map of Boolean cube
  - wrap-around at edges
  - hard to draw and visualize for more than 4 dimensions
  - virtually impossible for more than 6 dimensions
- Alternative to truth-tables to help visualize adjacencies
  - guide to applying the uniting theorem
  - on-set elements with only one variable changing value are adjacent
    unlike the situation in a linear truth-table

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{F} \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

Karnaugh maps (cont'd)

- Numbering scheme based on Gray–code
  - e.g., 00, 01, 11, 10
  - only a single bit changes in code for adjacent map cells

\[
\begin{array}{cccc}
\text{A} & \text{B} & \text{C} & \text{D} \\
0 & 4 & 12 & 8 \\
1 & 5 & 13 & 9 \\
3 & 7 & 15 & 11 \\
2 & 6 & 14 & 10 \\
\end{array}
\]

\[13 = 1101 = ABC'D\]
Adjacencies in Karnaugh maps

- Wrap from first to last column
- Wrap top row to bottom row

Karnaugh map examples

- \( F = \) 
- \( \text{Cout} = \) 
- \( f(A,B,C) = \Sigma m(0,4,6,7) \)

Among the subcubes, the complement of the function can be obtained by covering the 0s with subcubes.

\[ AB' + AC' + B'C' \]
More Karnaugh map examples

\[
G(A,B,C) = A
\]

\[
F(A,B,C) = \Sigma m(0,4,5,7) = AC + B'C'
\]

\[
F' \text{ simply replace 1's with 0's and vice versa}
F'(A,B,C) = \Sigma m(1,2,3,6) = BC' + A'C
\]

Karnaugh map: 4-variable example

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

\[
F = C + A'BD + B'D'
\]

find the smallest number of the largest possible subcubes to cover the ON-set (fewer terms with fewer inputs per term)
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
  \[ f = A'D + B'C'D \]

Karnaugh maps: don’t cares (cont’d)

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

- without don’t cares
  \[ f = A'D + B'C'D \]
- with don’t cares
  \[ f = A'D + C'D \]

by using don’t care as a “1”
a 2-cube can be formed
rather than a 1-cube to cover
this node

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

Minimize the function \( F = \Sigma m(0, 2, 7, 14, 15) + d(3, 6, 9, 12, 13) \)

Design example: two-bit comparator

block diagram and truth table

we'll need a 4-variable Karnaugh map for each of the 3 output functions
Design example: two-bit comparator (cont'd)

Two alternative implementations of eq with and without XOR

XNOR is implemented with at least 3 simple gates
Design example: 2x2-bit multiplier

4-variable K-map for each of the 4 output functions

Design example: 2x2-bit multiplier (cont’d)
Design example: BCD increment by 1

<table>
<thead>
<tr>
<th>I8</th>
<th>I4</th>
<th>I2</th>
<th>I1</th>
<th>O8</th>
<th>O4</th>
<th>O2</th>
<th>O1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

4-variable K-map for each of the 4 output functions

Design example: BCD increment by 1 (cont’d)

O8 = I4 I2 I1 + I8 I1'
O4 = I4 I2' + I4 I1' + I4' I2 I1
O2 = I8' I2' I1 + I2 I1'
O1 = I1'

O8

I8

I4

I2

I1

O2

I8

I4

I2

I1

O4

I8

I4

I2

I1
**Definition of terms for two-level simplification**

- **Implicant**
  - single element of ON-set or DC-set or any group of these elements that can be combined to form a subcube

- **Prime implicant**
  - implicant that can't be combined with another to form a larger subcube

- **Essential prime implicant**
  - prime implicant is essential if it alone covers an element of ON-set
  - will participate in ALL possible covers of the ON-set
  - DC-set used to form prime implicants but not to make implicant essential

- **Objective:**
  - grow implicant into prime implicants
    - (minimize literals per term)
  - cover the ON-set with as few prime implicants as possible
    - (minimize number of product terms)

**Examples to illustrate terms**

- **6 prime implicants:**
  - \( A'B'D, BC', AC, A'C'D, AB, B'CD \)
  - essential
  - minimum cover: \( AC + BC' + A'B'D \)

- **5 prime implicants:**
  - \( BD, ABC', ACD, A'BC, A'C'D \)
  - essential
  - minimum cover: 4 essential implicants
Algorithm for two-level simplification

- **Algorithm:** minimum sum-of-products expression from a Karnaugh map

  - **Step 1:** choose an element of the ON-set
  - **Step 2:** find “maximal” groupings of 1s and Xs adjacent to that element
    - consider top/bottom row, left/right column, and corner adjacencies
    - this forms prime implicants (number of elements always a power of 2)

  - Repeat Steps 1 and 2 to find all prime implicants

  - **Step 3:** revisit the 1s in the K-map
    - if covered by single prime implicant, it is essential, and participates in final cover
    - 1s covered by essential prime implicant do not need to be revisited

  - **Step 4:** if there remain 1s not covered by essential prime implicants
    - select the smallest number of prime implicants that cover the remaining 1s

Algorithm for two-level simplification (example)
Activity

Combinational logic optimization summary

- Alternate representations of Boolean functions
  - cubes
  - karnaugh maps
- Simplification
  - two-level simplification
- Later (in CSE 467)
  - automation of simplification
  - optimization of multi-level logic
  - verification/equivalence