Technology Independent Logic Optimization

Prof. Kurt Keutzer

EECS

University of California

Berkeley, CA

Thanks to R. Rudell, S. Malik

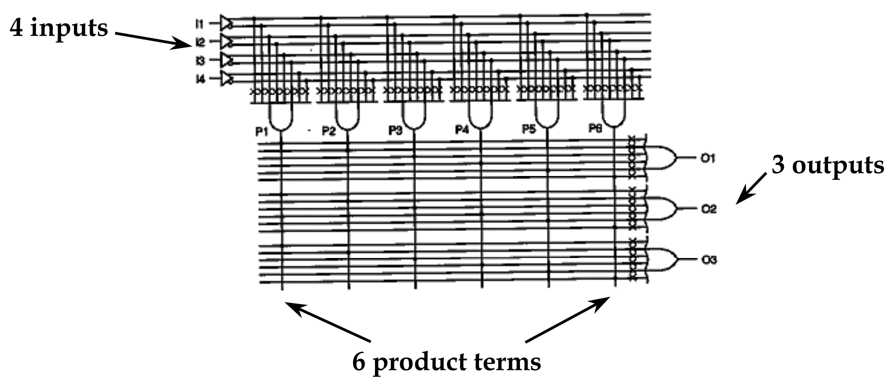
Outline

- Motivation for Multilevel
- Overview of Techniques
- Details on multilevel techniques

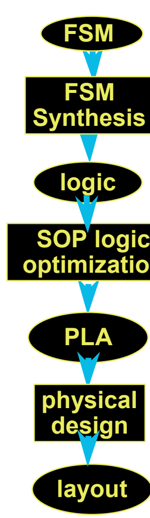
2-Level =>Programmable Logic Arrays (PLAs)

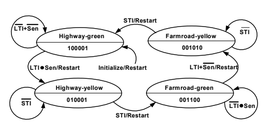
We can represent any Boolean function in a 2-level/SOP form

Any such representation can be implemented in a PLA

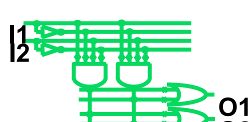


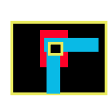
Early "Synthesis" Flow





F1 = B + D + A C + A C





Why Multilevel Combinational Circuits?

There are many functions that are too "expensive" to implement in two-level form

Try 16-bit adder ⇒ 32 input lines and 2^{16} product terms!

2-level: control logic design

multi-level: datapath logic or random logic

Two-Level versus Multilevel

Even simple functions expressed in 2-Level:

$$f_1 = AB + AC + AD$$
$$f_2 = \bar{A}B + \bar{A}C + \bar{A}E$$

6 product terms which cannot be shared.
24 transistors in static CMOS

May be more efficient in multi-level:

Note that $B + C$ is a common term in f_1 and f_2

$$K = B + C$$

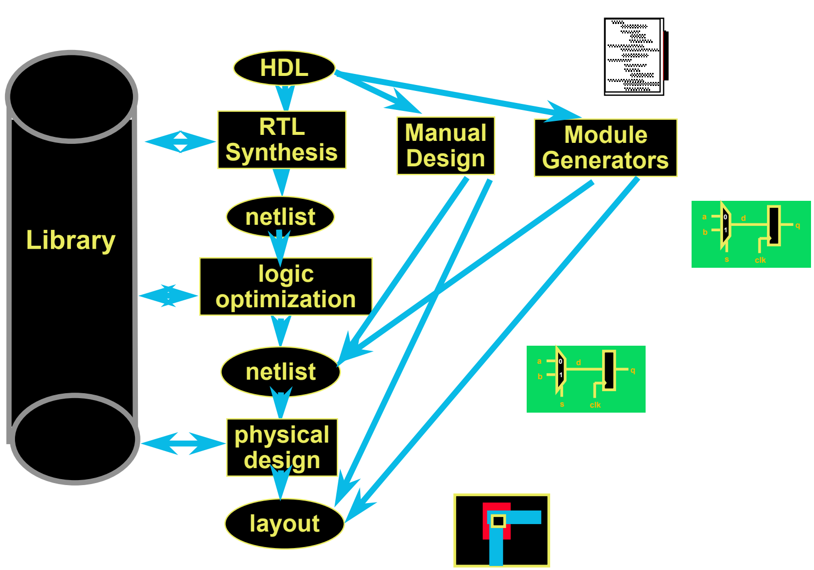
3 Levels
20 transistors in static CMOS
not counting inverters

