VLSI System Design
Part II : Logic Synthesis (2)

Oct.2006 - Feb.2007

Lecturer : Tsuyoshi Isshiki

Dept. Communications and Integrated Systems,

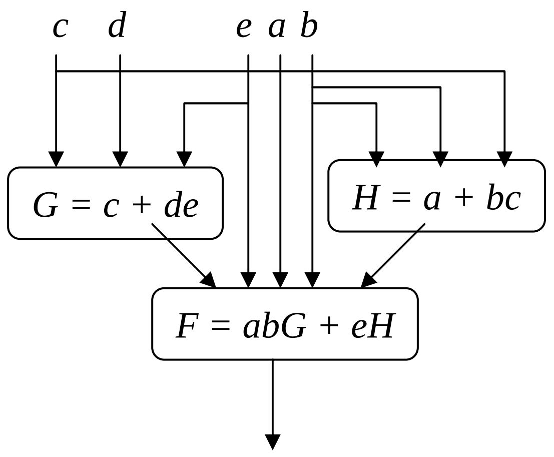
Tokyo Institute of Technology

isshiki@vlsi.ss.titech.ac.jp

http://www.vlsi.ss.titech.ac.jp/~isshiki/VLSISystemDesign/top.html

Multi-Level Logic Optimization (1)

- *Limitation of two-level logic*: When describing the logic circuit in two-level logic, even in its optimized form, the number of cubes can grow exponentially to the number of variables (such as arithmetic circuits).
- *Multi-level logic optimization* is done by iteratively modifying the structure. Objective functions can be area (circuit size), speed, and/or power consumption.
- *Boolean Network* : Directed acyclic graph *G*(*V*, *E*) to represent multi-level logic
  - Each node represents a two-level logic.
  - Each arc represents logic variables.



Multi-Level Logic Optimization (2)

- Operations of logic structure modification
  - Decomposition : decompose a function (node) described in sum-of-product form into multiple smaller functions by factoring
    - Ex : $F = a \cdot b + b \cdot c + a \cdot c \rightarrow G = a + b, F' = c \cdot G + a \cdot b$
  - Extraction : decompose multiple functions with common subfunctions
    - Ex : $F = a \cdot b + b \cdot c + a \cdot c, G = a \cdot d + b \cdot d + a \cdot c,$
      $\rightarrow H = a + b, F' = c \cdot H + a \cdot b, G' = d \cdot H + a \cdot c$
  - Collapsing : merge multiple functions into a single function by expanding into sum-of-product form
    - Ex : $G = a + b, F' = c \cdot G + a \cdot b \rightarrow F'' = a \cdot b + b \cdot c + a \cdot c$

Multi-Level Logic Optimization (3)

- Circuit size and number of literals
  - Any logic circuit can be implemented using only 2-input NAND gates (NAND2 gates) and inverters (INV gates).
  - Let $L(f)$ be the number of literals for node f. The number of NAND2 gates required to implement f is $L(f) - 1$. *Note that this is independent of the actual functionality*.
  - The number of NAND2 gates to implement *N* nodes f_i $(i = 1, 2, \ldots, N)$ is $\sum_i L(f_i) - N$.
  - ➢ The term "gate count" usually refers to the number of NAND2 gates in the circuit implemented only with NAND2 gates and INV gates. This is easily calculated by counting the number of literals and the number of nodes in the Boolean network.

Boolean Division

- Given two nodes f and g, decomposition $f = g \cdot q + r$ ($q \neq \phi$) is referred to as Boolean division. If there exists q such that $r = \phi$, g is the Boolean factor of f, otherwise g is the Boolean divisor of f.
- Optimization methods using Boolean division:
  - Factorization : For node f, compute g, q and r such that $f = g \cdot q + r$ and the total number of literals included in g, q and r is minimized. Repeat this operation on the newly added nodes (g, q and r) recursively.
  - Extraction : For nodes f_0 and f_1, compute the common Boolean divisor g and decompose these nodes to $f_0 = g \cdot q_0 + r_0$ and $f_1 = g \cdot q_1 + r_1$.
- Complications in Boolean division :
  - Given f and g, decomposition $f = g \cdot q + r$ can have multiple solutions for q and r. Therefore, merely computing the optimal divisor g can become too complicated.

Algebraic Division (1)

- The support for g, denoted as $sup(g)$ is defined as a set of variables which appear in g (either complemented or non-complemented). On functions g and q, if the two do not contain any common variables, g and q is said to be orthogonal and written as $g \perp q$ or $sup(g) \cap sup(q) = \phi$.
- Given f and g, if there exists $q \neq \phi$ such that $f = g \cdot q + r$ and $g \perp q$, then g is the algebraic divisor of f. Furthermore, if g is not the algebraic divisor of r ($q' \neq \phi$ which satisfies $r = g \cdot q' + r'$ does not exist), q is the algebraic quotient and written as $q = f / g$. In this case, r is the algebraic remainder (satisfies $r / g = \phi$).
- On algebraic division $f = g \cdot q + r$, if $r = \phi$, then g is said to *evenly divide* f. Here, g and q are both the algebraic factors of f.
- Algebraic division is a restricted form of Boolean division where the division solution becomes unique (can assume that variables are not Boolean but merely algebraic)

Algebraic Division (2)

- Changes in gate counts by algebraic division $f = g \cdot q + r$
  - On node f, let $L(f)$ be # of literals and $C(f)$ be # of cubes.
  - # of literals in the original node f (obtained by collapsing the right side expression) is calculated as $L(f) = L(g) \cdot C(q) + C(g) \cdot L(q) + L(r)$ (When collapsing the term $(g \cdot q)$, each literal in g is duplicated $C(q)$ times and each literal in q is duplicated $C(g)$ times). Here, # of gates required is $L(f) - 1$.
  - The 3 new nodes g, q and r includes a total of $L(g) + L(q) + L(r)$ literals. These 3 nodes consumes a total of $L(g) + L(q) + L(r) - 3$ gates.
  - After the decomposition, f is expressed with 3 literals g, q and r (this consumes 2 gates). This results in a total of $L(g) + L(q) + L(r) - 1$ gates after the decomposition.
  - Thus, **$L(g)\ (C(q) - 1) + L(q)\ (C(g) - 1)$** gates are saved by this decomposition.
- By applying the algebraic division on the two functions with a common divisor $f_0 = g \cdot q_0 + r_0$, $f_1 = g \cdot q_1 + r_1$:
  - **$L(g)\ (C(q_0) + C(q_1) - 1) + (L(q_0) + L(q_1))\ (C(g) - 1) - 1$** gates are saved.

