Optimization and Tradeoffs

Chapter 6: Optimization and Tradeoffs


http://www.ddvahid.com

Copyright © 2007 Frank Vahid

Instructors of courses requiring Vahid’s *Digital Design* textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice intact and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites. PowerPoint source (or pdf with animations) may not be posted to publicly-accessible websites, but may be posted for student use only, provided that no attribution information is omitted or altered, either in slides or animations. For other uses requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see http://www.ddvahid.com for information.

Introduction

- We now know how to build digital circuits
  - How can we build **better** circuits?
- Let’s consider two important design criteria
  - Delay – the time from inputs changing to new correct stable output
  - Size – the number of transistors
  - For quick estimation, assume
    - Every gate has delay of “1 gate-delay”
    - Every gate input requires 2 transistors
    - Ignore inverters

Transforming $F_1$ to $F_2$ represents an **optimization**: Better in all criteria of interest

<table>
<thead>
<tr>
<th>$F_1$</th>
<th>$F_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$wxy + wxy'$</td>
<td>$wx$</td>
</tr>
<tr>
<td>16 transistors, 2 gate-delays</td>
<td>4 transistors, 1 gate-delay</td>
</tr>
<tr>
<td>$wxy(y+y') = wx$</td>
<td>(b)</td>
</tr>
</tbody>
</table>

Transforming $G_1$ to $G_2$ represents a **tradeoff**: Some criteria better, others worse.

<table>
<thead>
<tr>
<th>$G_1$</th>
<th>$G_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$wx + wy + z$</td>
<td>$w(x+y) + z$</td>
</tr>
<tr>
<td>14 transistors, 2 gate-delays</td>
<td>12 transistors, 3 gate-delays</td>
</tr>
<tr>
<td>$w$</td>
<td>12</td>
</tr>
<tr>
<td>$x$</td>
<td>12</td>
</tr>
<tr>
<td>$y$</td>
<td>12</td>
</tr>
<tr>
<td>$z$</td>
<td>12</td>
</tr>
</tbody>
</table>

Delay (gate-delays)

Size (transistors)
### Optimization and Tradeoffs

#### Algebraic Two-Level Size Minimization

**Optimizations**
- All criteria of interest are improved, while others are worsened

**Tradeoffs**
- Some criteria of interest are improved, while others are worsened

We obviously prefer optimizations, but often must accept tradeoffs.
- You can't build a car that is the most comfortable, and has the best fuel efficiency, and is the fastest – you have to give up something to gain other things.

- Previous example showed common algebraic minimization method
- Previous example showed common algebraic minimization method
- Previous example showed common algebraic minimization method

#### Karnaugh Maps for Two-Level Size Minimization

**Example**
- \( F = xy'z + xyz + xyz' + x'y'z' \)
- \( F = x'y' + x'y \)
- 4 literals + 2 terms = 6
- 5 gate inputs = 12 transistors

**Note:** Assuming 4-transistor 2-input AND/OR circuits; in reality, only NAND/NOR are so efficient.

### Optimization and Tradeoffs

#### Combinational Logic Optimization and Tradeoffs

- Easy to miss "seeing" possible opportunities to combine terms

#### Karnaugh Maps (K-maps)

**Graphical** method to help us find opportunities to combine terms
- Minterms differing in one variable are adjacent in the map
- Can clearly see opportunities to combine terms – look for adjacent 1s
- For \( F \), clearly two opportunities
- Top left circle is shorthand for \( x'y'z' + x'y'z = x'y'(z + z') = x'y' \)
- Draw circle, write term that has all the literals except the one that changes in the circle
- Circle \( xy, x=1 \ & y=1 \) in both calls of the circle, but \( z \) changes \( z=1 \) in one cell, \( 0 \) in the other
- Minimized function: OR the final terms

Easier than all that algebra...
**Optimization and Tradeoffs**

**Karnaugh Maps for Two-Level Size Minimization**

- Four adjacent 1s means two variables can be eliminated
  - Makes intuitive sense – those two variables appear in all combinations, so one must be true
  - Draw one big circle – shorthand for the algebraic transformations above

\[ G = x'y'z' + xy'z + xz + yz' \]

- No need to cover 1s more than once
  - Yields extra terms – not minimized

**Optimization and Tradeoffs**

**Karnaugh Maps for Two-Level Size Minimization**

- Cycles can cross left/right sides
  - Remember, edges are adjacent
  - Minterms differ in one variable only

- Cycles must have 1, 2, 4, or 8 cells – 3, 5, or 7 not allowed
  - 3/5/7 doesn’t correspond to algebraic transformations that combine terms to eliminate a variable

- Circling all the cells is OK
  - Function just equals 1

\[ G = x'y'z' + xy'z + xz + yz' \]
Optimization and Tradeoffs

Karnaugh Maps for Two-Level Size Minimization

General K-map method
1. Convert the function’s equation into sum-of-products form
2. Place 1s in the appropriate K-map cells for each term
3. Cover all 1s by drawing the fewest largest circles, with every 1 included at least once; write the corresponding term for each circle
4. OR all the resulting terms to create the minimized function.