$$f_1 = AK + AD$$
$$f_2 = \bar{A}K + \bar{A}E$$

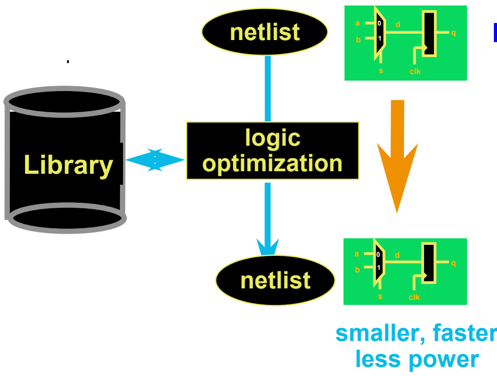
Outline

- Motivation for Multilevel
- Overview of Techniques
- Details on multilevel techniques

RTL Design Flow



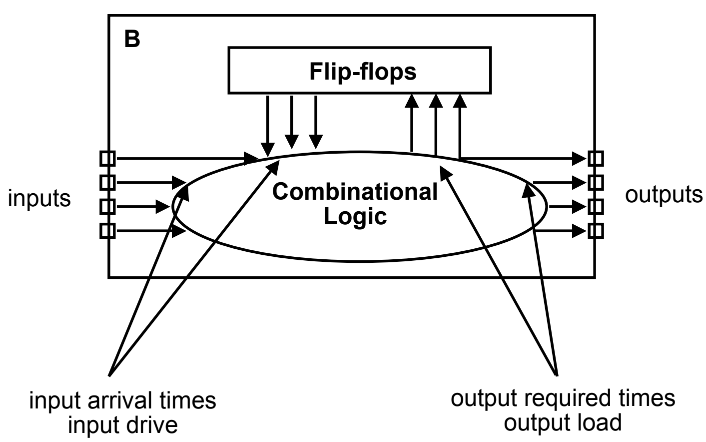
Logic Optimization



Perform a variety of transformations and optimizations

- Structural graph transformations
- Boolean transformations
- Mapping into a physical library

Reduce to Combinational Optimization



Representation: Boolean Network

$y_5 = H_5 = \bar{y}_1 y_2$
$y_6 = H_6 = y_3 y_4$
$y_7 = H_7 = y_1 \bar{y}_6 + y_2$
$y_8 = H_8 = y_5 + y_6$
$y_9 = H_9 = y_7$
$y_{10} = H_{10} = y_8$

A *Boolean network* is designated $\eta = (\vec{y}, \vec{H})$ where:

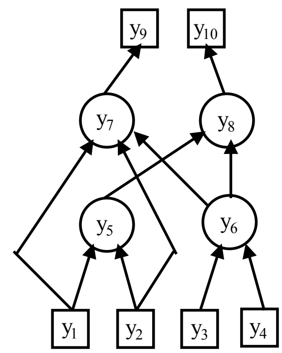
$\vec{y} = (y_1, \ldots, y_{n+m+r})$ is a vector of variables

$\vec{H} = (H_1, \ldots, H_{n+m+r})$ is a vector of functions

$y_1, \ldots, y_n$ are the primary input variables
$y_{n+1}, \ldots, y_{n+r}$ are the intermediate variables
$y_{n+r+1}, \ldots y_{n+m+r}$ are the primary output variables

$y_i = H_i(y_1, \ldots y_{n+m+r})$

A Boolean network has an associated graph which shows the function dependencies; i.e., the edge (i, j) is present if $y_i \in \sup(H_j)$.



Combinational Logic Optimization

Input:

Initial Boolean network

Timing characterization for the module

- input arrival times and drive factors
- output loading factors

Optimization goals

- output required times

Target library description

Output:

Minimum-area net-list of library gates which meets timing constraints

A very difficult optimization problem !