Algebraic Functions and Sets

- In algebraic division, expressions are treated as algebraic functions.
- Algebraic functions and class calculus (set theory)
  - Variables do not represent a set, but merely an element for the cube set. Complemented and non-complemented variables are treated as separate names.
  - A cube is a set of literals.
  - A cover (sum-of-product) is a set of cubes.
  - Examples:

$f = ab + bc = \{ab, bc\}, g = b = \{b\}, h = ab = \{ab\}$

$f \supseteq h$, but $f \not\supseteq g$, $f \not\subseteq g$

(*in Boolean space* : $f \supseteq h , f \subseteq g , h \subseteq g$)

$f \cap g = \phi , f \cap h = ab$

(*in Boolean space* : $f \cap g = ab + bc, f \cap h = ab + abc = ab$)

Multi-Level Logic Optimization With Algebraic Division

- *Need* efficient implementation of algebraic division computation method (given a divisor)
- *Need* methods for selecting good divisors
  - Select divisors with large # of literals and cubes
    - $\rightarrow$ *cubes*, *kernels*
  - Select divisors which are common among multiple functions.
    - $\rightarrow$ *common cubes, nontrivial kernel intersections*

Algebraic Quotient Calculation (Method 1)

- Calculate f / g by intersecting cube factors sets
  - Describe the two given nodes f and g as sets of cubes:
    - $f = \{a_1, a_2, \ldots, a_{|f|}\}, g = \{b_1, b_2, \ldots, b_{|g|}\}$
  - For $i = 1, 2, \ldots, |g|$, calculate a set of *cube factors* on b_i with respect to each cube $a_j \in f$:
    - $q_i = \{c_{ij} \mid c_{ij} = a_j / b_i \neq \phi, a_j \in f, b_i \in g\}$.
  - $f / g = q_1 \cap q_2 \cap \ldots \cap q_{|g|}$.

    (note: operator $\cap$ is *set intersection*, not a *logical-AND*)

Ex : $f = ab\overline{c} + abd + \overline{c}e + \overline{b}ce + de, g = ab + e$

$f = \{ab\overline{c}, abd, \overline{c}e, \overline{b}ce, de\}, g = \{ab, e\}$

$q_0 = \{\overline{c}, d\}, q_1 = \{\overline{c}, \overline{b}c, d\}$

$f / g = q_0 \cap q_1 = \{\overline{c}, d\}$

$f = (ab + e)(\overline{c} + d) + \overline{b}ce$

Cube-Literal Matrix (1)

- Cube-literal matrix for algebraic division
  - Assign cubes to rows and literals (both non-complemented and complemented) to columns.
  - If the cube on the ith row includes the literal on the jth column, set element (i, j) to 1, otherwise set to 0.
  - Bit vectors assigned to each cube can be seen as indices.
    - $ind(abc) = 101100 = 44$
    - $ind(\overline{b}ce) = 010101 = 21$
  - The inverse of the index function can be defined also:
    - $ind^{-1}(001111) = bcde$
    - $ind^{-1}(101000) = ab$

Ex : $f = abc + abd + \overline{b}ce + de$

$$\Downarrow$$

$$\begin{array}{c|cccccc} & a & \overline{b} & b & c & d & e \\ \hline abc & 1 & 0 & 1 & 1 & 0 & 0 \\ abd & 1 & 0 & 1 & 0 & 1 & 0 \\ \overline{b}ce & 0 & 1 & 0 & 1 & 0 & 1 \\ de & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Cube-Literal Matrix (2)

- Propositions on cube-literal matrix
  1. For cubes c_i and c_j, c_j is an algebraic factor of c_i *if and only if*

     $$ind(c_i)\ \&\ ind(c_j) = ind(c_j)$$

     ('&' is a bitwise AND operator).
  2. If c_j is an algebraic factor of c_i, then the index value for the *evenly divided* quotient c_i / c_j can be written as

     $$ind(c_i / c_j) = ind(c_i) \wedge ind(c_j)$$

     or $$c_i / c_j = ind^{-1}((ind(c_i) \wedge ind(c_j)))$$

     ('^' is a bitwise EXOR operator).

Algebraic Quotient Calculation (Method 1’)

- Calculate f / g using cube-literal matrix

```
$a_k$ : cube on the $k$th row of the cube-literal matrix for $f$ ;
$b_k$ : cube on the $k$th row of the cube-literal matrix for $g$ ;
$q = U$ (universal set) ;
FOR ($i$ = 1 to $|g|$) {
        $q' = \phi$ ;
        FOR ($j$ = 1 to $|f|$) {
                // if $b_i$ is a factor of $a_j$ , add $a_j / b_i$ to $q'$
                IF ($ind(b_i)$ & $ind(a_j) = ind(b_i)$)
                        $q' = q' + \{ind^{-1}((ind(b_i) \wedge ind(a_j))\}$ ;
        }
        $q = q \cap q'$ ;
}
```

Calculation of Algebraic Quotient (3)

Ex : $f = ab\overline{c} + abd + \overline{c}e + \overline{b}ce + de,\ g = ab + e$

$$f = \{ab\overline{c}, abd, \overline{c}e, \overline{b}ce, de\},\ g = \{ab, e\}$$

$$q_0 = \{\overline{c}\ , d\},\ q_1 = \{\overline{c}\ , \overline{b}c, d\}$$

$$f\,/\,g = q_0 \cap q_1 = \{\overline{c}\ , d\}$$

$$f = (ab + e)\,(\overline{c} + d) + \overline{b}ce$$

	a	$\overline{b}$	b	$\overline{c}$	c	d	e
$f : ab\overline{c}$	1	0	1	1	0	0	0
$f : abd$	1	0	1	0	0	1	0
$f : \overline{c}e$	0	0	0	1	0	0	1
$f : \overline{b}ce$	0	1	0	0	1	0	1
$f : de$	0	0	0	0	0	1	1
$g : ab$	1	0	1	0	0	0	0
$q_0 : \overline{c}$	0	0	0	1	0	0	0
$q_0 : d$	0	0	0	0	0	1	0

	a	$\overline{b}$	b	$\overline{c}$	c	d	e
$f : ab\overline{c}$	1	0	1	1	0	0	0
$f : abd$	1	0	1	0	0	1	0
$f : \overline{c}e$	0	0	0	1	0	0	1
$f : \overline{b}ce$	0	1	0	0	1	0	1
$f : de$	0	0	0	0	0	1	1
$g : e$	0	0	0	0	0	0	1
$q_1 : \overline{c}$	0	0	0	1	0	0	0
$q_1 : \overline{b}c$	0	1	0	0	1	0	0
$q_1 : d$	0	0	0	0	0	1	0

Cube-Literal Matrix (3)

- Propositions on cube-literal matrix

3. For cubes c_i and c_j, if $ind(c_i) < ind(c_j)$, then c_j cannot be the algebraic factor of c_i.

   (applies to any column order)

4. For cubes $a_j \in f$, b_i , $b_k \in g$, let $a_j / b_k \neq \phi$ and $a_j / b_i \neq \phi$. If $ind(b_k) > ind(b_i)$, then $a_j / b_i \notin f / g$.

