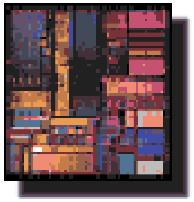
# ECE 595Z
# Digital VLSI Design Automation

## Module 5 (Lectures 14-20): Multi-level Synthesis
## Lecture 15




Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

# Lecture #14: Re-cap

- Optimization problems involved in multi-level synthesis are MUCH more computationally challenging than two-level synthesis
- Iterative improvement strategy: Transformations iteratively applied to Boolean network
  - Decomposition, Elimination, Extraction, Simplification
  - Common operation: Identifying factors

# Multi-level synthesis operations: Summary

- Global
  - Extraction
    - Find common **factors** for two or more nodes
  - Elimination / Collapsing
    - Collapse a node into it's fan-outs
  - Re-substitution
    - Re-substitute node into other nodes in the network through **factoring**
- Local
  - Decomposition
    - Decompose a node through **factoring**
  - Simplification
    - Use ESPRESSO to simplify the expression inside a node

# Factoring

- Common operation involved in multi-level network transformations
- Two models (classes of techniques)
  - Algebraic
  - Boolean

# Algebraic Model for Boolean Expressions

- Think of Boolean expressions as polynomials in real valued variables
- Only use a sub-set of laws of Boolean algebra that apply to Real numbers and polynomials
  - Do not use:
    - $x + x' = 1$
    - $x.x' = 0$
    - $x + x = x$
    - $x.x = x$
    - $x + y.z = (x + y)(x + z)$
    - $x + x.y = x$
    - $x.(x + y) = x$
    - De Morgan's law

# Algebraic Expressions and Operations

- **Definition**: An SOP expression *f* is an *algebraic expression* if no single cube contains another (minimal with respect to single cube containment)

  Examples:

  *ab + ac* is an algebraic expression

  *a + ab* is NOT an algebraic expression (*a* contains *ab*)

Treat a and a' as DIFFERENT variables

- **Definition**: *fg* is an *algebraic product* if *f* and *g* are algebraic expressions and have *disjoint support* (that is, they have no variables in common)

  Examples:

  *(a+b)(c+d) = ac+ad+bc+bd* is an algebraic product

  *(a+b)(a+d) = a + bd* is NOT an algebraic product

# Algebraic Division

- Boolean algebra does not have a multiplicative inverse ($x.x^{-1} = 1$)
  - No division operation
- However, we can define the concept of **algebraic division**
  - Given two expressions f and p, p is a divisor of f if there are expressions q and r such that
    - $f = p \cdot q + r$
    - p.q is an algebraic product
    - q : quotient, r : remainder
- If remainder is 0, the quotient and divisor are exact factors

Example:
f = abc + abd + h

f = ab(c + d) + h

# Algebraic Division : Examples

Example:
f = ac + ad + bc + bd + e

| Divisor | Quotient | Remainder | Exact Factor? |
|---|---|---|---|
| ac + ad + bc + bd + e | | | |
| a + b | | | |
| c + d | | | |
| a | | | |
| b | | | |
| c | | | |
| d | | | |
| e | | | |
| 1 | | | |

# Algebraic Division

- How do we do it systematically (an algorithm)?

# Algebraic Division Algorithm : Weak Division

- Simple algorithm that implements algebraic division
  - Walk through all pairs of cubes in F and G

```
ALGORITHM WEAK_DIVISION(F,G) {
  // G={g1,g2,...}, F=(f1,f2,...}
  foreach gi∈ G {
    V^gi=∅
    foreach fj ∈ F {
      if(fj contains all literals of gi) {
        vij=fj - literals of gi
        V^gi=V^gi ∪ vij
      }
    }
  }
  H = ∩i V^gi
  R = F - GH
  return (H,R);
}
```

F, G are SOP expressions (set of cubes)