Example: Minimize:
\[ G = a + a'b'c' + b*(c' + bc') \]

1. Convert to sum-of-products:
\[ G = a + a'b'c' + bc' + bc' \]
2. Place 1s in appropriate cells

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
3. Cover 1s

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4. OR terms: \[ G = a + c' \]

Optimization and Tradeoffs

Karnaugh Maps: Don’t Care Input Combinations

What if particular input combinations can never occur?
- e.g., Minimize \( F = xy'z' \), given that \( x'y'z' \) (xyz=000) can never be true, and that \( xy'z \) (xyz=101) can never be true
- So it doesn’t matter what \( F \) outputs when \( x'y'z' \) or \( xy'z \) is true, because those cases will never occur
- Thus, make \( F \) be 1 or 0 for those cases in a way that best minimizes the equation

On K-map
- Draw Xs for don’t care combinations
  - Include X in circle ONLY if minimizes equation
  - Don’t include other Xs

Minimize:
\[ F = a'bc' + abc + a'b'c \]
Given don’t cares: \( a'bc, abc \)

Note: Use don’t cares with caution
- Must be sure that we really don’t care what the function outputs for that input combination
- If we do care, even the slightest, then it’s probably safer to set the output to 0

\[ F = a'c + b \]
Optimization and Tradeoffs

Karnaugh Maps: Don’t Care Input Combinations

**Example:**
- Switch with 5 positions
- 3-bit value gives position in binary

**Want circuit that**
- Outputs 1 when switch is in position 2, 3, or 4
- Outputs 0 when switch is in position 1 or 5
- Note that the 3-bit input can never output binary 0, 6, or 7

- Treat as don’t care input combinations

\[ \begin{array}{c|ccc|c}
2 & 3 & 4 & 5 \\
\hline
0 & X & X & X \\
1 & 1 & 1 & 1 \\
\end{array} \]

\[ F = x'y + xy'z' \]

\[ F = y + z' \]

Optimization and Tradeoffs

Automating Two-Level Logic Size Minimization

**Minimizing by hand**
- Is hard for functions with 5 or more variables
- May not yield minimum cover depending on order we choose
- Is error prone

**Minimization thus typically done by automated tools**
- **Exact algorithm**: finds optimal solution
- **Heuristic**: finds good solution, but not necessarily optimal

Optimization and Tradeoffs

Basic Concepts Underlying Automated Two-Level Logic Minimization

**Definitions**
- **On-set**: All minterms that define when \( F=1 \)
- **Off-set**: All minterms that define when \( F=0 \)
- **Implicant**: Any product term (minterm or other) that when 1 causes \( F=1 \)
  - On K-map, any legal (but not necessarily largest) circle
  - Cover: Implicant \( xy \) covers minterms \( x'y'z \) and \( xy'z' \)
  - **Expanding** a term: removing a variable (like larger K-map circle)
  - \( xyz \rightarrow xy \) is an expansion of \( xyz \)

- **Prime implicant**: Maximally expanded implicant—any expansion would cover 1s not in on-set
  - \( x'y'z \), and \( xy \), above
  - But not \( xyz \) or \( xyz' \)—they can be expanded

**Essential prime implicant**: The only prime implicant that covers a particular minterm in a function’s on-set

- Importance: We **must** include all essential PIs in a function’s cover
- In contrast, some, but not all, non-essential PIs will be included
Optimization and Tradeoffs
Automated Two-Level Logic Minimization Method

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Determine prime implicants</td>
</tr>
<tr>
<td>2</td>
<td>Add essential prime implicants to the function’s cover</td>
</tr>
<tr>
<td>3</td>
<td>Cover remaining minterms with non-essential prime implicants</td>
</tr>
</tbody>
</table>

- Steps 1 and 2: Exact

Optimization and Tradeoffs
Example of Automated Two-Level Minimization

1. Determine all prime implicants
2. Add essential PIs to cover
   - Italicized 1s are thus already covered
   - Only one uncovered 1 remains
3. Cover remaining minterms with non-essential PIs
   - Pick among the two possible PIs

Solution to Computation Problem

- Don’t generate all minterms or prime implicants
- Instead, just take input equation, and try to “iteratively” improve it
- Ex: F = abcdefgh + abcdefgh’ + jklmnop
  - Note: 15 variables, may have thousands of minterms
  - But can minimize just by combining first two terms:
    - F = abcdefgh’ + jklmnop = abcd + jklmnop

Problem with Methods that Enumerate all Minterms or Compute all Prime Implicants

- Too many minterms for functions with many variables
  - Function with 32 variables: 2^{32} = 4 billion possible minterms.
  - Too much compute time/memory
- Too many computations to generate all prime implicants
  - Comparing every minterm with every other minterm, for 32 variables, is (4 billion)^2 = 1 quadrillion computations
  - Functions with many variables could requires days, months, years, or more of computation – unreasonable
Optimization and Tradeoffs

Two-Level Minimization using Iterative Method

- Method: Randomly apply “expand” operations, see if helps
  - Expand: remove a variable from a term
    - Like expanding circle size on K-map
    - e.g., Expanding x’z to z legal, but expanding x’z to z’ not legal, in shown function
    - After expand, remove other terms covered by newly expanded term
  - Keep trying (iterate) until doesn’t help

Ex:

F = abcdefgh + abcdefgh’ + jklmnop

F = abcdefg + abcdefgh’ + jklmnop

F = abcdefg + jklmnop

Optimization and Tradeoffs

Multi-Level Logic Optimization – Performance/Size Tradeoffs

- We don’t always need the speed of two level logic
  - Multiple levels may yield fewer gates
  - Example
    - F1 = ab + acd + ace
      → F2 = ab + ac(d + e) = a(b + c(d + e))
  - General technique: Factor out literals – xy + xz = x(y+z)

Optimization and Tradeoffs

Multi-Level Example: Non-Critical Path

- Critical path: longest delay path to output
  - Optimization: reduce size of logic on non-critical paths by using multiple levels

Optimization and Tradeoffs

Multi-Level Example: Non-Critical Path

- Critical path: longest delay path to output
  - Optimization: reduce size of logic on non-critical paths by using multiple levels