   (cube a_j / b_i is not included in the quotient f / g)

Algebraic Quotient Calculation (Method 2)

- Calculate f / g by cube-literal matrix with pruning

  Merge the two cube-literal matrices for *f* and *g*, and sort the rows in the increasing order of the cube indices.

```
$c_k$ : cube on the $k$th row ;
$max = | f | + | g |$ ; $k = max$ ; $q = U$ (universal set);
REPEAT {
        WHILE ($c_k \in f$ ) { // move $c_k$ to the last element of $g$ in the list.
                $k = k - 1$;
                if ($k = 0$) return $q$ ; // all elements are evaluated.
        }
        $q' = \phi$ ; mark row $k$ ;
        FOR ($j = k + 1$ to $max$) {
                IF ($ind(c_k)$ & $ind(c_j) = ind(c_k)$) {
                        $q' = q' + \{ind^{-1}((ind(c_j) \wedge ind(c_k))\}$ ;
                        mark row $j$ ;
                }
        }
        $q = q \cap q'$ ;
        IF ($q = \phi$) return $\phi$ ; // $f / g = \phi$
        delete all marked rows ;
        $k = k - 1$; $max = max$ – (# of marked rows) ;
}
```

Calculation of Algebraic Quotient (3)

Ex : $f = abc + abd + \bar{b}cd + ace + a\bar{b}de,\ g = ab + \bar{b}d + ae$

	a	$\bar{b}$	b	c	d	e
$g : \bar{b}d$	0	1	0	0	1	0
$f : \bar{b}cd$	0	1	0	1	1	0
$g : ae$	1	0	0	0	0	1
$f : ace$	1	0	0	1	0	1
$g : ab$	1	0	1	0	0	0
$f : abd$	1	0	1	0	1	0
$f : abc$	1	0	1	1	0	0
$f : a\bar{b}de$	1	1	0	0	1	1

	a	$\bar{b}$	b	c	d	e
$g : \bar{b}d$	0	1	0	0	1	0
$f : \bar{b}cd$	0	1	0	1	1	0
$g : ae*$	1	0	0	0	0	1
$f : ace*$	1	0	0	1	0	1
$f : a\bar{b}de*$	1	1	0	0	1	1

$q' = \{c, \bar{b}d\},\ q = \{c\}$

	a	$\bar{b}$	b	c	d	e
$g : \bar{b}d$	0	1	0	0	1	0
$f : \bar{b}cd$	0	1	0	1	1	0
$g : ae$	1	0	0	0	0	1
$f : ace$	1	0	0	1	0	1
$g : ab*$	1	0	1	0	0	0
$f : abd*$	1	0	1	0	1	0
$f : abc*$	1	0	1	1	0	0
$f : a\bar{b}de$	1	1	0	0	1	1

$q' = \{d, c\},\ q = \{d, c\}$

	a	$\bar{b}$	b	c	d	e
$g : \bar{b}d*$	0	1	0	0	1	0
$f : \bar{b}cd*$	0	1	0	1	1	0

$q' = \{c\},\ \boxed{q = \{c\}}$

Multi-Level Logic Optimization With Algebraic Division

- *Need* efficient implementation of algebraic division computation method (given a divisor)
- *Need* methods for selecting good divisors
  - Select divisors with large # of literals and cubes
    - $\rightarrow$ *cubes, kernels*
  - Select divisors which are common among multiple functions.
    - $\rightarrow$ *common cubes, nontrivial kernel intersections*

Primary Divisor (1)

- Primary divisor : $P(f) = \{ f / c \mid c$ is a cube$\}$
  (a set of algebraic quotients when dividing f by an arbitrary cube)
  - Ex: $f = abc + ade \rightarrow f / a = bc + de$ is a primary divisor.
- Theorem 3: If g is an algebraic divisor of f, then there exists $p \in P(f)$ such that $g \subseteq p$.
  - In other words, the set of primary divisors of f contains all possible divisor of f .
- Collorary 3.1 : $g \subseteq f / (f / g)$.
  - Consider $f = g \cdot q + r = g \cdot (f / g) + r$. $(r / g = \phi)$
  - Let $r = q \cdot q' + r'$. $(r' / q = \phi)$
  - Then $f = g \cdot q + r = g \cdot q + q \cdot q' + r' = (g + q') \cdot q + r'$

  $\therefore f / (f / g) = f / q = g + q' \supseteq g$

Primary Divisor (2)

- Collorary 3.2 : If $g \supseteq g'$, then $f/g \subseteq f/g'$.
  - Let $g = g' + g''$. Then :

    $f = g \cdot (f/g) + r = (g' + g'') \cdot (f/g) + r$

    $= g' \cdot (f/g) + g'' \cdot (f/g) + r$
  - Let $h = g'' \cdot (f/g) + r$. By dividing h with g', we get :

    $h = g' \cdot q' + r'$ where $r'/g' = \phi$. Then :

    $f = g' \cdot (f/g) + g' \cdot q' + r'$

    $= g' \cdot ((f/g) + q') + r'$
  - Since $r'/g' = \phi$, the algebraic quotient for f divided by g' is

    $f/g' = (f/g) + q'$

  $\therefore f/g \subseteq f/g'$

Primary Divisor (3)

- Theorem 3: If g is an algebraic divisor of f, then there exists $p \in P(f)$ such that $g \subseteq p$.
- Proof: For a cube $c \in f/g$,
  - Since $\{c\} \subseteq f/g$ and by corollary 3.2 $\rightarrow f/(f/g) \subseteq f/c$
  - By corollary 3.1 $\rightarrow g \subseteq f/(f/g)$

  $\therefore\ g \subseteq f/(f/g) \subseteq f/c \in P(f)$
- Cube-free : f is said to be *cube-free* if there are no cubes which divide f evenly other than the universal cube "1".

  (*a cube-free function obviously includes two or more cubes*)

Kernel (1)

- Kernel is a set of primary divisors which are cube-free:

$$K(f) = \{\, k \mid k \in P(f),\, k \text{ is cube-free}\}$$

- 0-level kernel set $K_0(f)$ is a set of kernel which do not contain other kernels. → <u>Often, 0-level kernel set is used for divisor candidates.</u>
- Ex: $f = abc + abde + abeg$.
  - $f / a = bc + bde + beg$ is a primary divisor of f but <u>*not*</u> a kernel since b is the algebraic factor of f / a.
  - $f / (ab) = c + de + eg$ is a kernel (there is no cube that can divide $c + de + eg$ evenly). Here, ab is called the *cokernel* of the kernel $f / (ab)$.
  - $f / (abe) = d + g$ is a *0-level kernel* (its cokernel is abe).

