

COM 251 Logic Design and Circuits

Optimization and Tradeoffs: Combinational Logic

Prof. Dr. Halûk Gümüşkaya

haluk.gumuskaya@gediz.edu.tr

haluk@gumuskaya.com http://www.gumuskaya.com

Computer Engineering Department

GEDİZÜNİVERSİTESİ
izmir

Thursday, October 27, 2011

1

Acknowledgement

The slides have been based in-part upon original slides of a number of books including:

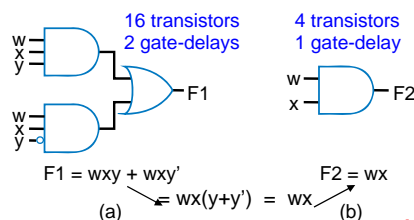
- *Digital Design with RTL Design, Verilog and VHDL*, 2nd ed., Frank Vahid, John Wiley, 2011.

2

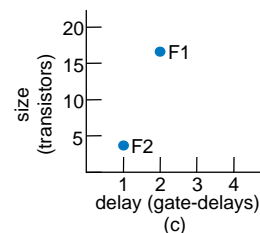
6.1

Introduction

- We now know how to build digital circuits
 - How can we build **better** circuits?
- Let's consider 2 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

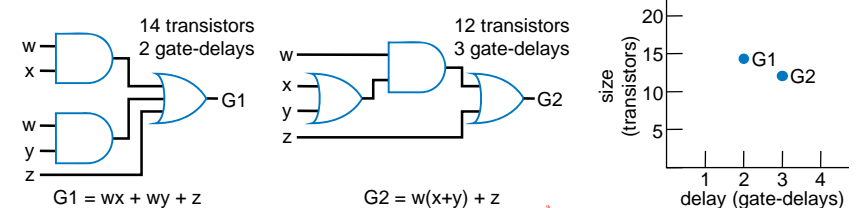


Note: Slides with animation are denoted with a small red "a" near the animated items

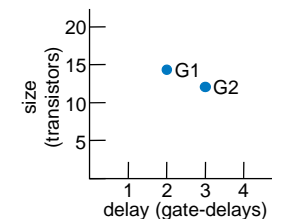
3

Introduction

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



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

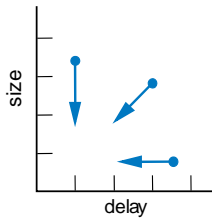


4

Introduction

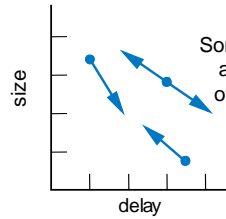
Optimizations

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.

Combinational Logic Optimization and Tradeoffs

- Two-level size optimization using algebraic methods
 - Goal: Two-level circuit (ORed AND gates) with fewest transistors
 - Though transistors getting cheaper (Moore's Law), 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)
 - For simplicity, ignore inverters

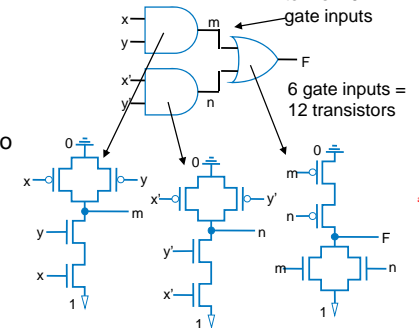
Example

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

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

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

$$F = xy + x'y'$$



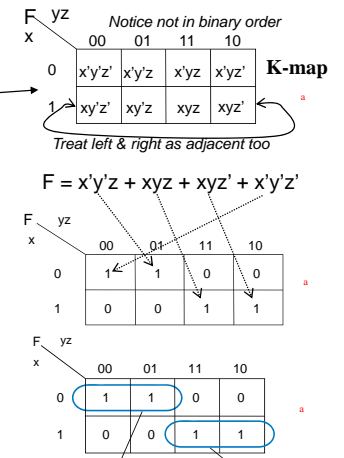
Note: Assuming 4-transistor 2-input AND/OR circuits; in reality, only NAND/NOR use only 4 transistors.