Divide F by cube $g_i$

Take intersection of quotients obtained for all $g_i$

**Note: Use algebraic interpretation!**

**abc contains ab**

**abc – literals of ab = c**

**$c \cup d = c + d$**

**$(a + b + c) \cap (b + c + d) = b + c$**

**(abc + abd + e ) – (abc + abd) = e**

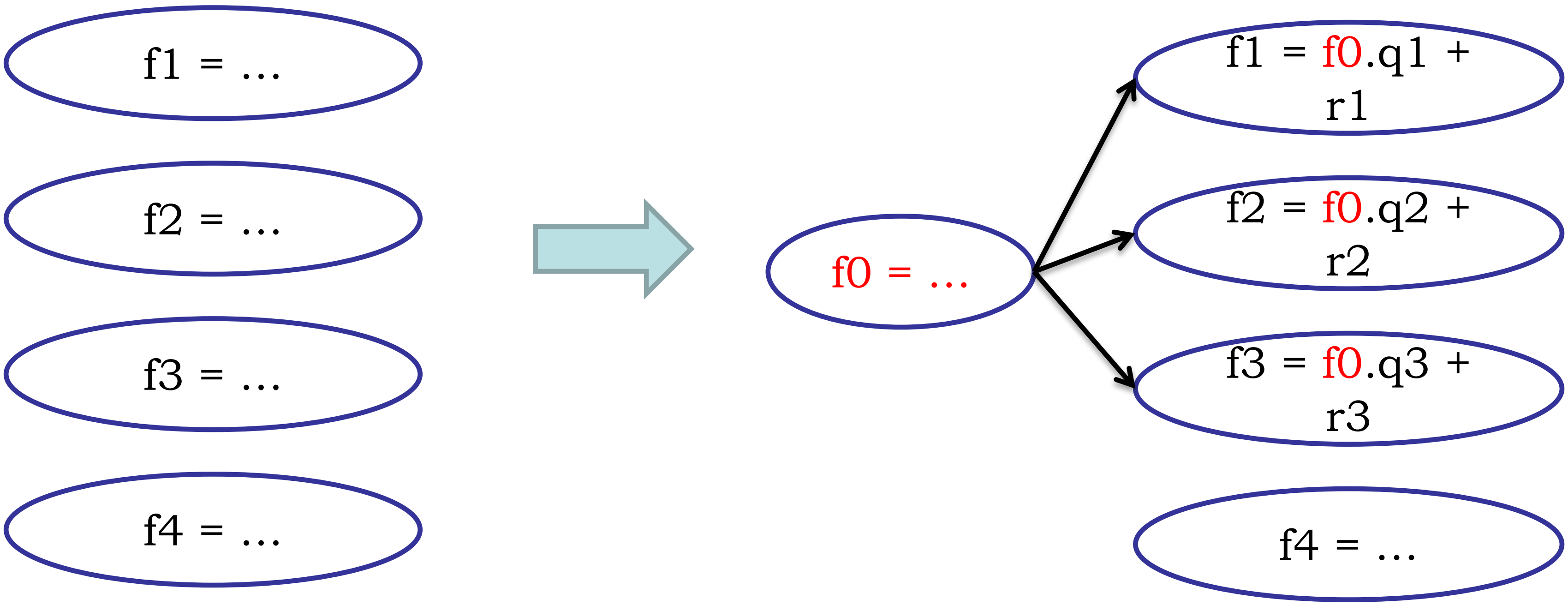
# Weak Division : Example

F = axc + axd + axe + bc + bd + e

G = ax + b

# Algebraic Division and Factoring

- Common Divisors of multiple functions can be used for factoring




Number of possible divisors can be LARGE!