Modern Approach to Logic Optimization

Divide logic optimization into two subproblems:

- Technology-independent optimization
  - determine overall logic structure
  - estimate costs (mostly) independent of technology
  - simplified cost modeling
- Technology-dependent **optimization** (technology mapping)
  - binding onto the gates in the library
  - detailed technology-specific cost model

Orchestration of various optimization/transformation techniques for each subproblem

Formats

Boolean Network, Boolean equations

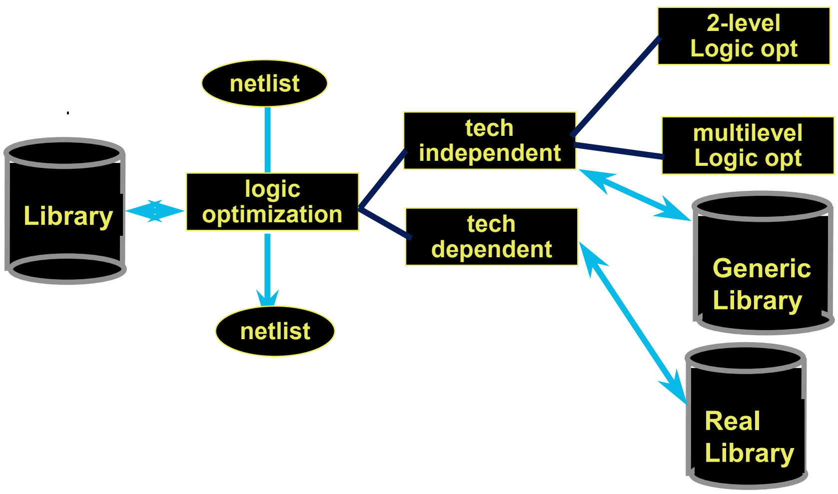
Generic library – technology independent

- **Has standard functions – ND2, ND4, AOI22, pos-edge-FF**
- **only an estimate of timing**

Actual technology library

- **Represents logic functions and their physical characteristics of a library cells offered by a particular silicon vendor – e.g. TSMC**
- **E.g. captured in .lib file**
- **Contains complete logical, timing information**

Logic Optimization



Tech.-Independent Optimization

Involves:
- Minimizing two-level logic functions.
- Finding common subexpressions.
- Substituting one expression into another.
- Factoring single functions.

Factored versus Disjunctive forms

$$f = ac + ad + bc + bd + a\bar{e}$$

sum-of-products or disjunctive form

$$f = (a+b)(c+d) + a\bar{e}$$

factored form

multi-level or complex gate

Decomposition

G is a *Boolean divisor* of F if $F = G \cdot H + R$ for functions $H \neq 0,\ R \neq 0$

Searching for divisors which are common to many functions in the network

Decomposition:

identify divisors which are common to several functions

introduce common divisor as a new node

re-express existing nodes using the new divisor

Technology-independent measure of cost to measure *goodness*

area cost: total number of literals

delay cost: levels of logic on the critical path

Division

- If $h \neq 0$, and h can be obtained using algebraic division, then g is an **algebraic divisor** of f. Otherwise, g is a **Boolean divisor** of f.
- Example:

  $f = ad + ae + bcd + j$
  $g_1 = a + bc$
  $g_2 = a + b$
- Algebraic division f//a = d + e, f//(bc) = d
- Also,
  f//a = d or f//a = e, i.e. algebraic division is not unique) $h_1 = f//g_1 = d$, $r_1 = ae + j$
- Boolean division: $h_2 = f \div g_2 = (a + c)d$, $r_2 = ae + j$.
  i.e. f = (a+b)(a+c)d + ae + j

Strong (or Boolean) Division

Given a function f to be strong divided by g

Add an extra input to f corresponding to g, namely G and obtain function h as follows

$$h_{DC} = G\bar{g} + \bar{G}g$$
$$h_{ON} = f_{ON} - h_{DC}$$
$$h_{OFF} = \overline{f_{ON} + h_{DC}}$$

