ECE 274 Digital Logic – Fall 2008

Optimization and Tradeoffs

Two-Level Minimization, Karnaugh Maps, Exact and Heuristic Minimization, Multi-level Minimization

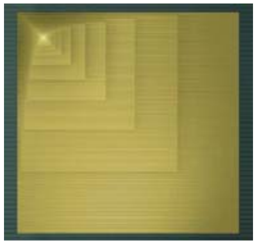
Digital Design 6.1 – 6.2

THE UNIVERSITY OF ARIZONA. TUCSON ARIZONA

Digital Design

Chapter 6: Optimization and Tradeoffs

Slides to accompany the textbook *Digital Design*, First Edition, by Frank Vahid, John Wiley and Sons Publishers, 2007. http://www.ddvahid.com

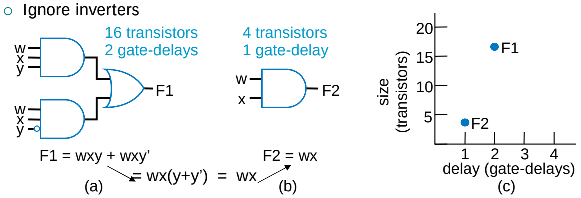


Optimization and Tradeoffs

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 F1 to F2 represents an ***optimization***: Better in all criteria of interest

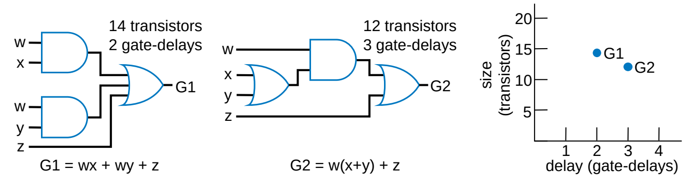


Optimization and Tradeoffs

Introduction

- Tradeoff
  - Improves some, but worsens other, criteria of interest

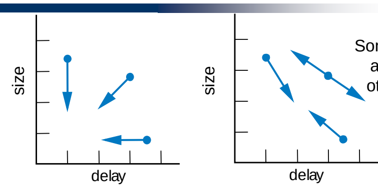
Transforming G1 to G2 represents a ***tradeoff***: Some criteria better, others worse.



Optimization and Tradeoffs
Introduction

Optimizations

All criteria of interest are improved (or at least kept the same)



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.

Optimization and Tradeoffs
Combinational 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 = 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

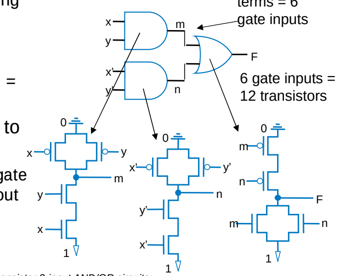
F = xyz + xyz' + x'y'z' + x'y'z

F = xy(z + z') + x'y'(z + z')

F = xy*1 + x'y'*1

F = xy + x'y'

4 literals + 2 terms = 6 gate inputs



6 gate inputs = 12 transistors

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

Optimization and Tradeoffs
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') = a*1 = a
    - "Combining terms to eliminate a variable"
      - (Formally called the "Uniting theorem")
  - Duplicating a term sometimes helps
    - Note that doesn't change function
      - c + d = c + d + d = c + d + d + d + d ...
  - Sometimes after combining terms, can combine resulting terms

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'

F = x'y'z' + **x'y'z** + x'yz
F = x'y'z' + **x'y'z** + **x'y'z** + x'yz
F = x'y'(z+z') + x'z(y'+y)
F = x'y' + x'z

G = xy'z' + xy'z + xyz + xyz'
G = xy'(z'+z) + xy(z+z')
G = xy' + xy   *(now do again)*
G = x(y'+y)
G = x

Optimization and Tradeoffs
Karnaugh Maps for Two-Level Size Minimization

- Easy to miss "seeing" possible opportunities to combine terms
- ***Karnaugh Maps (K-maps)***
  - **Graphical** method to help us find opportunities to combine terms
  - Minterms <u>differing in one variable</u> 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'(1) = 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 cells of the circle, but z changes (z=1 in one cell, 0 in the other)
    - Minimized function: OR the final terms

Notice not in binary order

K-map

F (x \ yz)	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

Treat left & right as adjacent too

F = x'y'z + xyz + xyz' + x'y'z'

F (x \ yz)	00	01	11	10
0	1	1	0	0
1	0	0	1	1

F (x \ yz)	00	01	11	10
0	1	1	0	0
1	0	0	1	1

F = x'y' + xy

x'y'   xy

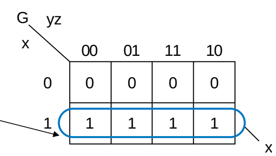
Easier than all that algebra:

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'

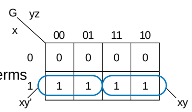
Optimization and Tradeoffs
Karnaugh Maps for Two-Level Size Minimization

- 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 = xy'z' + xy'z + xyz + xyz'
G = x(y'z'+ y'z + yz + yz') (must be true)
G = x(y'(z'+z) + y(z+z'))
G = x(y'+y)
G = x



