Technology Independent Multi-Level Logic Optimization

Prof. Kurt Keutzer

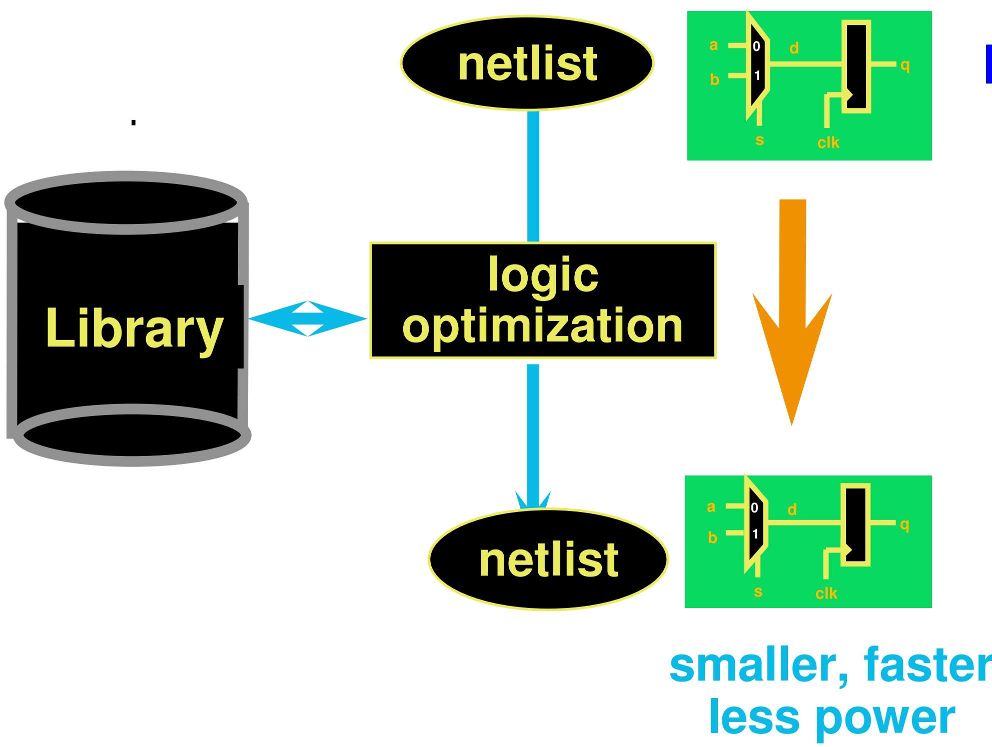
Prof. Sanjit Seshia

EECS

University of California, Berkeley, CA

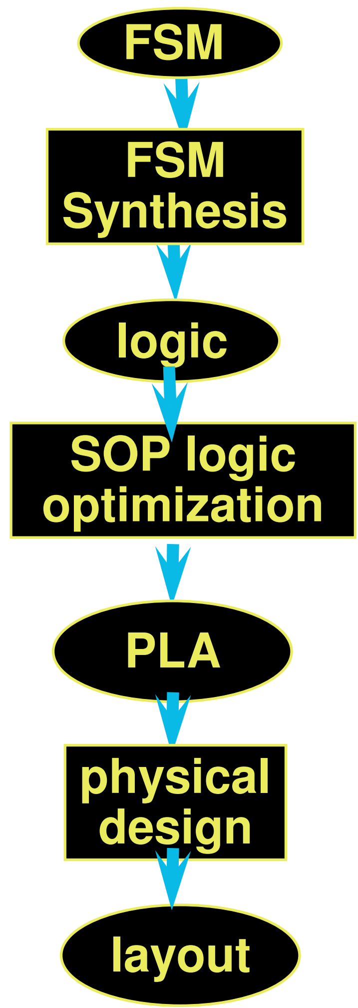
Thanks to R. Rudell, S. Malik, R. Rutenbar