Minimize h using two-level minimizer

Algebraic vs. Boolean Methods

Algebraic techniques view equations as polynomials and attempt to factor equations or "divide" them

Do not exploit Boolean identities e.g., $a\,\bar{a} = 0$

In algebraic substitution (or division) if a function $f = f(a, b, c)$ is divided by $g = g(a, b)$, a and b will <u>not</u> appear in f/g

Algebraic division: O(n log n) time

Boolean division: unmanageable number of divisors

Comparison of factorization

$$f = a\bar{b} + a\bar{c} + b\bar{a} + b\bar{c} + c\bar{a} + c\bar{b}$$

Algebraic factorization procedures

$$f = a(\bar{b} + \bar{c}) + \bar{a}(b + c) + b\bar{c} + c\bar{b}$$

Boolean factorization produces

$$f = (a + b + c)(\bar{a} + \bar{b} + \bar{c})$$

Comparison

Substitution is the factoring of one node in the Boolean Network (e.g. *l*) by another (e.g. *r*)

Algebraic substitution of l into r fails

Boolean substitution yields results

$$l = (b\bar{f} + \bar{b}f)(a + e) + \bar{a}\bar{e}(\bar{b}\bar{f} + bf)$$

$$r = (b\bar{f} + \bar{b}f)(\bar{a} + \bar{e}) + ae(\bar{b}\bar{f} + bf)$$

After resub:

$$r = a(\bar{e}\bar{l} + el) + \bar{a}(\bar{e}l + e\bar{l})$$

$$l = a(er + \bar{e}\bar{r}) + \bar{a}(\bar{e}r + e\bar{r})$$

Algebraic Decomposition

Algebraic approximation (informal definition)

- simplify Boolean function using two-level minimization
- treat result as a polynomial; i.e.,

  x_i and $\bar{x}_i$ are different variables
  i.e., $x_i \cdot \bar{x}_i \neq 0$ and $x_i \cdot x_i \neq x_i$

- identify common divisors as algebraic divisors of the polynomials

Algebraic Decomposition

Algebraic approximation (informal definition)

- simplify Boolean function using two-level minimization
- treat result as a polynomial; i.e.,

  x_i and $\bar{x}_i$ are different variables
  i.e., $x_i \cdot \bar{x}_i \neq 0$ and $x_i \cdot x_i \neq x_i$

- identify common divisors as algebraic divisors of the polynomials

Motivation

- manipulating polynomials is fast (linear time algorithms)
- # algebraic divisors still exponential, but usually manageable
- loss of optimality, but experimentally shows good results
- interleave Boolean simplification procedures to improve results

Algebraic Decomposition

Algebraic approximation (informal definition)

- **simplify Boolean function using two-level minimization**
- **treat result as a polynomial; i.e.,**

  x_i and $\bar{x}_i$ are different variables
  i.e., $x_i \cdot \bar{x}_i \neq 0$ and $x_i \cdot x_i \neq x_i$

- **identify common divisors as algebraic divisors of the polynomials**

Techniques

- **single-cube algebraic divisors (common-cube decomposition)**
- **multiple-cube algebraic divisors (kernel decomposition)**

Decomposition?

$$F = a\,d\,e + b\,d\,e + c\,d\,e + f$$

$$G = b\,g + c\,g + d\,g + a\,e\,f$$

$$H = a\,e\,g + b\,c$$

How can this logic be further simplified?

Common Cube Decomposition

$F = ade + bde + cde + f$ $G = bg + cg + dg + aef$ $H = aeg + bc$	$\rightarrow$	$F = Xd + bde + cde + f$ $G = bg + cg + dg + Xf$ $H = Xg + bc$ $X = ae$

Finds algebraic divisors which are single cubes

"Common cubes" are easy to detect

Greedy algorithm:

- **enumerate all maximal common cubes**
- **select cube which saves the most literals**
- **add node to the network and re-express affected nodes**
- **repeat until no common cubes remain**

References: [Dietmeyer-69], [Brayton-82], [Rudell-89]

