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

Winter 2001  CSE370 - N - Combinational Logic Optimization

---

Boolean cubes

**Visual technique for identifying when the uniting theorem can be applied**

**n input variables = n-dimensional "cube"

1-cube

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

2-cube

<table>
<thead>
<tr>
<th>01</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

3-cube

<table>
<thead>
<tr>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>X</td>
</tr>
</tbody>
</table>

4-cube

<table>
<thead>
<tr>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111</td>
</tr>
<tr>
<td>1011</td>
</tr>
<tr>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>X</td>
</tr>
</tbody>
</table>

Winter 2001  CSE370 - N - Combinational Logic Optimization
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>

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'

ON-set = solid nodes
OFF-set = empty nodes
DC-set = ×'d nodes

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>0</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

\[ \text{Cout} = B\text{Cin} + AB + AC\text{in} \]
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

```
A B F
0 0 1
0 1 0
1 0 1
1 1 0
```

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

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

```
A B
0 4
1 5
3 7
4 12
5 13
7 15
12 13
13 11
```

13 = 1101 = ABC'D
**Adjacencies in Karnaugh maps**

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

**Karnaugh map examples**

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

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

- \( AB + AC\bar{C} + \bar{B}\bar{C} \)

obtain the complement of the function by covering 0s with subcubes
More Karnaugh map examples

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

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

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

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

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

F: 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
  \[ f = A'D + B'CD \]

**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
  \[ f = A'D + B'CD \]

- 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

\[ \text{don't cares can be treated as} \]
\[ \text{1s or 0s} \]
\[ \text{depending on which is more} \]
\[ \text{advantageous} \]
Activity

** Minimize the function \( F = \Sigma m(0, 2, 7, 8, 14, 15) + d(3, 6, 9, 12, 13) \)

Design example: two-bit comparator

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>LT</th>
<th>EQ</th>
<th>GT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\( A \text{ B } C \text{ D} \) LT EQ GT

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

LT = \( A' B' D + A' C + B' C D \)
EQ = \( A' B' C D' + A' B C D + A B C D + A B' C D' = (A \text{xor} C) \cdot (B \text{xor} D) \)
GT = \( B' C D' + A C' + A B D' \)

LT and GT are similar (flip A/C and B/D)

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

<table>
<thead>
<tr>
<th>A2</th>
<th>A1</th>
<th>B2</th>
<th>B1</th>
<th>P8</th>
<th>P4</th>
<th>P2</th>
<th>P1</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>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Design example: 2x2-bit multiplier (cont'd)

K-map for P8

P8 = A2A1B2B1

K-map for P4


K-map for P2


K-map for P1

P1 = A1B1

Winter 2001 CSE370 - N - Combinational Logic Optimization
Design example: BCD increment by 1

<table>
<thead>
<tr>
<th>( I_8 )</th>
<th>( I_4 )</th>
<th>( I_2 )</th>
<th>( I_1 )</th>
<th>( O_8 )</th>
<th>( O_4 )</th>
<th>( O_2 )</th>
<th>( O_1 )</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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>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>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)

\[ O_8 = I_4 I_2 I_1 + I_8 I_1' \]

\[ O_4 = I_4 I_2' + I_4 I_1' + I_4' I_2 I_1 \]

\[ O_2 = I_8' I_2 I_1 + I_2 I_1' \]

\[ O_1 = I_1' \]
Definition of terms for two-level simplification

- **Implicant**: a single element of ON-set or DC-set or any group of these elements that can be combined to form a subcube.
- **Prime implicant**: an implicant that can't be combined with another to form a larger subcube.
- **Essential prime implicant**: a prime implicant is essential if it alone covers an element of ON-set, will participate in all possible covers of the ON-set, and DC-set used to form prime implicants but not to make implicant essential.
- **Objective**: grow implicant into prime implicants (minimize literals per term) and 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

** List all prime implicants for the following K-map:

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

** Which are essential prime implicants?

** What is the minimum cover?

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