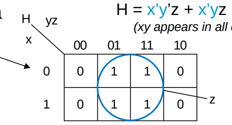
Draw the biggest circle possible, or you'll have more terms than really needed

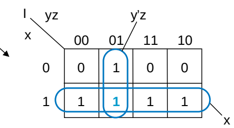


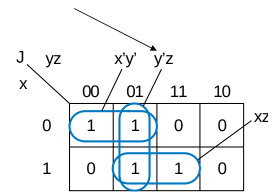
Optimization and Tradeoffs
Karnaugh Maps for Two-Level Size Minimization

- Four adjacent cells can be in shape of a square
- *OK* to cover a 1 twice
  - Just like duplicating a term
    - Remember, c + d = c + d + d
- No *need* to cover 1s more than once
  - Yields extra terms – not minimized

H = x'y'z + x'yz + xy'z + xyz
(xy appears in all combinations)



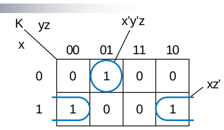


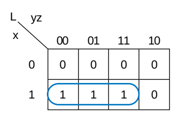


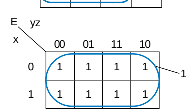
The two circles are shorthand for:
I = x'y'z + xy'z' + xy'z + xyz + xyz'
I = x'y'z + xy'z + xy'z' + xy'z + xyz + xyz'
I = (x'y'z + xy'z) + (xy'z' + xy'z + xyz + xyz')
I = (y'z) + (x)

Optimization and Tradeoffs
Karnaugh Maps for Two-Level Size Minimization

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

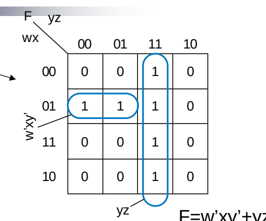


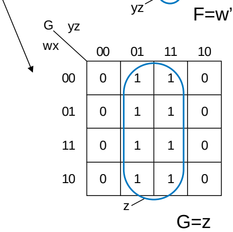


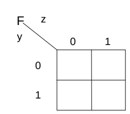


Optimization and Tradeoffs
Karnaugh Maps for Two-Level Size Minimization

- 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







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

a'b'c'

bc'

G (a \ bc)	00	01	11	10
0	1	0	0	1
1	1	1	1	1

a

3. Cover 1s

c'

G (a \ bc)	00	01	11	10
0	1	0	0	1
1	1	1	1	1

a

4. OR terms: **G = a + c'**

Optimization and Tradeoffs
Karnaugh Maps for Two-Level Size Minimization

- 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

a'b'c'd' a'b'cd' ab'c'd' a'bd a'bcd' ab'cd'

H (ab \ cd)	00	01	11	10
00	1	0	0	1
01	0	1	1	1
11	0	0	0	0
10	1	0	0	1

b'd'

a'bc

a'bd

Funny-looking circle, but remember that left/right adjacent, and top/bottom adjacent

H = b'd' + a'bc + a'bd

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 **X**s for don't care combinations
    - Include X in circle ONLY if minimizes equation
    - Don't include other Xs

y'z'

F (x \ yz)	00	01	11	10
0	X	0	0	0
1	1	X	0	0

Good use of don't cares

y'z' unneeded

F (x \ yz)	00	01	11	10
0	X	0	0	0
1	1	X	0	0

xy'

Unnecessary use of don't cares; results in extra term

Optimization and Tradeoffs
Karnaugh Maps: Don't Care Input Combinations

- 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

a'c b

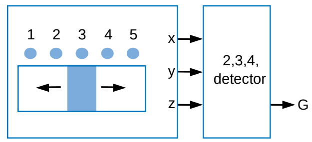
F (a \ bc)	00	01	11	10
0	0	1	X	1
1	0	0	X	1

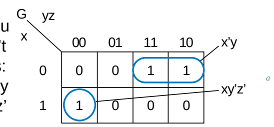
F = a'c + b

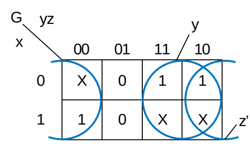
Optimization and Tradeoffs

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



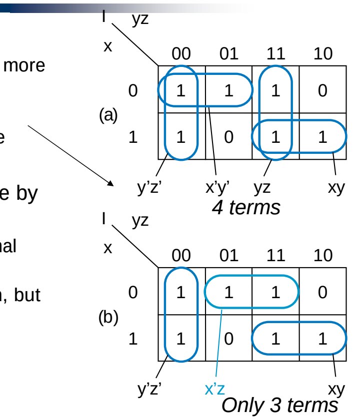




Optimization and Tradeoffs

Automating Two-Level Logic Size Minimization

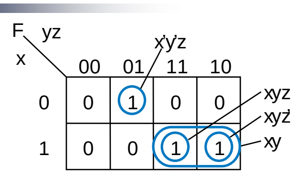
- Minimizing by hand
  - Is hard for functions with 5 or more variables
  - May not yield minimum cover depending on order we choose
  - Is error prone
- Minimization thus typically done by automated tools
  - ***Exact algorithm***: finds optimal solution
  - ***Heuristic***: finds good solution, but 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 xyz and xyz'
  - ***Expanding*** a term: removing a variable (like larger K-map circle)
    - xyz → xy is an expansion of xyz



4 implicants of F

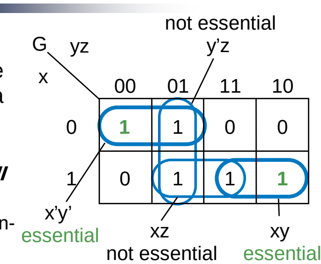
Note: We use K-maps here just for intuitive illustration of concepts; automated tools do __not__ use K-maps.