Kernel Decomposition

$F = ade + bde + cde + f$ $G = bg + cg + dg + aef$ $H = aeg + bc$	$\rightarrow$	$F = (a + b + c)de + f$ $G = (b + c + d)g + aef$ $H = aeg + bc$	$\rightarrow$	$F = (a + X)de + f$ $G = (X + d)g + aef$ $H = aeg + bc$ $X = b + c$

cube-free: **an expression is cube-free if no literal appears in every cube**

kernel: **A kernel of an expression is a cube-free divisor which is not contained in any other cube-free divisor (e.g., $a + b + c$ is a kernel of *F*, while $b + c$ is not a kernel because it is contained in $a + b + c$)**

Kernels are useful because:

- **if *f* and *g* share a common multiple-cube divisor, then the intersection of some kernel from *f* and some kernel from *g* yields a common multiple- cube divisor [Brayton-82].**
- **practical algorithms exist to find intersections of kernels [Rudell-89].**
- **kernel decomposition finds solutions which are difficult to find using only common-cube decomposition**

Selective Collapsing

"Collapse" **nodes into their fanout to increase the size of each node**

$f = a\,h + b\,h' + c\,d$
$g = b\,h$
$h = a\,c + d'$

↓ collapse h

$f = a\,(a\,c + d') + b\,(a'\,d' + c'\,d') + c\,d$
$g = b\,(a\,c + d')$

↓ simplify

$f = a\,c + a\,d' + a'\,b\,d' + b\,c'\,d' + c\,d$
$g = a\,b\,c + b\,d'$

Goals:

- remove bad initial structure
- reduce logic level depth
- expose further optimization opportunities

Summary of Typical Recipe

Selective-collapse

Simplify: Two-level minimization at Boolean network node

Structuring/Algebraic decomposition

Local Boolean optimizations

↕ **Technology Independent Optimization**

Tree-covering for gate selection

Load-buffering for fanout-tree construction

Local transformation improvement of circuit structure

↕ **Technology Mapping**

Restructure and iterate if timing constraints not met

Outline

- **Motivation for Multilevel**
- **Overview of Techniques**
- **Details on multilevel techniques**

Decomposition - details

$$F = \begin{cases} f_1 = AB + AC + AD + AE + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E} \\ f_2 = \bar{A}B + \bar{A}C + \bar{A}D + \bar{A}F + \bar{A}\bar{B}\bar{C}\bar{D}\bar{F} \end{cases}$$

Factor F

$$F = \begin{cases} f_1 = A(B + C + D + E) + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E} \\ f_2 = \bar{A}(B + C + D + F) + \bar{A}\bar{B}\bar{C}\bar{D}\bar{F} \end{cases}$$

Extract common expression

$$G = \begin{cases} g_1 = B + C + D \\ f_1 = A(g_1 + E) + \bar{A}\bar{E}\bar{g}_1 \\ f_2 = \bar{A}(g_1 + F) + \bar{A}\bar{F}\bar{g}_1 \end{cases}$$

Representation: Boolean Network

A *Boolean network* is designated $\eta = (\vec{y}, \vec{H})$ where:

$\vec{y} = (y_1, \ldots, y_{n+m+r})$ is a vector of variables

$\vec{H} = (H_1, \ldots, H_{n+m+r})$ is a vector of functions

$y_1, \ldots, y_n$ are the primary input variables

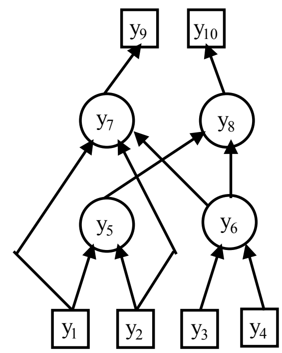
$y_{n+1}, \ldots, y_{n+r}$ are the intermediate variables

$y_{n+r+1}, \ldots y_{n+m+r}$ are the primary output variables

$y_i = H_i(y_1, \ldots y_{n+m+r})$

A Boolean network has an associated graph which shows the function dependencies; i.e., the edge (i, j) is present if $y_i \in \sup(H_j)$.