Kernel (2)

- Algebraic factorization on $f = ac + ad + bc + bd + be$:
  - Use literals as divisors

$f / a = c + d \rightarrow f = a(c + d) + bc + bd + be = a(c + d) + b(c + d + e)$

$f / b = c + d + e \rightarrow f = b(c + d + e) + ac + ad = b(c + d + e) + a(c + d)$

$f / c = a + b \rightarrow f = c(a + b) + ad + bd + be = c(a + b) + d(a + b) + be$

$f / d = a + b \rightarrow f = d(a + b) + ac + bc + be = d(a + b) + c(a + b) + be$

  - Use kernels as divisors

$f / (c + d) = a + b \rightarrow f = (c + d)(a + b) + be$

$f / (c + d + e) = b \rightarrow f = (c + d + e)\, b + ac + ad$

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

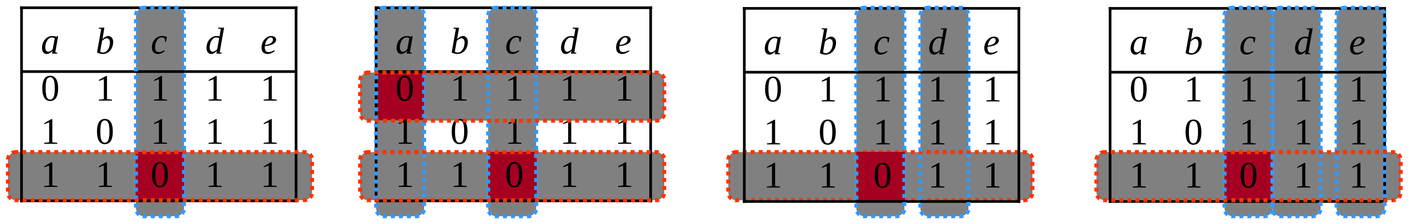
$f / (a + b) = c + d \rightarrow f = (a + b)(c + d) + be$

- In many case, by using kernels as divisors, more compact formulation can be derived with smaller number of steps.

Computing the Kernels (1)

- Algebraic cube division on cube-literal matrix f / c (c is a cube)
  - Disable all columns whose corresponding literals are included in c.
  - Disable all rows having 0 in one of the disabled columns.
  - The remaining sub-matrix, consisting of enabled columns and rows, is the quotient. This sub-matrix can be described by a pair of bit-vectors V_R and V_C indicating the enable/disable status of each row and column, without rewriting the entire matrix.

Ex : $f = abde + acde + bcde$



$f / c = ade + bde$ $f / ac = de$ $f / cd = ae + be$ $f / cde = a + b$

Computing the Kernels (2)

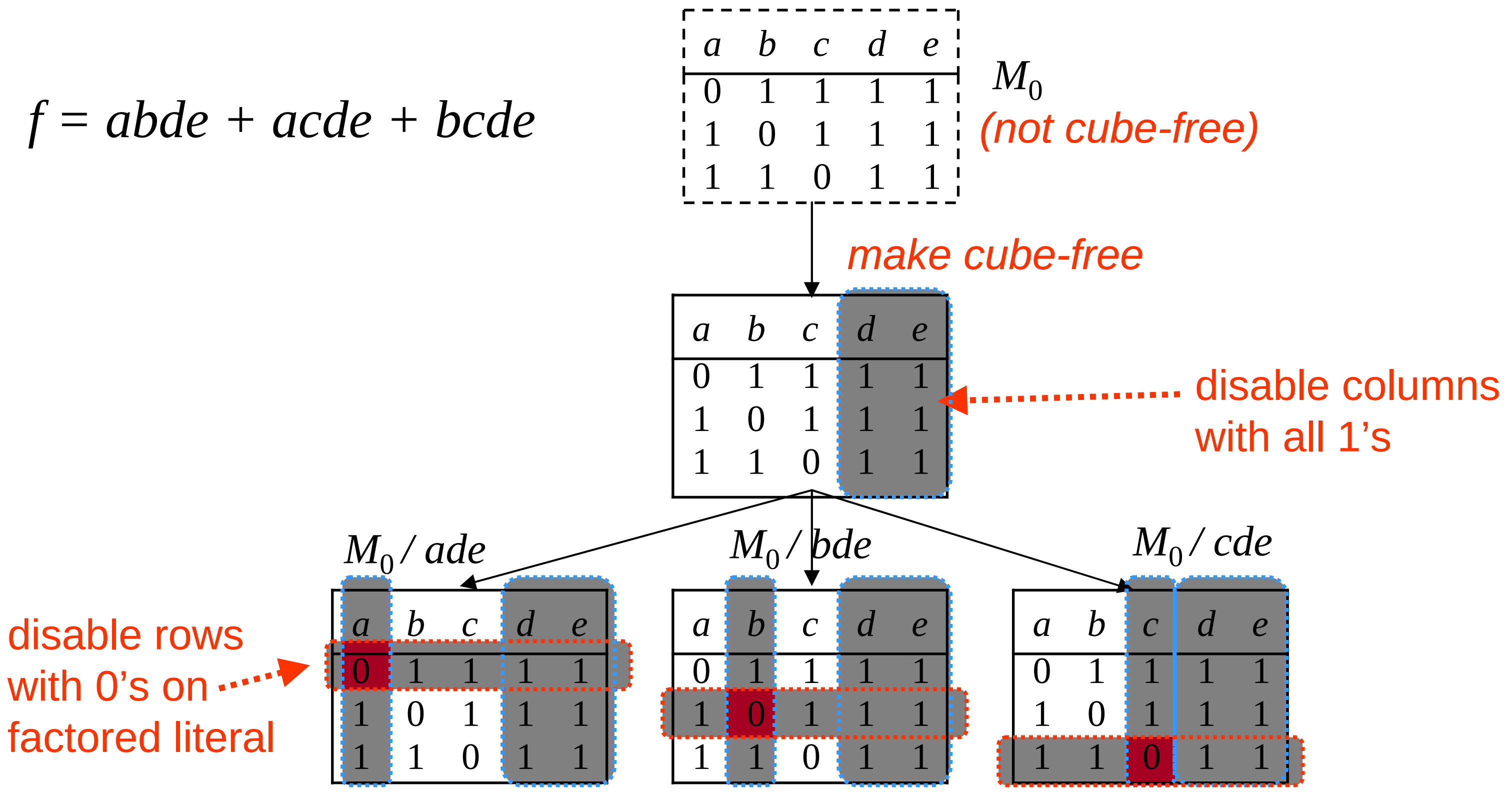
- Basic flow of kernel computation :
  - A) Push the original (nontrivial) function to STACK.
  - B) Pop a function from the STACK, and divide this function by the largest cube factor (divides the function evenly) so that the quotient becomes cube-free. The resulting quotient is a *kernel*.
  - C) For each literal L in the original function, divide the obtained kernel by L. If the resulting quotient is nontrivial, push this quotient to STACK.
    - ➢ Computation pruning strategy : When dividing the obtained kernel by each literal, skip the ones which have already been evaluated prior to being pushed onto STACK. (*Put an attribute of the dividing literal to each function being push onto the STACK*)
  - D) Go to B).