Algebraic Two-Level Size Optimization

- Previous example showed common algebraic minimization method
 - (Multiply out to sum-of-products, then...)
 - Apply following as much as 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
 - Doesn't change function
 - $c + d = c + d + d = c + d + d + d + d \dots$
 - Sometimes after combining terms, can combine resulting terms
 - $G = xy'z' + xy'z + xyz + xyz'$
 - $G = xy'(z' + z) + xy(z + z')$
 - $G = xy' + xy$ (now do again)
 - $G = x(y' + y)$
 - $G = x$

Karnaugh Maps for Two-Level Size Optimization

- Easy to miss possible opportunities to combine terms when doing algebraically
- 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: $x'y'z' + x'y'z = x'y'(z' + z) = x'y'(1) = x'y'$
 - 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 $F = x'y' + xy$



Easier than algebraically:

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

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

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

$$F = xy + x'y'$$

K-maps

- Four adjacent 1s means two variables can be eliminated

- Makes intuitive sense – those two variables appear in all combinations, so one term *must* be true
- Draw one big circle – shorthand for the algebraic transformations above

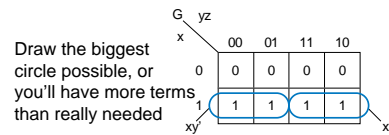
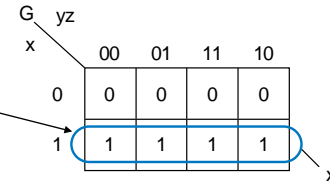
$$G = xy'z' + xy'z + xyz + xyz'$$

$$G = x(y'z' + y'z + yz + yz')$$

$$G = x(y'(z'+z) + y(z+z'))$$

$$G = x(y'+y)$$

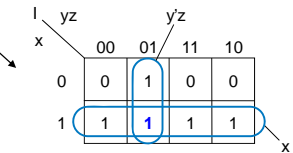
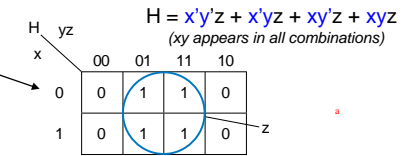
$$G = x$$



9

K-maps

- 4 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



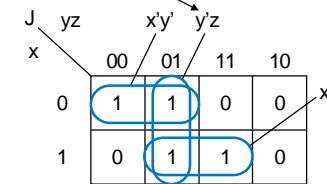
The two circles are shorthand for:

$$I = x'y'z + xy'z' + xy'z + xyz + xyz'$$

$$I = x'y'z + xy'z' + xy'z + xy'z + xyz + xyz'$$

$$I = (x'y'z + xy'z) + (xy'z' + xy'z + xyz + xyz')$$

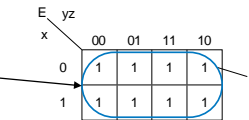
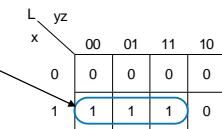
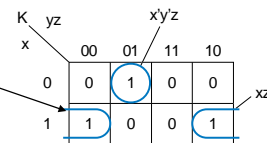
$$I = (y'z) + (x)$$



10

K-maps

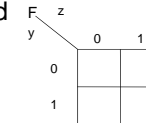
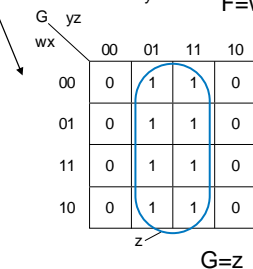
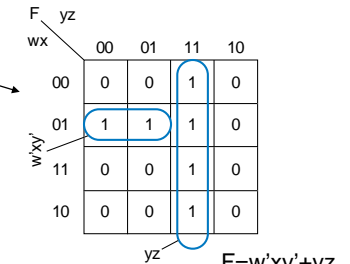
- Circles can cross left/right sides
 - Remember, edges are adjacent
 - Minterms differ in one variable only
- Circles must have 1, 2, 4, or 8 cells – 3, 5, or 7 not allowed
 - 3/5/7 doesn't correspond to algebraic transformations that combine terms to eliminate a variable
- Circling all the cells is OK
 - Function just equals 1



11

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



12

Two-Level Size Optimization Using K-maps

General K-map method

1. *Convert* the function's equation into sum-of-minterms form
2. *Place* 1s in the appropriate K-map cells for each minterm
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.