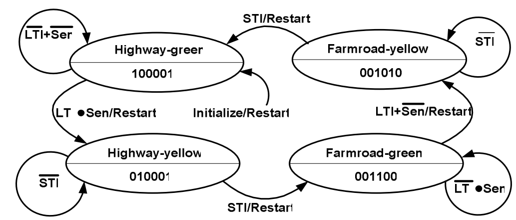
Logic Optimization



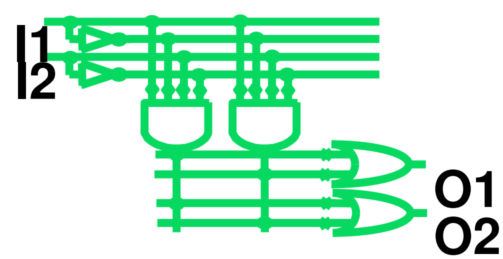
- Perform a variety of transformations and optimizations
  - Structural graph transformations
  - Boolean transformations
  - Mapping into a physical library

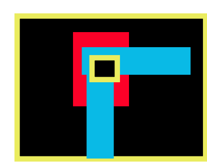
What We’ve Seen: Early “Synthesis” Flow & 2-Level Minimization



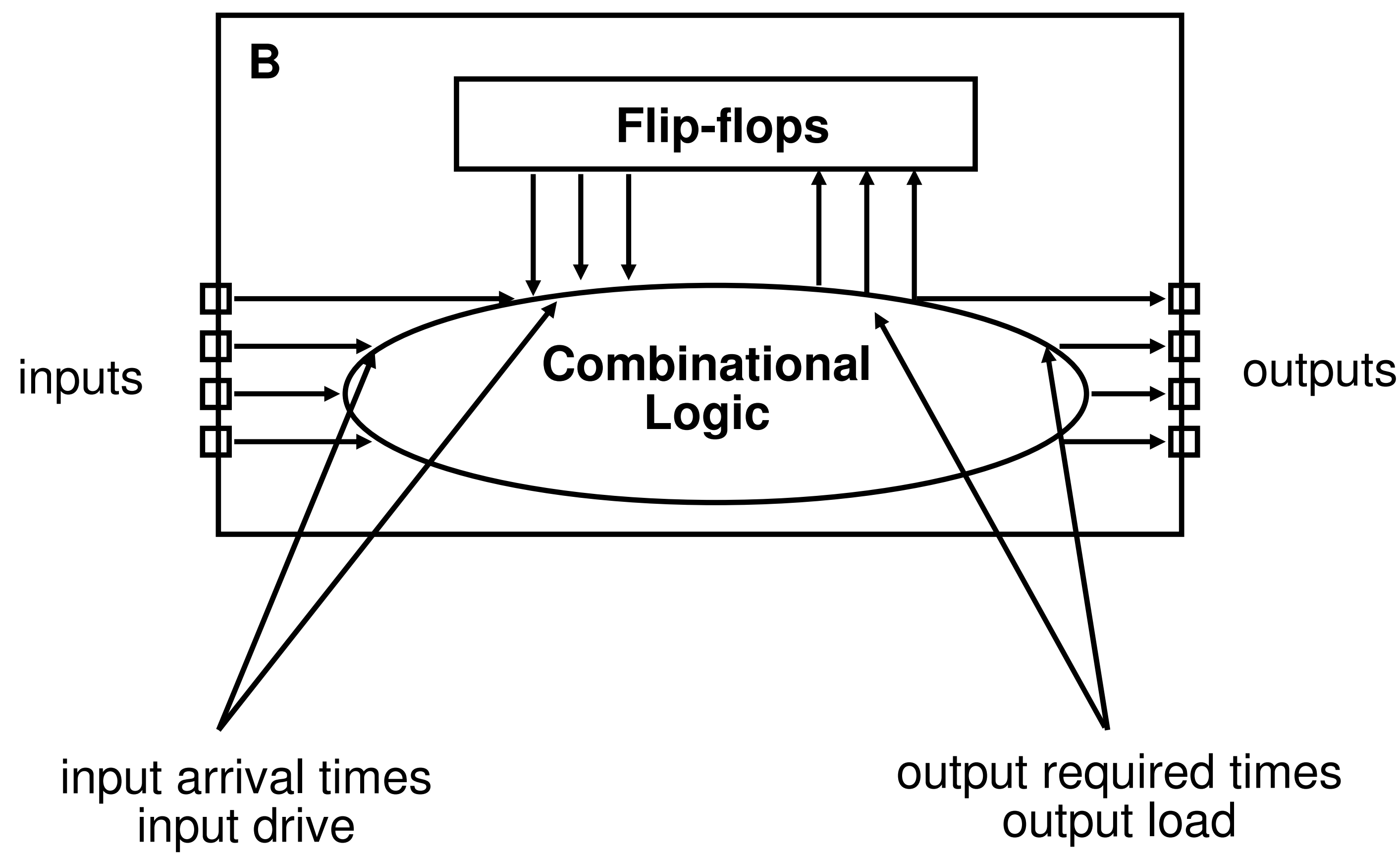


F1 = B + D + A C + A C





Reduce to Combinational Optimization



Combinational Logic Optimization

Input:

Initial Boolean circuit

Timing characterization for the module

- input arrival times and drive factors
- output loading factors

Optimization goals

- output required times output load

Target library description

Output:

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

A very difficult optimization problem !