Kernel Computation Algorithm

```
$STACK$ = $\phi$ ; $K$ = $\phi$ ;
Generate the cube-literal matrix $M$ on given function $f$ ;
$L$ = 0 ; $W$ = (# of columns of $M$) ;
PUSH [$M$, $L$] to $STACK$ ;
WHILE ($STACK$ is non-empty) {
        POP [$M$, $L$] from $STACK$;
        IF (column $i$ have 1's on all enabled rows for some $i \leq L$)
                continue ; // skip further computation
        make $M$ cube-free; // disable columns having all 1's
        $K = K + \{M\}$;
        FOR ($j = L + 1$ to $W$) {
                IF ((column $j$  is enabled) &&
                    (# of 1's on enabled rows in column $j$)  > 1) {
                        disable all rows $i$ in $M$ where $M(i, j) = 0$;
                        disable column $j$ in $M$ ;
                        $L = j + 1$;
                        PUSH [$M$, $L$] to $STACK$ ;
                }
        }
}
```

Kernel Computation Example (1)

$f = abde + acde + bcde$



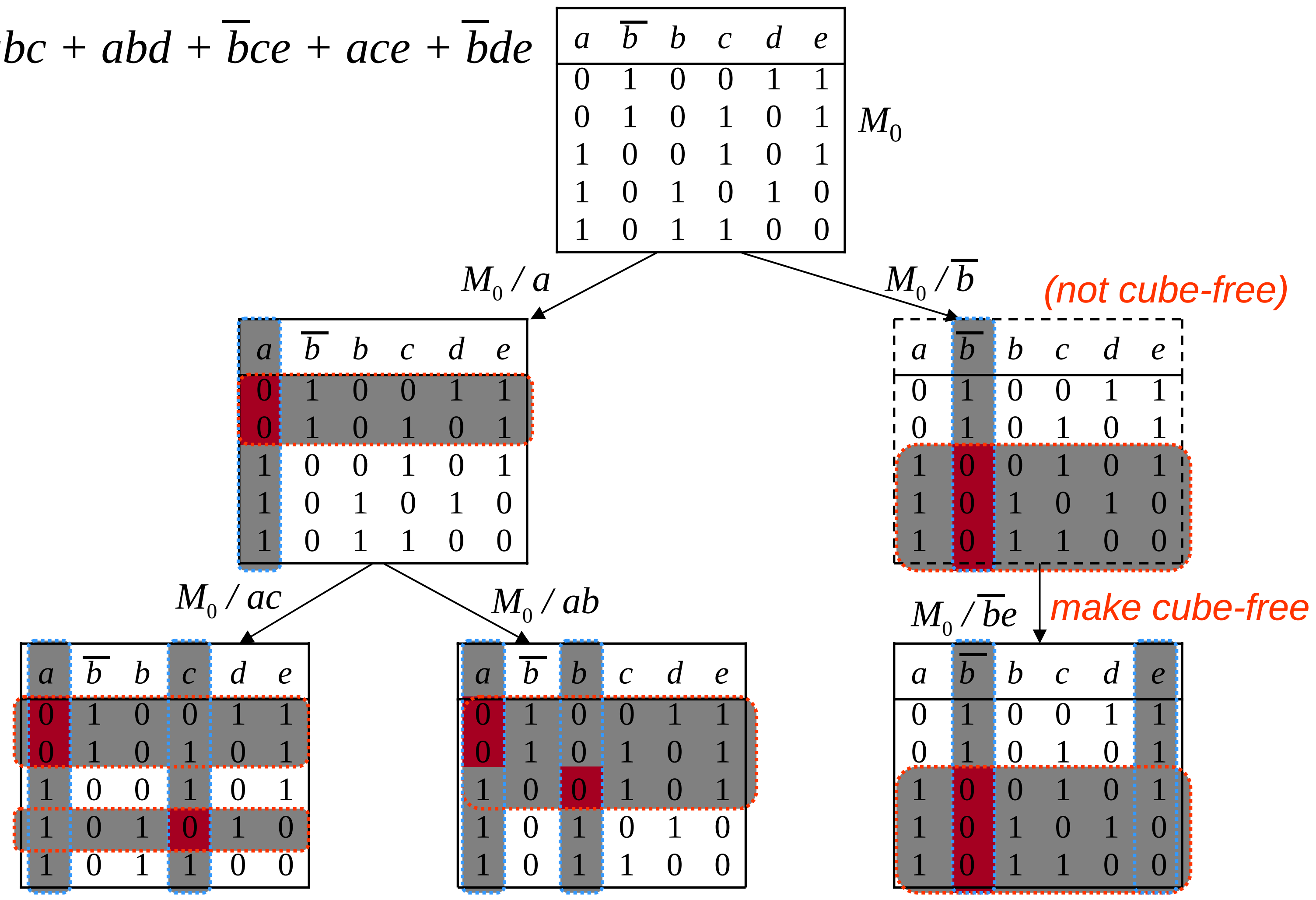
kernel set : $K = \{bc + ac + ab, b + c, a + c, a + b\}$

0-level kernel set : $K_0 = \{b + c, a + c, a + b\}$

0-level cokernel set : $CK_0 = \{ade, bde, cde\}$

Kernel Computation Example (2.a)