13

Two-Level Size Optimization Using K-maps

General K-map method

1. *Convert* the function's equation into sum-of-minterms form
2. *Place* 1s in the appropriate K-map cells for each minterm

Common to revise (1) and (2):

- Create *sum-of-products*
- Draw 1s for each product

Ex: $F = w'xz + yz + w'xy'z'$

F		yz		yz	
		w'xz	01	11	10
wx	00	0	0	1	0
	01	1	1	1	0
	11	0	0	1	0
	10	0	0	1	0

14

Two-Level Size Optimization Using K-maps

General K-map method

1. *Convert* the function's equation into sum-of-minterms form
2. *Place* 1s in the appropriate K-map cells for each minterm
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

G		bc			
		00	01	11	10
a	0	1	0	0	1
	1	1	1	1	1

3. Cover 1s

G		bc			
		00	01	11	10
a	0	1	0	0	1
	1	1	1	1	1

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

15

Two-Level Size Optimization Using K-maps

– Four Variable Example

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

H		cd			
		00	01	11	10
ab	00	0	0	1	0
	01	0	1	1	1
	11	0	0	0	0
	10	1	0	0	1

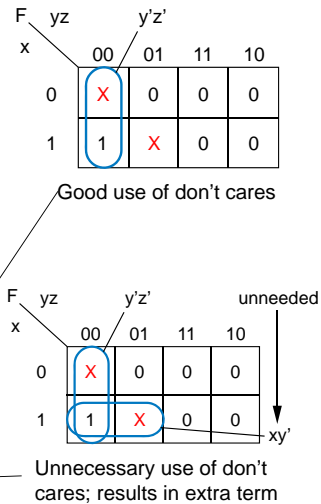
$$H = b'd' + a'bc + a'bd$$

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

16

Don't Care Input Combinations

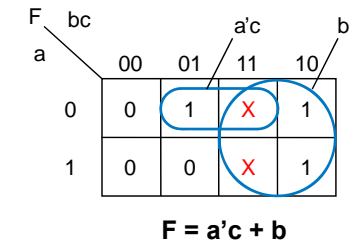
- What if we know that particular input combinations can never occur?
 - e.g., Minimize $F = xy'z'$, given that $x'y'z'$ ($xyz=000$) can *never* be true, and that $xy'z$ ($xyz=101$) can *never* be true
 - So it doesn't matter what F outputs when $x'y'z'$ or $xy'z$ is true, because those cases *will never 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



17

Optimization Example using Don't Cares

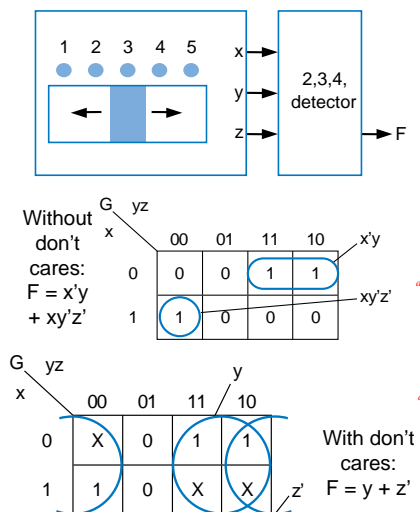
- Minimize:
 - $F = a'bc' + abc' + a'b'c$
 - Given don't cares: $a'bc, abc$
- Note: Introduce don't cares with caution
 - Must be *sure* that we really don't care what the function outputs for that input combination
 - If we do care, even the slightest, then it's probably safer to set the output to 0