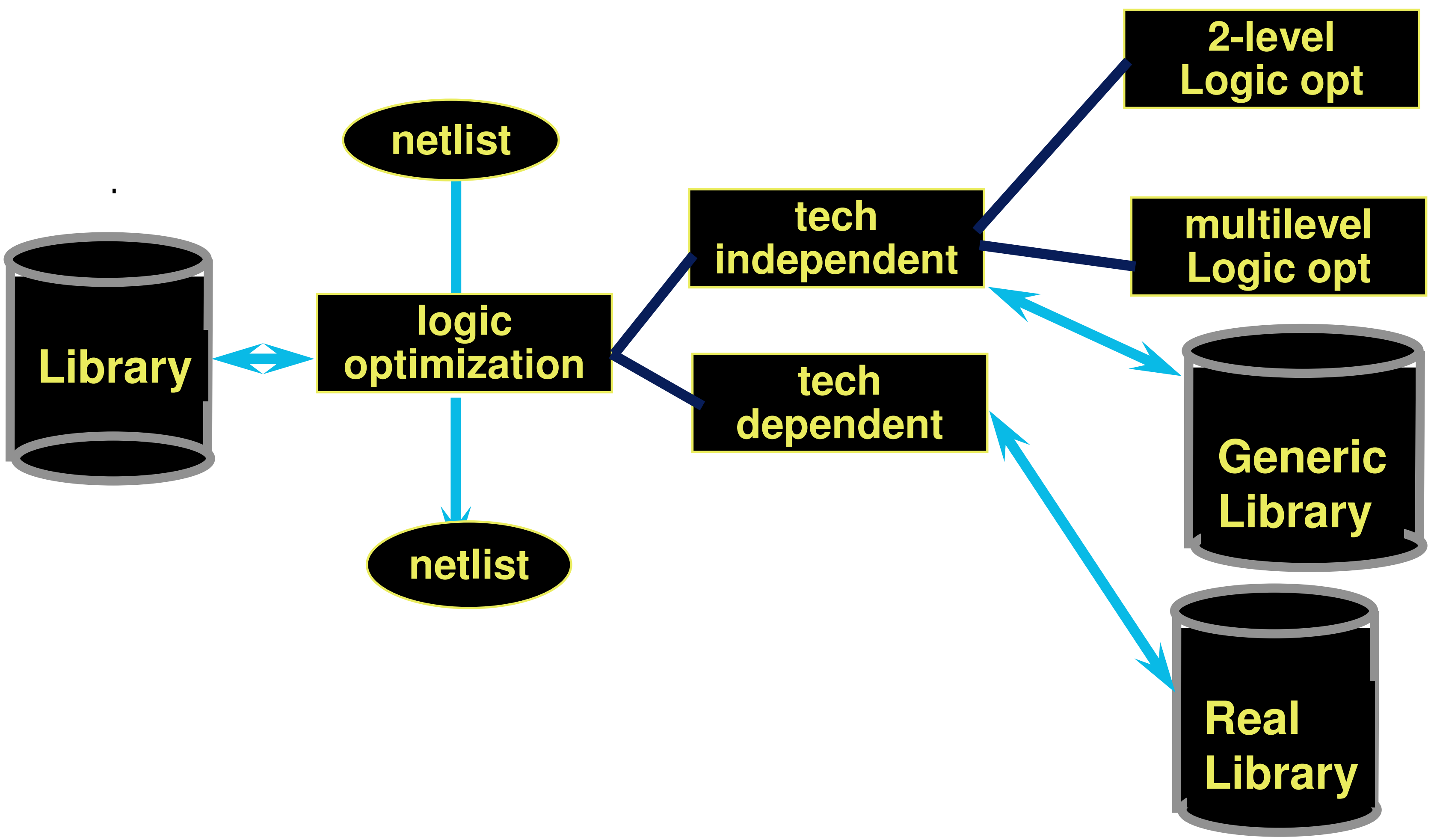
Modern Approach to Logic Optimization

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

Logic Optimization



Outline

- Motivation for Multilevel Ckts
- Overview of Multilevel Optimization
- Details on Multilevel Optimization Techniques