$f = abc + abd + \bar{b}ce + ace + \bar{b}de$



M_0

a	$\bar{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / a

a	$\bar{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

$M_0 / \bar{b}$

a	$\bar{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / ac

a	$\bar{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / ab

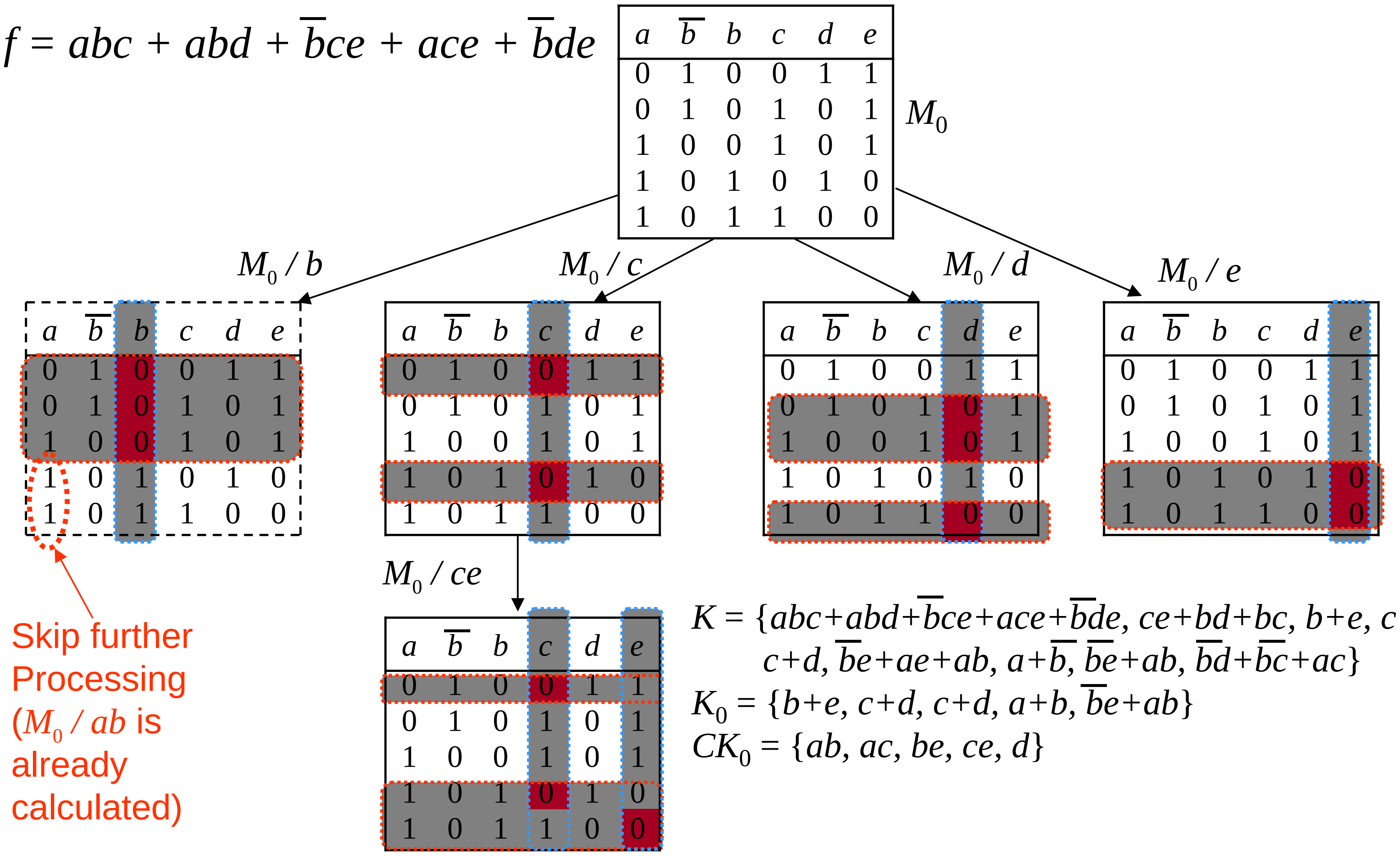
a	$\bar{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

$M_0 / \bar{b}e$

a	$\bar{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

Kernel Computation Example (2.b)

$f = abc + abd + \overline{b}ce + ace + \overline{b}de$



M_0

a	$\overline{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / b

a	$\overline{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / c

a	$\overline{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / d

a	$\overline{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / e

a	$\overline{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

M_0 / ce

a	$\overline{b}$	b	c	d	e
0	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0

Skip further Processing (M_0 / ab is already calculated)

$K = \{abc+abd+\overline{b}ce+ace+\overline{b}de,\ ce+bd+bc,\ b+e,\ c+d,$
$c+d,\ \overline{b}e+ae+ab,\ a+\overline{b},\ \overline{b}e+ab,\ \overline{b}d+\overline{b}c+ac\}$

$K_0 = \{b+e,\ c+d,\ c+d,\ a+b,\ \overline{b}e+ab\}$

$CK_0 = \{ab,\ ac,\ be,\ ce,\ d\}$

Multi-Level Logic Optimization With Algebraic Division

- *Need* efficient implementation of algebraic division computation method (given a divisor)
- *Need* methods for selecting good divisors
  - Select divisors with large # of literals and cubes
    - $\rightarrow$ *cubes*, *kernels*
  - Select divisors which are common among multiple functions.
    - $\rightarrow$ *common cubes, nontrivial kernel intersections*

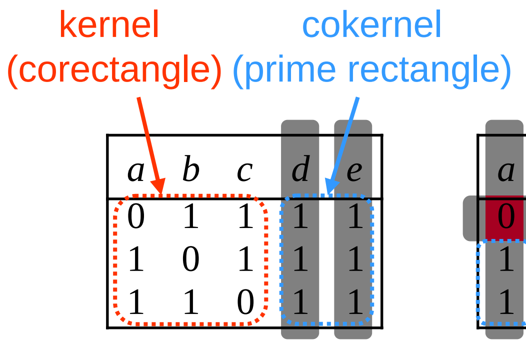
Rectangles in Cube-Literal Matrix

- A rectangle (R, C) in the cube-literal matrix M is a subset of rows R and subset of columns C such that $M(i, j) = 1$ for all $i \in R$ and $j \in \mathrm{C}$.
- A corectangle (R, C') is the same row subset R with the complement column subset of C $(C' = \overline{C})$.
- A containment of rectangles is defined as :

$$(R_0, C_0) \subseteq (R_1, C_1) \textit{ implies } R_0 \subseteq R_1 \textit{ and } C_0 \subseteq C_1.$$

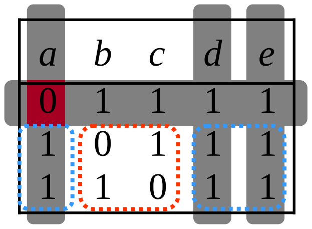
- A prime rectangle is a rectangle not contained by other rectangles. On the cube-literal matrix :
  - Prime rectangle represents a cokernel.
  - Corectangle of a prime rectangle represents a kernel.