# Finding Common Divisors (Efficiently)

- Recall that divisors are also SOP expressions
  - Case 1 : Divisors are a single cube
    - e.g., abc
  - Case 2 : Divisors consist of multiple cubes
    - e.g., a + c' + bd

- **Key idea**: Use of **Kernels** and **Co-kernels** to identify divisors

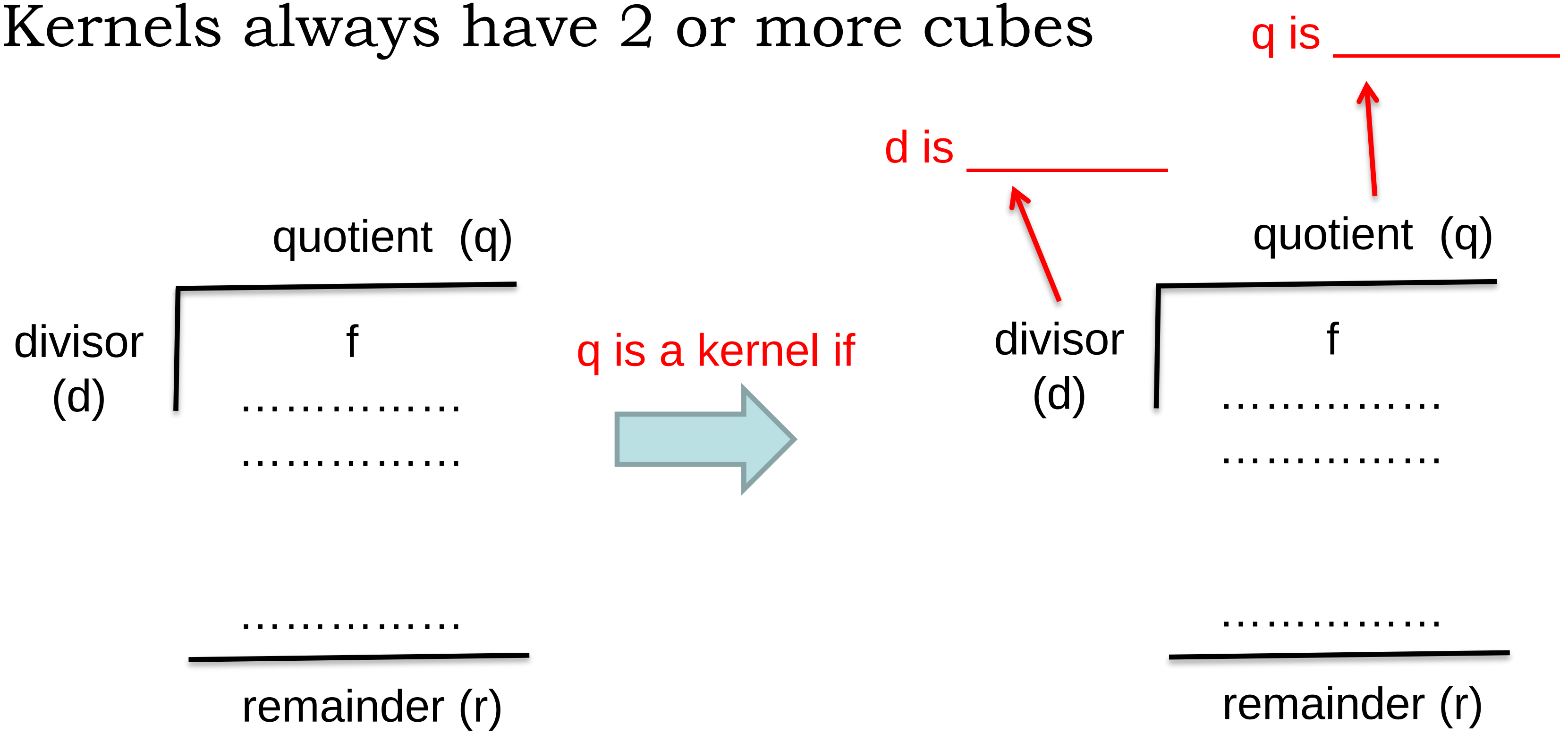
R. K. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," Proc. ISCAS, 1982.