Why Multilevel Combinational Circuits?

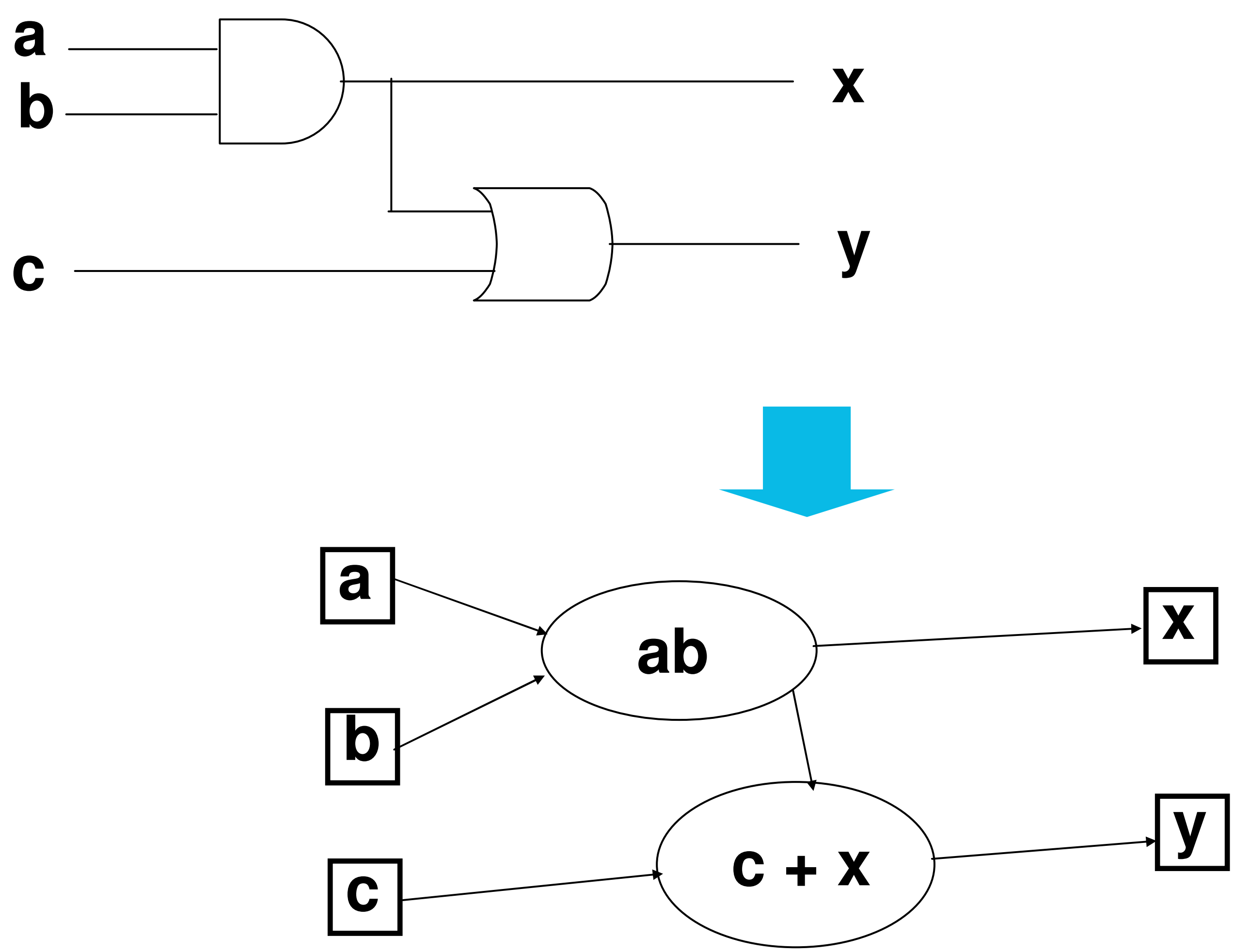
- There are many functions that are too "expensive" to implement in two-level form
  - Try 16-bit adder $\Rightarrow$ 32 input lines and 2^{16} product terms!