Ex : $f = abde + acde + bcde$



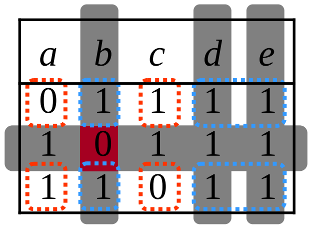
$$\begin{array}{ccccc} a & b & c & d & e \\ \hline 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

$f\,/\,de = bc + ac + ab$



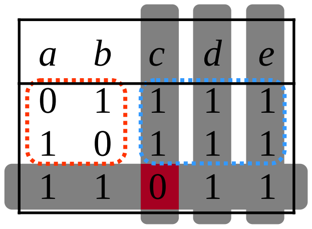
$$\begin{array}{ccccc} a & b & c & d & e \\ \hline 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

$f\,/\,ade = b + c$



$$\begin{array}{ccccc} a & b & c & d & e \\ \hline 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

$f\,/\,bde = a + c$



$$\begin{array}{ccccc} a & b & c & d & e \\ \hline 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

$f\,/\,cde = a + b$

Common Cube Extraction by Rectangle Covering (1)

- Common cube extraction within multiple functions:
  - Common cube is a common divisor among multiple functions in the form of a cube.
  - Common cubes can be extracted by generating the cube-literal matrix for the multiple functions, and extracting rectangles (R, C) on this matrix such that $|R| \geq 2$ and $|C| \geq 2$.
  - *Maximal* common cubes are the cubes which are not contained *(in the algebraic sense)* by other cubes. Such common cubes correspond to prime rectangles, and can be extracted by applying the same algorithm for computing the kernels on this matrix.

Common Cube Extraction by Rectangle Covering (2)

$$F = abc + abdf + eg,\ G = abfg + bcdf,\ H = bdf + be$$

	a	*b*	*c*	*d*	*e*	*f*	*g*
F	**1**	**1**	1	0	0	0	0
F	**1**	**1**	0	1	0	1	0
F	0	0	0	0	1	0	1
G	**1**	**1**	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

	a	*b*	*c*	*d*	*e*	*f*	*g*
F	1	1	1	0	0	0	0
F	**1**	**1**	0	1	0	**1**	0
F	0	0	0	0	1	0	1
G	**1**	**1**	0	0	0	**1**	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

	a	*b*	*c*	*d*	*e*	*f*	*g*
F	1	**1**	**1**	0	0	0	0
F	1	1	0	1	0	1	0
F	0	0	0	0	1	0	1
G	1	1	0	0	0	1	1
G	0	**1**	**1**	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

	a	*b*	*c*	*d*	*e*	*f*	*g*
F	1	1	1	0	0	0	0
F	1	**1**	0	**1**	0	**1**	0
F	0	0	0	0	1	0	1
G	1	1	0	0	0	1	1
G	0	**1**	1	**1**	0	**1**	0
H	0	**1**	0	**1**	0	**1**	0
H	0	1	0	0	1	0	0

Cost Function of Rectangles on Cube-Literal Matrix

- Difference in total # of gates before and after the extraction
  - # of literals = # of 1s in matrix
  - # of gates = # of literals – # of nodes

	a	*b*	*c*	*d*	*e*	*f*	*g*
F	1	1	1	0	0	0	0
F	1	1	0	1	0	1	0
F	0	0	0	0	1	0	1
G	1	1	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

Divide by *ab*

	a	*b*	*c*	*d*	*e*	*f*	*g*	*X*	F_X
X	1	1	0	0	0	0	0	0	0
F_X	0	0	1	0	0	0	0	0	0
F_X	0	0	0	1	0	1	0	0	0
F	0	0	0	0	0	0	0	1	1
F	0	0	0	0	1	0	1	0	0
G	0	0	0	0	0	1	1	1	0
G	0	1	1	1	0	0	0	0	0
H	0	1	0	1	0	1	0	0	0
H	0	1	0	0	1	0	0	0	0

$X = ab$
$F_X = c + df$
$F = XF_X + eg$
$G = Xfg + bcdf$
$H = bdf + be$

$F = abc + abdf + eg$
$G = abfg + bcdf$
$H = bdf + be$

22 literals, 19 gates

21 literals, 16 gates

Cost(*R*, *C*) = 16 – 19 = – 3

Breakdown of Cost Function (1)

- Step 1: extract cube *C*
  - ➢ *Weight*(*R*, *C*) = # of 1s in rectangle (*R*, *C*)
    - ✓ If all elements are 1, then *Weight*(*R*, *C*) = |*R*|•|*C*|
  - ➢ *Overhead*(*R*, *C*) = |*R*| + |*C*|
    - ✓ |*R*| : # of appearances of cube *C*
    - ✓ |*C*| : # of literals in cube *C*
  - ➢ Difference in # of literals = *Overhead*(*R*, *C*) – *Weight*(*R*, *C*)
  - ➢ # of nodes added = 1 (to compute cube *C*).

	a	*b*	*c*	*d*	*e*	*f*	*g*
F	1	1	1	0	0	0	0
F	1	1	0	1	0	1	0
F	0	0	0	0	1	0	1
G	1	1	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

$F = abc + abdf + eg$

$G = abfg + bcdf$

$H = bdf + be$

22 literals, 19 gates

extract *ab*

	a	*b*	*c*	*d*	*e*	*f*	*g*	*X*
X	1	1	0	0	0	0	0	0
F	0	0	1	0	0	0	0	1
F	0	0	0	1	0	1	0	1
F	0	0	0	0	1	0	1	0
G	0	0	0	0	0	1	1	1
G	0	1	1	1	0	1	0	0
H	0	1	0	1	0	1	0	0
H	0	1	0	0	1	0	0	0

$X = ab$

$F = Xc + Xdf + eg$

$G = Xfg + bcdf$

$H = bdf + be$

21 literals, 17 gates

Breakdown of Cost Function (2)

- Step 2 : divide by cube *C*
  - Nodes containing only a single instance of *C* are already divided by *C* *(no change)*
  - If a node *f* contains 2 or more instances of *C*, then add a new node $f_C = f / C$ (algebraic quotient of *f* divided by *C*)
    - ✓ Difference in # of literals : $Merge(R, C) = 2 \times |Q'| - |R'|$
      - Q' = set of nodes containing multiple instances of cube *C*
      - $|R'|$ = (# of appearances of cube *C* in Q')
    - ✓ # of nodes added : $|Q'|$ (# of new quotients f/C)

	a	*b*	*c*	*d*	*e*	*f*	*g*	*X*
X	1	1	0	0	0	0	0	0
F	0	0	1	0	0	0	0	1
F	0	0	0	1	0	1	0	1
F	0	0	0	0	1	0	1	0
G	0	0	0	0	0	1	1	1
G	0	1	1	1	0	1	0	0
H	0	1	0	1	0	1	0	0
H	0	1	0	0	1	0	0	0