18

Optimization with Don't Cares Example: Sliding Switch

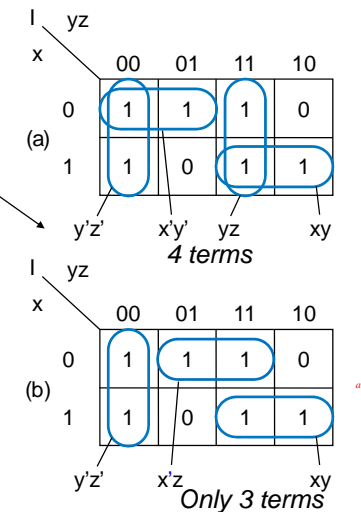
- Switch with 5 positions
 - 3-bit value gives position in binary
- Want circuit that
 - Outputs 1 when switch is in position 2, 3, or 4
 - Outputs 0 when switch is in position 1 or 5
 - Note that the 3-bit input can never output binary 0, 6, or 7
 - Treat as don't care input combinations



19

Automating Two-Level Logic Size Optimization

- 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

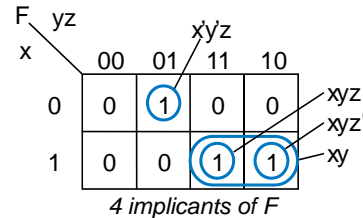


20

Basic Concepts Underlying Automated Two-Level Logic Size Optimization

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)
 - $xyz \rightarrow xy$ is an expansion of xyz



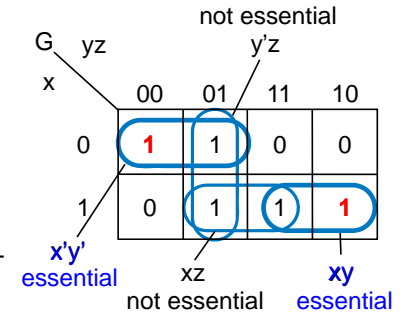
Note: We use K-maps here just for intuitive illustration of concepts; automated tools do **not** use K-maps.

- **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

Basic Concepts Underlying Automated Two-Level Logic Size Optimization

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



Automated Two-Level Logic Size Optimization Method

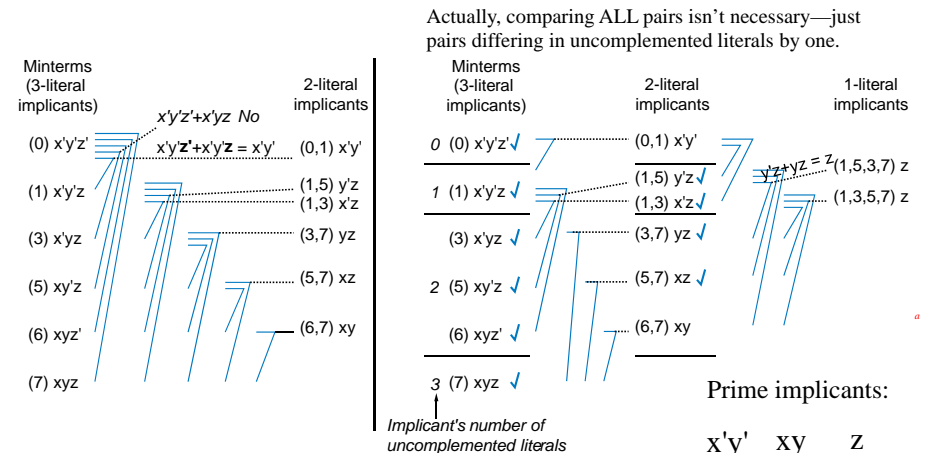
TABLE 6.1 Automatable tabular method for two-level logic size optimization.