- Delay vs. Area tradeoff
  - 2-level ckt: tiny delay, large area (many gates & literals)
  - multi-level: bigger delay, less area

Outline

- **Motivation for Multilevel Ckts**
- **Overview of Multilevel Optimization**
- **Details on Multilevel Optimization Techniques**

Representation: Boolean Network



Boolean Network, Explained

It's a graph:

- Primary inputs (variables)
- Primary outputs
- Intermediate nodes (in SOP form in terms of its inputs)

Generic library – technology independent

- Has standard functions – ND2, ND4, AOI22

- Quality of network: area, delay, …
  - measured in terms of #(literals), depth, …

Tech.-Independent Multi-Level Optimization: Operations

Involves performing the following operations "iteratively" until "good enough" result is obtained:

1. Simplification
   - Minimizing two-level logic function (SOP for a single node)
2. Elimination
   - Substituting one expression into another.
3. Decomposition
   - Expressing a single SOP with 2 or more simpler forms
4. Extraction
   - Finding & pulling out subexpressions common to many nodes
5. Substitution
   - Like extraction, but nodes in the network are re-used

Example

(due to G. De Micheli)

Inputs: a, b, c, d, e

$v = a'd + bd + c'd + ae'$

$p = ce + de$

$r = p + a'$

$s = r + b'$

$t = ac + ad + bc + bd + e$

$q = a + b$

$u = q'c + qc' + qc$

Outputs: w, x, y, z

#literals = 33, depth = 3

Example: Elimination

a $\quad$ $v = a'd + bd + c'd + ae'$ $\quad$ w

b

$p = ce + de$ $\quad$ $r = p + a'$ $\quad$ $s = r + b'$ $\quad$ x

c

d $\quad$ $t = ac + ad + bc + bd + e$ $\quad$ y

e $\quad$ $q = a + b$ $\quad$ $u = q'c + qc' + qc$ $\quad$ z

#literals = 33, depth = 3

Example: Eliminate node r

a

b

c

d

e

v = a'd + bd + c'd + ae'

p = ce + de

s = p + a' + b'

t = ac + ad + bc + bd + e

q = a + b

u = q'c + qc' + qc

w

x

y

z

#literals = 32, depth = 2

Example: Simplification

a

b

c

d

e

v = a'd + bd + c'd + ae'

p = ce + de

s = p + a' + b'