- ***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

Optimization and Tradeoffs

Basic Concepts Underlying Automated Two-Level Logic Minimization

- Definitions (cont)
  - ***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



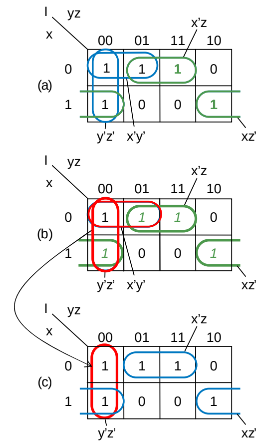
Optimization and Tradeoffs
Automated Two-Level Logic Minimization Method

Step	Description
1 *Determine prime implicants*	For every minterm in the function's on-set, maximally expand the term (meaning eliminate literals from the term) such that the term still only covers minterms in the function's on-set (like drawing the biggest circle possible around each 1 in a K-map). Repeat for each minterm. If don't cares exist, use them to maximally expand minterms into prime implicants (like using X's to create the biggest circles possible for a given 1 in a K-map).
2 *Add essential prime implicants to the function's cover*	Find any minterms covered by only one prime implicant (i.e., by an essential prime implicant). Add those prime implicants to the cover, and mark the minterms covered by those implicants as already covered.
3 *Cover remaining minterms with nonessential prime implicants*	Cover the remaining minterms using the minimal number of remaining prime implicants.

- Steps 1 and 2: Exact
- Step 3: Hard. Checking all possibilities: exact, but computationally expensive. Checking some but not all: heuristic.

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



Optimization and Tradeoffs
Problem with Methods that Enumerate all Minterms or Compute all Prime Implicants

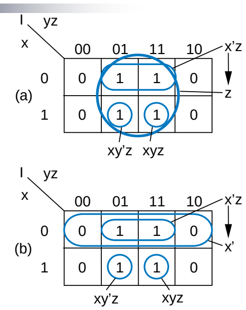
- Too many minterms for functions with many variables
  - Function with 32 variables:
    - 2^{32} = 4 billion possible minterms.
    - Too much compute time/memory
- Too many computations to generate all prime implicants
  - Comparing every minterm with every other minterm, for 32 variables, is (4 billion)^2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

Optimization and Tradeoffs
Solution to Computation Problem

- Solution
  - 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 = abcdefg(h+h') + jklmnop = abcdefg + jklmnop

Optimization and Tradeoffs
Two-Level Minimization using Iterative Method

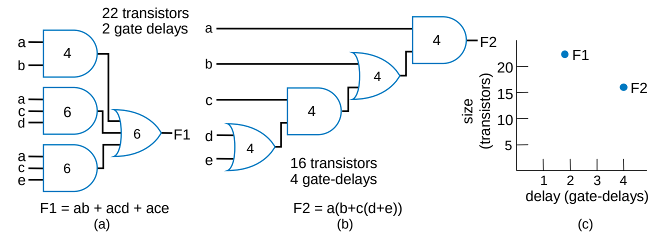
- Method: Randomly apply “expand” operations, see if helps
  - Expand: remove a variable from a term
    - Like expanding circle size on K-map
      - e.g., Expanding x'z to z legal, but expanding x'z to z' not legal, in shown function
      - After expand, remove other terms covered by newly expanded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



Ex:
F = abcdefgh + abcdefgh'+ jklmnop
F = abcdefg + abcdefgh' + jklmnop
F = abcdefg + jklmnop

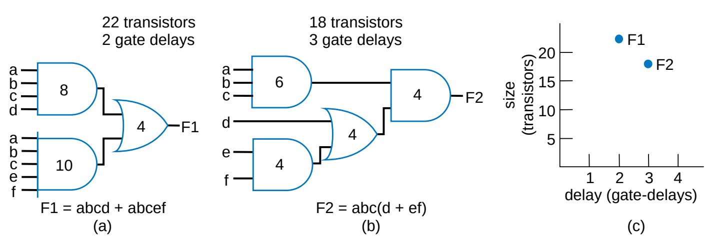
Optimization and Tradeoffs
Multi-Level Logic Optimization – Performance/Size Tradeoffs

- 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 Logic Optimization – Performance/Size Tradeoffs

- Use multiple levels to reduce number of transistors for
  - F1 = abcd + abcef
- Solution
  - F2 = abcd + abcef = abc(d + ef)
  - Has fewer gate inputs, thus fewer transistors



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