# Kernels

- Primary divisors of a Boolean expression : Quotients obtained by dividing function with a single cube
  – D(F) = {F/c | c is a cube} denotes the set of primary divisors
- Kernels of a Boolean expression are it's cube-free primary divisors
  – K(F) = {g | g ∈ D(F) and g is cube-free} denotes the set of kernels
- Cube-free : You cannot factor out a single-cube divisor that leaves no remainder
  – e.g., bc + bde is NOT cube free, a+ b is cube free

# Kernels

- Intuitively, kernels result from dividing the function by "maximal" single-cube divisors
  - If quotient is not cube-free, you can always include additional literals in the divisor
- Kernels always have 2 or more cubes



# Kernels : Examples

- Consider f = abc + abd + bcd

| Divisor d | Quotient q | Is it a Kernel? |
|---|---|---|
| a | | |
| b | | |
| c | | |
| d | | |
| ab | | |
| ac | | |
| ad | | |
| bc | | |
| bd | | |
| cd | | |
| abc | | |
| acd | | |

# Kernels : Examples

- Do the following functions have kernels?

| f | Single-cube divisors | Kernels |
|---|---|---|
| a | | |
| a + b | | |
| ab + ac | | |
| abc + abd | | |
| ab + acd + bd | | |

# Co-Kernels

- Remember that kernels are obtained by dividing the expression with a divisor that is a cube
  - These single-cube divisors are called **co-kernels**
  - $C(F) = \{ c: F/c \in K(F)\}$ denotes set of co-kernels

Example:

$x = adf + aef + bdf + bef + cdf + cef + g$

$= (a + b + c)(d + e)f + g$

| kernels | co-kernels |
|---|---|
| $a+b+c$ | $df, ef$ |
| $d+e$ | $af, bf, cf$ |
| $(a+b+c)(d+e)$ | $f$ |
| $(a+b+c)(d+e)f+g$ | $1$ |

# Why care about kernels and co-kernels?

- Recall that we are looking for good single-cube and multi-cube common divisors
  - Kernels are multi-cube divisors
  - Co-kernels are single-cube divisors

- OK, but how do we find common divisors across different expressions?
  - Do we look at ALL POSSIBLE divisors of each expression?

# Finding Common Divisors using Kernels

- **Theorem [Brayton and McMullen]:** Given two expressions f and g, f and g have common algebraic divisors with more than one cube if and only if there exist $k_f \in K(f)$ and $k_g \in K(g)$ such that $|k_f \cap k_g| > 1$
- Significance : Only need to look at the kernels to identify all multi-cube common divisors for two functions!
  - Huge reduction in search space
- The "intersection" of the kernels IS the common divisor!
  - NOTE: In this context, "intersection" means common cubes

R. K. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," Proc. ISCAS, 1982.

# Finding Common Divisors using Kernels

- Example

f = ae + be + cde + ab

| Co-kernel | Kernel |
| --- | --- |
| a | |
| b | |
| e | |
| 1 | |

g = ad + ae + bd + be + bc

| Co-kernel | Kernel |
| --- | --- |
| a or b | |
| d or e | |
| b | |
| 1 | |

Multi-cube intersection of kernels of f and g:

Can each be used as a common divisor?

# Kernels: Summary

- Kernels are cube-free primary divisors of an expression
  - Intuitively: Quotients obtained when the expression is divided by "maximal" cubes
- Kernels tell us when two expressions have multi-cube common factors

# Finding Kernels

- Two key ideas
  1. Remember, we start off with co-kernels to get kernels
     - How do we efficiently find the set of co-kernels?

# Co-kernel Selection

- **Theorem [Brayton and McMullen]:** The co-kernels of an expression in SOP form correspond to the intersections of 2 or more of it's cubes
  - NOTE: In this context, "intersection" means just take the common literals.
    - Recall that we are only performing algebraic manipulations.