t = ac + ad + bc + bd + e

q = a + b

u = q'c + qc' + qc

w

x

y

z

#literals = 32, depth = 2

Example: Simplifying node u

a

b

c

d

e

$v = a'd + bd + c'd + ae'$

$p = ce + de$

$s = p + a' + b'$

$t = ac + ad + bc + bd + e$

$q = a + b$

$u = q + c$

w

x

y

z

#literals = 28, depth = 2

Example: Decomposition

a, b, c, d, e

$v = a'd + bd + c'd + ae'$

$p = ce + de$

$s = p + a' + b'$

$t = ac + ad + bc + bd + e$

$q = a + b$

$u = q + c$

w, x, y, z

#literals = 28, depth = 2

Example: Decomposing node v

a, b, c, d, e

$j = a' + b + c'$

$v = jd + ae'$

$p = ce + de$

$s = p + a' + b'$

$t = ac + ad + bc + bd + e$

$q = a + b$

$u = q + c$

w, x, y, z

#literals = 27, depth = 2

Example: Extraction

a, b, c, d, e

$j = a' + b + c'$

$v = jd + ae'$

$p = ce + de$

$s = p + a' + b'$

$t = ac + ad + bc + bd + e$

$q = a + b$

$u = q + c$

w, x, y, z

#literals = 27, depth = 2

Example: Extracting from p and t

a, b, c, d, e

$j = a' + b + c'$

$p = ke$

$k = c + d$

$q = a + b$

$v = jd + ae'$

$s = p + a' + b'$

$t = ka + kb + e$

$u = q + c$

w, x, y, z

#literals = 23, depth = 3

Example: What next?

a | b | c | d | e

$j = a' + b + c'$

$p = ke$

$k = c + d$

$q = a + b$

$v = jd + ae'$

$s = p + a' + b'$

$t = ka + kb + e$

$u = q + c$

w | x | y | z

#literals = 23, depth = 3

Which Operations Do We Know How to Do?

Involves performing the following operations "iteratively" until "good enough" result is obtained:

1. **Simplification**
   - **Minimizing two-level logic function (SOP for a single node)**
2. **Elimination**
   - **Substituting one expression into another.**
3. **Decomposition**
   - **Expressing a single SOP with 2 or more simpler forms**
4. **Extraction**
   - **Finding & pulling out subexpressions common to many nodes**
5. **Substitution**
   - **Like extraction, but nodes in the network are re-used**

Outline

- **Motivation for Multilevel Ckts**
- **Overview of Multilevel Optimization**
- **Details on Multilevel Optimization Techniques**

Need for Factoring/Division

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

What we need is a way to do "division"

Divisors and Decomposition

Given Boolean function F, we want to write it as

$$F = D \,.\, Q + R$$

where D – Divisor, Q – Quotient, R – Remainder

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

- **identify divisors which are common to several functions**
- **introduce common divisor as a new node**
- **re-express existing nodes using the new divisor**

Boolean Division

Given Boolean function F, we want to write it as

$$F = D \cdot Q + R$$

- D is a factor of F iff $F \cdot D' = 0$
  - ON-SET(D) contains ON-SET(F)
- If $F \cdot D \neq 0$, then D is a divisor of F
- How many possible factors D can there be for a given F?

Algebraic Model

Idea: Perform division using only the rules (axioms) of real numbers, not all of Boolean algebra

Real Numbers	Boolean Algebra
a.b = b.a	a.b = b.a
a+b = b+a	a+b = b+a
a.(b.c) = (a.b).c	a.(b.c) = (a.b).c
a+(b+c) = (a+b)+c	a+(b+c) = (a+b)+c
a.(b+c) = a.b + a.c	a.(b+c) = a.b + a.c
a.1 = a a.0 = 0 a+0 = a	a.1 = a a.0 = 0 a+0 = a
	a+(b.c) = (a+b). (a+c)
	a+a’ = 1 a.a’ = 0 a.a = a a+a=a
	a+1 = 1 a+ab = a a.(a+b) = a
	…

Algebraic Division

- **A literal and its complement are treated as unrelated**
  - Each literal as a fresh variable
  - E.g.