Step	Description
1 Determine prime implicants	Starting with minterm implicants, methodically compare all pairs (actually, all pairs whose numbers of uncomplemented literals differ by one) to find opportunities to combine terms to eliminate a variable, yielding new implicants with one less literal. Repeat for new implicants. Stop when no implicants can be combined. All implicants not covered by a new implicant are prime implicants.
2 Add essential prime implicants to the function's cover	Find every minterm covered by only one prime implicant, and denote that prime implicant as essential. Add essential prime implicants to the cover, and mark all 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 are exact
- Step 3: Hard. Checking all possibilities: exact, but computationally expensive. Checking some but not all: heuristic.

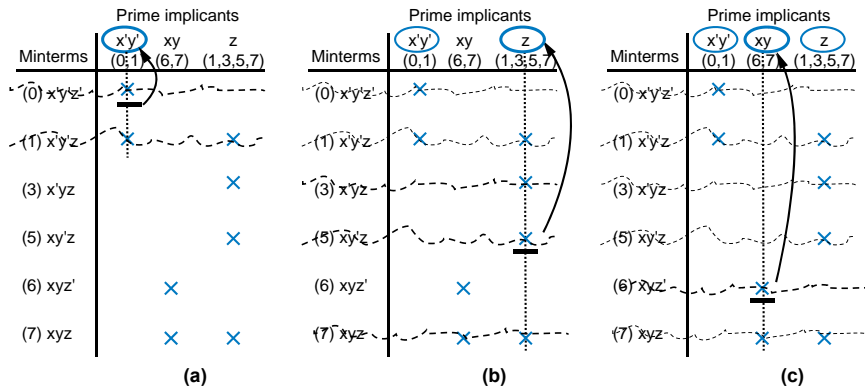
Tabular Method Step 1: Determine Prime Implicants Methodically Compare All Implicant Pairs, Try to Combine

- Example function: $F = x'y'z' + x'y'z + x'yz + xy'z + xyz' + xyz$



Tabular Method Step 2: Add Essential PIs to Cover

- Prime implicants (from Step 1): $x'y'$, xy , z

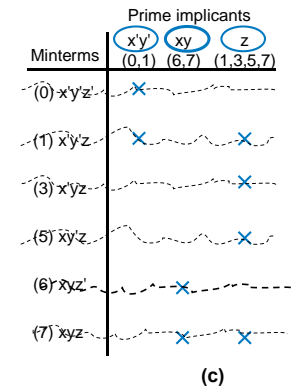


If only one **X** in row, then that PI is essential—it's the only PI that covers that row's minterm.

25

Tabular Method Step 3: Use Fewest Remaining PIs to Cover Remaining Minterms

- Essential PIs (from Step 2): $x'y'$, xy , z
 - Covered all minterms, thus nothing to do in step 3
- Final minimized equation:
 $F = x'y' + xy + z$



26

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 $(4 \text{ billion})^2 = 1$ quadrillion computations
 - Functions with many variables could require days, months, years, or more of computation – unreasonable

27

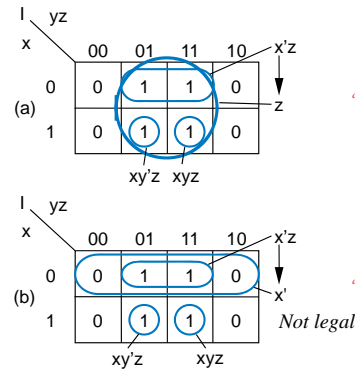
Solution to Computation Problem

- 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:
 - $F = abcdefg(h+h') + jklmnop = abcdefg + jklmnop$

28

Two-Level Optimization 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:
 $F = abcdefgh + abcdefgh' + jklmnop$
 $F = abcdefg + abcdefgh' + jklmnop$
 $F = abcdefg + jklmnop$

Covered by newly expanded term $abcdefg$

Illustrated above on K-map, but iterative method is intended for an automated solution (no K-map)