$y_5 = H_5 = \bar{y}_1 y_2$
$y_6 = H_6 = y_3 y_4$
$y_7 = H_7 = y_1 \bar{y}_6 + y_2$
$y_8 = H_8 = y_5 + y_6$
$y_9 = H_9 = y_7$
$y_{10} = H_{10} = y_8$



Weak (or Algebraic) Division

Definition: *support* of f, denoted $sup(f)$ = { set of all variables v that occur in f as $\bar{v}$ or v }

Example: $f = A\bar{B} + C$

$sup(f) = \{A, B, C\}$

Weak (or Algebraic) Division

Definition: *support* of f, denoted $sup(f)$ = { set of all variables v that occur in f as $\overline{v}$ or v }

Example: $f = A\overline{B} + C$

$sup(f) = \{ A, B, C \}$

Definition: we say that f is orthogonal to g, $f \perp g$, if $sup(f) \cap sup(g) = \phi$

Example: $f = A + B \qquad g = C + D$

$\therefore \; f \perp g$ since $\{ A, B \} \cap \{ C, D \} = \phi$

Weak Division - 2

We say that g *divides* f *weakly* if there exist h, r such that $f = gh + r$ where $h \neq \phi$ and $g \perp h$

Example:
$f = ab + ac + d$
$g = b + c$

$f = a(b + c) + d \qquad h = a \qquad r = d$

We say that g divides f evenly if $r = \phi$

The *quotient* f/g is the largest h such that $f = gh + r$ i.e., $f = (f/g)g + r$

Computing f/g

Given $f = \{ c_i \}$, $g = \{ a_i \}$ i.e., lists of sets of cubes

$$h_i = \{ b_j \mid a_i b_j \in f \} \ \forall i$$

i.e., all the multipliers of the cube a_i in g that produce elements of f are in h_i

Theorem: $f/g = \bigcap_{i=1}^{|g|} h_i = h_1 \cap h_2 \ \ldots \ h_{|g|}$

Weak Division Example

$$f = abc + abde + abh + bcd$$
$$g = c + de + h$$

Theorem says $f/g = f/c \cap f/de \cap f/h$

$$f/c = ab + bd$$
$$f/de = ab$$
$$f/h = ab$$

$$f/g = (ab + bd) \cap ab \cap ab = ab$$

$$f = ab(c + de + h) + bcd$$

Time complexity: O($|f| \ |g|$). $|f|$ the number of cubes in f

Types of Algebraic Divisors

Define divisors of f as the set
$D(f) = \{\ g \mid f/g \neq \phi\ \}$

Define primary divisors of f as
$P(f) = \{\ f/c\ \mid c$ is a cube $\}$

Example: $f = abc + abde$
$f/a = bc + bde$ is a primary divisor

Every divisor of *f* is contained in a primary divisor. If g divides f, then $g \subseteq p \in P(f)$

g is termed "cube-free" if the only cube dividing g evenly is 1.

Kernels and Divisors

Define the kernels of f as
$K(f) = \{\ k \mid k \in P(f),\ k$ is cube-free $\}$

Example: $f = abc + abde$

$f/a = bc + bde$ is a primary divisor but is not cube-free since b is a factor $f/a = b(c + de)$

$f/ab = c + de$ is a kernel

ab is the co-kernel

The co-kernel of a kernel is not unique.

Examples

Consider

$$f = acd + bcd + ae + be$$

DIVISOR TYPE?

$$f/a = cd + e$$

$$f/c = ad + bd$$

$$f/cd = a + b$$

$$f/e = a + b$$

Common Divisors and Kernels

Goal of multi-level logic optimizer is to find common divisors of two (or more) functions f and g

Theorem: f and g have a non-trivial common divisor d ($d \neq$ cube) if and only if there exist kernels

$k_f \in K(f),\ k_g \in K(g)$ such that

$k_f \cap k_g$ is non-trivial, i.e., not a cube

$\therefore$ can use kernels of f and g to locate common divisors

Algorithm to find All Kernels

Kernels(f)
- Find c_f so f/c_f is cube-free ;
- K = Kernel1(0, f/c_f) ;
- if (f is cube-free)
  - return($f \cup K$) ;
- return(K) ;

Kernel1(j, g) {

    $R = g$;