Example:
f = ace + bce + de + g

ace ∩ bce = _______

ace ∩ bce ∩ de = ______

} Potential co-kernels for f

# Finding Kernels

- Two key ideas
  1. Remember, we start off with co-kernels first to get kernels
     - How do we efficiently find the set of co-kernels?
  2. If k1 is a kernel of f, all kernels of k1 are also kernels of f
     - Suggests a recursive approach to kernel generation

# Recursive Computation of Kernels

- Consider a kernel k1 of an expression f
  - f = d1.k1 + r1, where d1 is the co-kernel of k1
- k1 itself is an expression, so it could have kernels. Let k2 be a kernel of k1
  - k1 = d2.k2 + r2
- Re-write f in terms of k2
  - f = d1.(d2.k2+r2) + r1
    = (d1.d2).k2 + (d1.r2 + r1)
    = d3.k2 + r3
- Important result
  - For $k \in K(f)$, $K(k) \in K(f)$
  - This "hierarchy" of kernels enables recursive computation

# Level of a Kernel

- A kernel is of level 0 ($K^0$) if it contains no kernels except itself.
- A kernel is of level $n$ ($K^n$) if it contains at least one kernel of level ($n$-$1$), but no kernels (except itself) of level $n$ or greater

Suppose $\mathbb{K}^n(F)$ denotes the set of kernels of level $n$ or less.

$\mathbb{K}^0(F) \subset \mathbb{K}^1(F) \subset \mathbb{K}^2(F) \subset \ldots \subset \mathbb{K}^n(F) \subset \ldots \subset \mathbb{K}(F)$.

level-$n$ kernels = $K^n = \mathbb{K}^n(F) - \mathbb{K}^{n-1}(F)$

Example:

$F = (a + b(c + d))(e + g)$

$k_1 = a + b(c + d) \quad \in K^1$

$\notin K^0$ - contains other kernel ($k_2$)

$k_2 = c + d \quad \in K^0$

$k_3 = e + g \quad \in K^0$

# Kernel Generation Algorithm

- Recursive algorithm – call it on any kernels that you find to discover additional kernels
- Processes variables in lexicographic order

```
KERNELS(j, G) {
  if (G is cube-free)
    R = {G};
  else R = {};
  for (i = j+1, …, n) {
    if (l_i appears in more than one cube) {
      if (∃k ≤ i, l_k ∈ all cubes of G/l_i )
        continue;
      else {
        R = R ∪ KERNELS(i, cube_free(G/l_i))
      }
    }
  }
return R;
}
```

A cube-free function is a trivial kernel for itself

Process remaining variables in lexicographic order

Speedup technique

Recursive call

KERNELS(0,f) returns all kernels of f

# Kernel Generation Illustrated

$F = ace + bce + de + g$ // n = 6 variables, $\{ a,b,c,d,e,g \}$

- Call KERNEL (0, F)
  - $R = \{(ace+bce+de+g)\}$
  - $i = 1, l_1 = a$ literal $a$ appears only once, continue
  - $i = 2, l_2 = b$ literal $b$ appears only once, continue
  - $i = 3, l_3 = c$
    - make_cube_free(F/c) = $(a+b)$
    - call KERNEL(3, $(a+b)$)
      - the call considers variables 4,5,6 = $\{d,e,g\}$ – no kernels
    - return $R = \{(a+b)\}$
  - $i = 4, l_4 = d$ literal $d$ appears only once, continue
  - $i = 5, l_5 = e$
    - make_cube_free(F/e) = $(ac+bc+d)$
    - call KERNEL(5, $(ac+bc+d)$)
      - the call considers variable 6 = $\{g\}$ – no kernels
      - return $R = \{(ac+bc+d)\}$
  - $i = 6, l_6 = g$ literal $g$ appears only once, continue
  - Stop, return $R = \{(a+b), (ac+bc+d), (ace+bce+de+g)\}$