29

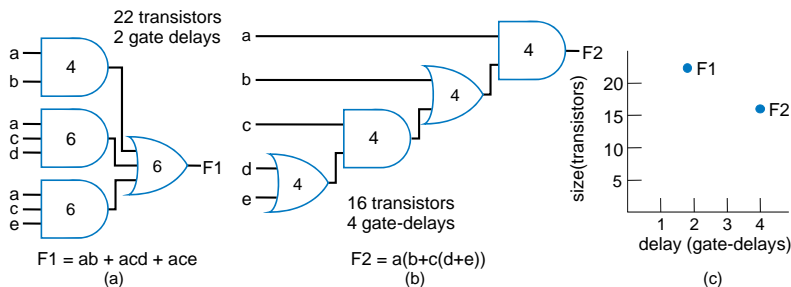
Ex: Iterative Hueristic for Two-Level Logic Size Optimization

- $F = xyz + xyz' + x'y'z' + x'y'z$ (minterms in on-set)
- Random expand: $F = xy\cancel{x} + xyz' + x'y'z' + x'y'z$
 - Legal: Covers xyz' and xyz , both in on-set
 - Any implicant covered by xy ? Yes, xyz' .
- $F = xy + \cancel{xy}z' + x'y'z' + x'y'z$
- Random expand: $F = \cancel{xy} + x'y'z' + x'y'z$
 - Not legal (x covers $xy'z'$, $xy'z$, xyz' , xyz : two not in on-set)
- Random expand: $F = xy + x'y'\cancel{x} + x'y'z$
 - Legal
 - Implicant covered by $x'y'$: $x'y'z$
- $F = xy + x'y'z' + \cancel{xy}z$

30

Multi-Level Logic Optimization – Performance/Size Tradeoffs

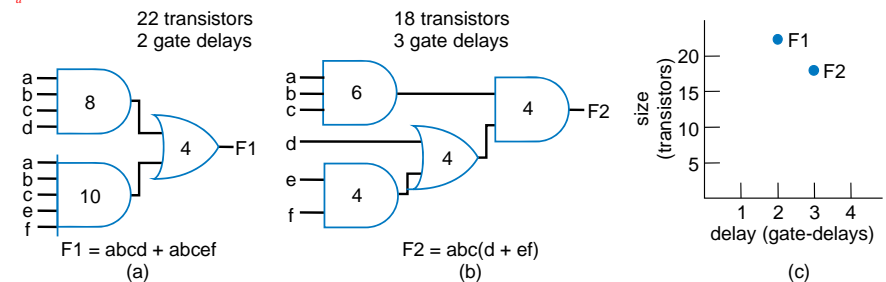
- We don't always need the speed of two-level logic
 - Multiple levels may yield fewer gates
 - Example
 - $F1 = ab + acd + ace \rightarrow F2 = ab + ac(d + e) = a(b + c(d + e))$
 - General technique: Factor out literals – $xy + xz = x(y+z)$



31

Multi-Level Example

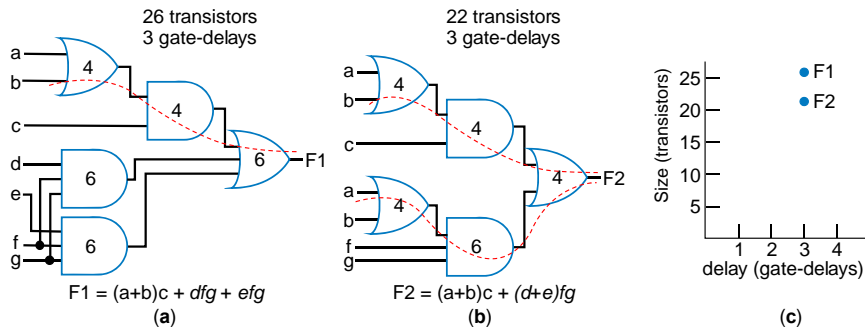
- Q: Use multiple levels to reduce number of transistors for
 - $F1 = abcd + abcef$
- A: $abcd + abcef = abc(d + ef)$
 - Has fewer gate inputs, thus fewer transistors



32

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



33

Automated Multi-Level Methods

- Main techniques use heuristic iterative methods
 - Define various operations
 - “Factoring”: $abc + abd = ab(c+d)$
 - Plus other transformations similar to two-level iterative improvement

34