Lecture 19 – Optimization

Two-Level Minimization

Karnaugh Maps

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

Example:

- 16 transistors, 2 gate-delays: $F_1 = wxy + w'xy'$
- 4 transistors, 1 gate-delay: $F_2 = wx$

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

Tradeoffs

Improves some, but worsens other criteria of interest.

Example:

- $G_1 = wx + wy + z$, 12 transistors, 3 gate-delays
- $G_2 = w(x+y) + z$, 10 transistors, 2 gate-delays

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

Combination Logic Optimization and Tradeoffs

Two-level size optimization using algebraic methods:

- Goal: circuit with only two levels (ORed AND gates), with minimum transistors
- Though transistors getting cheaper (Moore’s Law), they still cost something

Define problem algebraically:

- Sum-of-products yields two levels
- $F = \text{abc + abc'}$ is sum-of-products; $G = w(xy + z)$ is not.

Transform sum-of-products equation to have fewest literals and terms:

- Each literal and term translates to a gate input, each of which translates to about 2 transistors (see Ch. 2)
- Ignore inverters for simplicity

Example:

- $F = xyz + xyz' + x'y'z' + x'y'z$
- $F = xy(z + z') + x'y'(z + z')$
- $F = xy*1 + x'y'*1$
- $F = xy + x'y'$

Algebraic Two-Level Size Minimization

Previous example showed common algebraic minimization method:

- Multiply out to sum-of-products, then
- Apply following as much possible

- $ab + ab' = a(b + b') = a1 = a$

- “Combining terms to eliminate a variable”

- Duplicating a term sometimes helps

- Sometimes after combining terms, can combine resulting terms

- Define problem algebraically:

- Sum-of-products yields two levels
- $F = \text{abcd + abcd'}$ is sum-of-products; $G = w(xy + z)$ is not.

Transform sum-of-products equation to have fewest literals and terms:

- Each literal and term translates to a gate input, each of which translates to about 2 transistors (see Ch. 2)
- Ignore inverters for simplicity

Example:

- $F = \text{xy} + \text{xy'} + \text{xy''}$
- $F = \text{xy} + \text{xy'} + \text{xy''}$
- $F = x + xy'$
- $G = w(y' + y)$
- $G = \text{wxy' + wxy'' + wxy'''}$

Transforming $F_1$ to $F_2$ represents an optimization: Better in all criteria of interest.
Karnaugh Maps (K-maps)

- Two-variable maps exist
- Four-variable K-map follows

No need to cover 1s more than once
OK to cover a 1 twice

Four adjacent cells can be in shape of a square
Can clearly see opportunities to combine terms - look for adjacent 1s
- For f, clearly two opportunities
- Top left circle is shorthand for \( xy'z' + xy'z = x(y'z' + y'z) \)
- Draw circle, write term that has all the literals except the one that changes in the circle
- Circle \( x, y, z \) in both cells of the circle, but a changes \( x \) in one cell, \( z \) in the other
- Minimized function. OR the final terms

K-maps

- 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

K-maps for Four Variables

- Four-variable K-map follows same principle
- Adjacent cells differ in one variable
- Left/right adjacent
- Top/bottom also adjacent
- 5 and 6 variable maps exist
- But hard to use
- Two-variable maps exist
- But not very useful - easy to do algebraically by hand

Two-Level Size Minimization Using K-maps

General K-map method
- Convert the function's equation into sum-of-products form
- Place 1s in the appropriate K-map cells for each term
- Cover all 1s by drawing the fewest largest circles, with every 1 included at least once; write the corresponding term for each circle
- 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
\[ G = a + a'b'c' + b'c + bc' \]
2. Place 1s in appropriate cells
\[ a'b'c' \]
3. Cover 1s
\[ a'b'c' \]
4. OR terms:
\[ G = a + a'b'c' + b'c + bc' \]
Minimize:

\[ H = a'b'(cd' + c'd') + ab'c'd' + ab'cd' + a'bd + a'bcd' \]

1. Convert to sum-of-products:

\[ H = a'b'cd' + a'b'c'd' + ab'c'd' + ab'cd' + a'bd + a'bcd' \]

2. Place 1s in K-map cells

3. Cover 1s

4. OR resulting terms

Minimization using \( K \)-maps

- Four Variable Example

Minimization Example using Don’t Cares

- 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

Minimization with Don’t Cares Example: Sliding Switch

- 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

- Design Challenge

- Perform two-level logic size optimization for the function \( F(a,b,c) = ab'c' + abc + a'bc + abc' \) using (a) algebraic methods, (b) a \( K \)-map. Express the answers as sum-of-products.

- Due:

  - Next Lecture (Wednesday, November 16)
  - Extra Credit (Homework)

- 2 points

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

Design Challenge

- Perform two-level logic size optimization for the function \( F(a,b,c) = ab'c' + abc + a'bc + abc' \) using (a) algebraic methods, (b) a \( K \)-map. Express the answers as sum-of-products.

- Due:

  - Next Lecture (Wednesday, November 16)
  - Extra Credit (Homework)

- 2 points