    $f = ab + a'x + b'y$ as $f = ab + dx + ey$
- **Treat SOP expression as a polynomial**
  - Division/factoring then becomes polynomial division/factoring
- **Boolean identities are ignored**
  - Except in pre-processing
  - Simple local simplifications like $a + ab \rightarrow a$ performed

Algebraic vs. Boolean factorization

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

Algebraic factorization produces

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

Boolean factorization produces

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

Algebraic Division Example

$F = ac + ad + bc + bd + e$

Want to get Q, R, where $F = D\,Q + R$ for

1. $D = a + b$

2. $D = a$

Algebraic Division Algorithm

- **What we want:**
  - Given F, D, find Q, R
  - F, D expressed as sets of cubes (same for Q, R)
- **Approach:**
  - For each cube C in D {

```
  let B = {cubes in F contained in C}
  if (B is empty) return Q = { }, R = F
  let B = {cubes in B with variables in C removed}
  if (C is the first cube in D we're looking at)
    let Q = B;
  else Q = Q ∩ B;
}
R = F \ (Q x D);
```

Complexity?

Taking Stock

- **What we know:**
  - How to perform Algebraic division given a divisor D
- **What we don't**
  - How to pick D?
- **Recall what we wanted to do:**
  - Given 2 functions F and G, find a common divisor D and factorize them as

$$F = D\,Q1 + R1$$

$$G = D\,Q2 + R2$$

New Terminology: Kernels

- A **kernel** of a Boolean expression F is a **cube-free expression** that results when you divide F by a single cube
  - That "single cube" is called a **co-kernel**

- Cube-free expression: Cannot factor out a single cube that leaves behind no remainder

- Examples: Which are cube-free?
  - F = a
  - F = a + b
  - F = abc + abd

Kernels: Examples

F = ae + be + cde + ab

K(f)

Kernel	Co-kernel
{a,b,cd}	e
{e,b}	?
?	b
{ae,be,cde,ab}	?

Why are Kernels Useful?

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

Theorem: [Brayton & McMullen]

f and g have a non-trivial (multiple-cube) common divisor d 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

Theorem, Put Another Way

- F = D1 . K1 + R1
- G = D2 . K2 + R2

- K1 = (X + Y + …) + stuff1
- K2 = (X + Y + …) + stuff2

- Then,
  - F = (X + Y + …) D1 + stuff3
  - G = (X + Y + …) D2 + stuff4

- So, if we find kernels and intersect them, the intersection gives us our common divisor

Kernel Intersection: Example

F = ae + be + cde + ab

G = ad + ae + bd + be + bc

K(f)

Kernel	Co-kernel
{a,b,cd}	e
{e,b}	a
{e,a}	b
{ae,be,cde,ab}	1

K(g)