    /* n = number of literals */

    for ($i = j + 1;\ i \leq n;\ i = i + 1$) {

        if (l_i in one or no terms) continue ;

        c_e = Max. literal cube evenly dividing $(g/l_i)\, l_i$;

        if (l_k not in c_e, for all $k \leq i$)

            $R = R \cup$ Kernel1(i, $(g/l_i)/c_e$)

    }

    return(R) ;

}

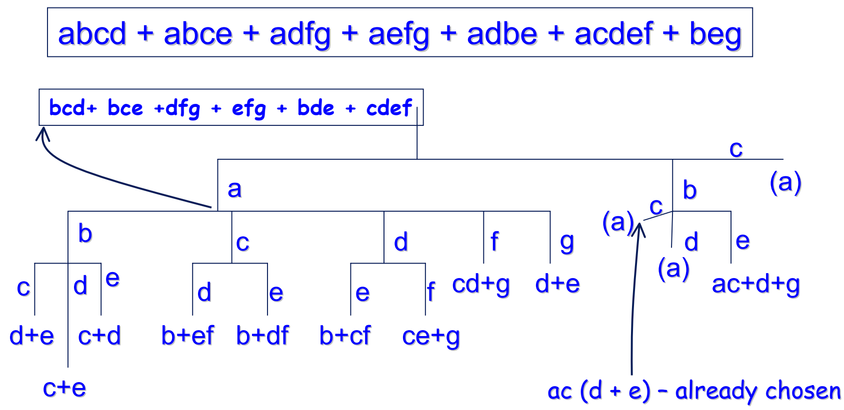
{Presume ordering on literals}

Kerneling Example

$$f = abcd + abce + adgh + aegh + abde + acdeg + beh$$

co-kernel	kernel
1	$a(bc + gh)(d + e) + ade(b + cg) + beh$
a	$(bc + gh)(d + e) + de(b + cg)$
ab	$c(d + e) + de$
abc	$d + e$
abd	$c + e$
ac	$b(d + e) + deg$
acd	$b + eg$
ace	$b + dg$
ad	$b(c + e) + g(ce + h)$
ade	$b + cg$
adg	$ce + h$
⋮	

Kerneling Illustrated



Pruning Condition and Example

If the largest cube factor c_e contains an already selected literal, then terminate current branch

All kernels found by continuing have already been seen

$$f = abc(d+e)(k+l) + agh + m$$

a

$$f/a = bc(d+e)(k+l) + gh$$

b $\qquad$ c

$$f/ab = c(d+e)(k+l) \qquad f/ac = b(d+e)(k+l)$$

$$(d+e)(k+l)$$

***b* already selected**

Orchestration of Optimization Techniques

Technology-independent:

- two-level minimization
- selective collapsing
- algebraic decomposition
- restructuring for timing
- redundancy removal
- transduction
- global-flow

Technology-dependent:

- **tree covering**
- **load buffering**
- **rule-based mapping**
- **signature analysis**
- **inverter phase assignment**
- **discrete sizing**

Logic optimization - summary

Current formulation of logic synthesis and optimization is the most common techniques for designing integrated circuits today

Has been the most successful design paradigm 1989 - present

Almost all digital circuits are touched by logic synthesis

- **Microprocessors (control portions/random glue logic ~ 20%)**
- **Application specific standard parts (ASSPs)- 20 - 90%**
- **Application specific integrated circuits (ASICS) - 40 - 100%**

Real logic optimization systems orchestrate optimizations

- **Technology independent**
- **Technology dependent**
- **Application specific (e.g. datapath oriented)**

Computing f/g

Given $f=\{c_i\},\ g=\{a_i\}$

1) Encode cubes $a_i \in g$ with unique integer codes, by assigning a unique bit position for every literal in $sup(g)$
   e.g., $g = ab + e$
   110 001

2) Encode cubes $c_j \in f$ similarly
   e.g., $f = abc + abd + de$
   110 110 001

3) Sort $\{a_i, c_j\}$ by their codes
   e.g., $\underbrace{ab,\ abc,\ abd,}_{110}\ \underbrace{e,\ de}_{001}$
   $h_1 = c + d \qquad h_2 = d$

Time complexity?