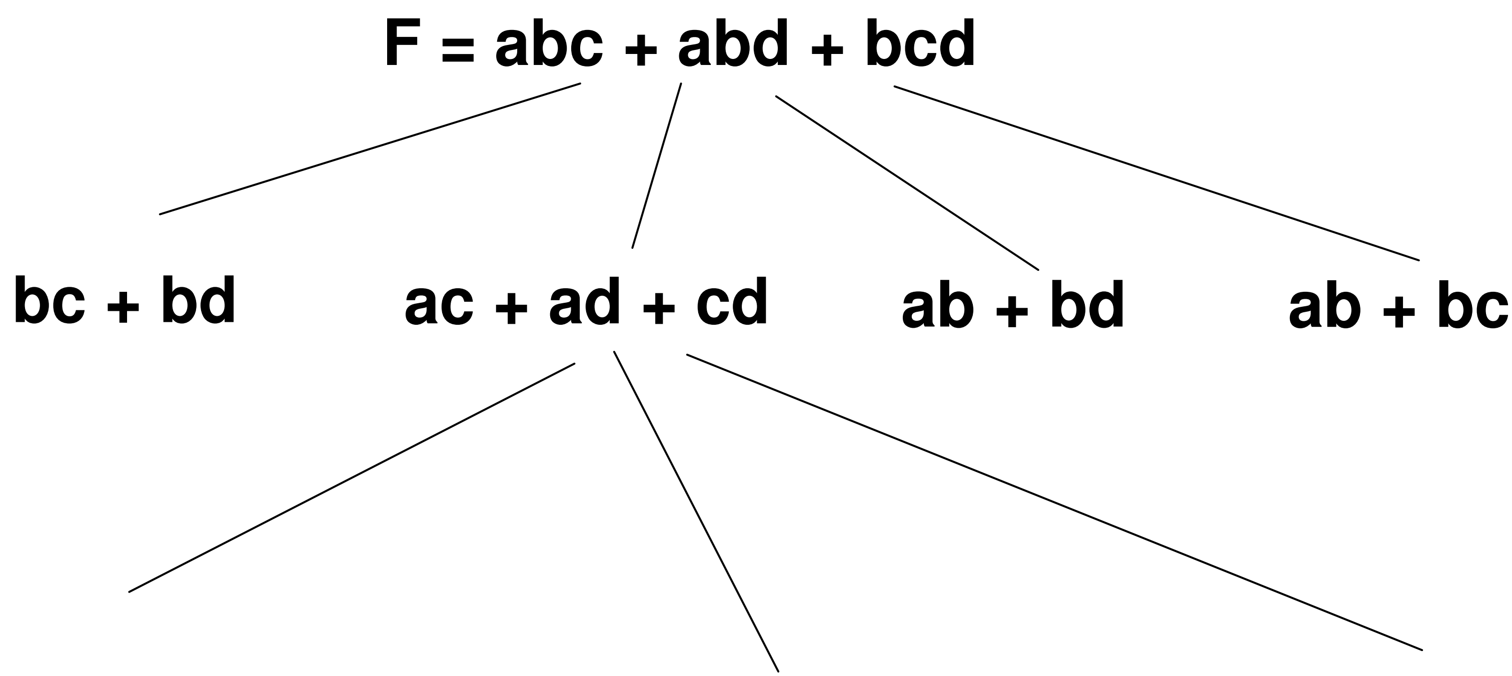
Kernel	Co-kernel
{a,b}	d or e
{d,e}	a or b
{d,e,c}	b
{ad,ae,bd,be,bc}	1

How do we find Kernels?

Overview: Given a function F

1. **Pick a variable x appearing in F, and use it as a divisor**
2. **Find the corresponding kernel K if one exists (at least 2 cubes in F contain x)**
   - **If not, go back to (1) and pick another variable**
3. **Use K in place of F and recurse to find kernels of K**
   - **F = x K + R and K = y M + S → F = xy M + …**
   - **Add kernels of K to those of F**
4. **Go back to (1) and pick another variable to keep finding kernels**

Finding Kernels: Example



Can we do better?

Finding Kernels: Example

F = abc + abd + bcd

bc + bd	abc + ad + cd	abc & bcd both contain c. Intersection is bc. Recurse on F/bc = a+d	ab + bc

Take intersection of all cubes containing a variable

Kernel Finding Algorithm

```
FindKernels(F) {
 K = { };
 for (each variable x in F) {
   if (F has at least 2 cubes containing x) {
     let S = {cubes in f containing x};
     let c = cube resulting from intersection of all cubes in S
     K = K ∪ FindKernels(F/c);  //recursion
    }
 }
 K = K ∪ F ;
 return K;
}
```

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

$$h_{DC} = G\bar{g} + \overline{G}g$$

Inputs to f that cannot occur

$$h_{ON} = f_{ON} - h_{DC}$$

$$h_{OFF} = \overline{f_{ON} + h_{DC}}$$

Minimize h using two-level minimizer

Get: h = Q G + R

Additional Reading

1. Read Chapter 7 upto Sec. 7.6

2. R. Rudell, ‘Logic Synthesis for VLSI Design’, PhD Thesis, UC Berkeley, 1989.

3. R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, ‘Multilevel Logic Synthesis’, Proceedings of the IEEE, Feb’90.

OPTIONAL

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 ← NEXT LECTURE
- Application specific (e.g. datapath oriented)