21 literals, 17 gates

$X = ab$

$F = Xc + Xdf + eg$

$G = Xfg + bcdf$

$H = bdf + be$

Divide by *X*

Q'

$(|R'| = 2)$

	a	*b*	*c*	*d*	*e*	*f*	*g*	*X*	F_X
X	1	1	0	0	0	0	0	0	0
F_X	0	0	1	0	0	0	0	0	0
F_X	0	0	0	1	0	1	0	0	0
F	0	0	0	0	0	0	0	1	1
F	0	0	0	0	1	0	1	0	0
G	0	0	0	0	0	1	1	1	0
G	0	1	1	1	0	1	0	0	0
H	0	1	0	1	0	1	0	0	0
H	0	1	0	0	1	0	0	0	0

21 literals, 16 gates

$X = ab$

$F_X = c + df$

$F = XF_X + eg$

$G = Xfg + bcdf$

$H = bdf + be$

Breakdown of Cost Function (3)

- Changes in # of literals when divided by cube C
  - $\Delta Literal(R, C) = Overhead(R, C) – Weight(R, C) + Merge(R, C)$
    $= (|R| + |C|) – (\#\text{ 1s in } (R, C)) + (2 \times |Q'| – |R'|)$
- Total # of nodes added : $\Delta Node(R, C) = |Q'| + 1$
- Changes in # of gates when divided by cube C
  - $\Delta Gate(R, C) = \Delta Literal(R, C) – \Delta Node(R, C)$
    $= (|R| + |C|) – (\#\text{ 1s in } (R, C)) + (|Q'| – |R'|) – 1$
    $= |C| – (\#\text{ 1s in } (R, C)) + |Q| – 1$
    where Q = set of nodes containing one or more instances of cube C
    $(|Q'| – |R'| = |Q| – |R|)$
  - Ex.
    $\Delta Gate(\{2, 5, 6\}, \{2, 4, 6\}) = 3 + 3 – 9 – 1 = – 4$
    $\Delta Gate(\{1, 2, 4\}, \{1, 2\}) = 2 + 2 – 6 – 1 = – 3$

$Q_1 = Q - Q'$: set of nodes containing one instance of cube C
$|R| - |R'|$: # of appearances of cube C in Q_1
$\rightarrow |R| - |R'| = |Q_1| = |Q| - |Q'|$
$\rightarrow |Q'| - |R'| = |Q| - |R|$

	a	b	c	d	e	f	g
F	1	1	1	0	0	0	0
F	1	1	0	1	0	1	0
F	0	0	0	0	1	0	1
G	1	1	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

Overlapping of Rectangle Covers (1)

- The cube *abdf* included in *F* can be covered by cube *ab* and cube *bdf* :

  $abdf = ab \cdot bdf$

  (unfortunately, this cannot be derived from algebraic division)

  → *"don't-care" cubes*
- Don't-care cubes can be eliminated (does not contribute to literal count or gate count costs), but also can be used in algebraic division.

$$F = abc + abdf + eg$$

$$\Downarrow$$

$$X = ab$$
$$F_X = c + df$$
$$F = XF_X + eg$$

CANNOT factor out *bdf*

$$\Nearrow$$

$$X = ab$$
$$F_X = abc + abdf$$
$$F = XF_X + eg$$

Don't-care cubes

$$\Downarrow$$

CAN factor out *bdf*

$$X = ab$$
$$Y = bdf$$
$$F_X = abc + aY$$
$$F = XF_X + eg$$

$$\Rightarrow$$

$$X = ab$$
$$Y = bdf$$
$$F_X = c + Y$$
$$F = XF_X + eg$$

Overlapping of Rectangle Covers (1)

- Overlapping of multiple rectangle covers
  - Instead of changing the 1s to 0s in the rectangle, change to "don't-care" (marked ' * ' below).
  - Allow rectangles to cover "don't-cares" as well as 1s.

	a	b	c	d	e	f	g
F_X	*	*	1	0	0	0	0
F_X	*	*	0	1	0	1	0
F	0	0	0	0	1	0	1
G	*	*	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

$$X = ab$$
$$F_X = c + df$$
$$F = XF_X + eg$$
$$G = X fg + bcdf$$
$$H = bdf + be$$

$$\Longrightarrow$$

$$X = ab$$
$$Y = bdf$$
$$F_X = c + Y$$
$$F = XF_X + eg$$
$$G = X fg + Yc$$
$$H = Y + be$$

# of 1s in (R, C) = 8

$\Delta Gate(R, C)$

= $|C|$ + (# nodes containing C) – (# 1s in (R, C)) – 1

= 3 + 3 – 8 – 1 = – 3

Overlapping of Rectangle Covers (2)

	a	b	c	d	e	f	g
F	1	1	1	0	0	0	0
F	1	1	0	1	0	1	0
F	0	0	0	0	1	0	1
G	1	1	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

$\Delta G = -3$

	a	b	c	d	e	f	g
F_X	*	*	1	0	0	0	0
F_X	*	*	0	1	0	1	0
F	0	0	0	0	1	0	1
G	*	*	0	0	0	1	1
G	0	1	1	1	0	1	0
H	0	1	0	1	0	1	0
H	0	1	0	0	1	0	0

$\Delta G = -3$

	a	b	c	d	e	f	g
F_X	*	*	1	0	0	0	0
F_X	*	*	0	*	0	*	0
F	0	0	0	0	1	0	1
G	*	*	0	0	0	1	1
G	0	*	1	*	0	*	0
H	0	*	0	*	0	*	0
H	0	1	0	0	1	0	0

$F = abc + abdf + eg$
$G = abfg + bcdf$
$H = bdf + be$

$X = ab$
$F_X = c + df$
$F = XF_X + eg$
$G = X\,fg + bcdf$
$H = bdf + be$

$X = ab$
$Y = bdf$
$F_X = c + Y$
$F = XF_X + eg$
$G = X\,fg + Yc$
$H = Y + be$

Multi-Level Logic Optimization With Algebraic Division

- *Need* efficient implementation of algebraic division computation method (given a divisor)
- *Need* methods for selecting good divisors
  - Select divisors with large # of literals and cubes
    - $\rightarrow$ *cubes*, *kernels*
  - Select divisors which are common among multiple functions.
    - $\rightarrow$ *common cubes, nontrivial kernel intersections*

Kernel Intersection for Optimizing Multiple Functions (1)

- A *nontrivial* function is a function with two or more cubes which cannot be *reduced* to a single cube.
  - $f = abc + abd$ is <u>*nontrivial*</u>
  - $g = abc + ab$ is <u>*trivial*</u> because it reduces to a single cube ab.
  - A cube-free function is nontrivial (therefore kernels are nontrivial)
- Theorem 4 : On functions f and g, the two functions have a *nontrivial* common divisor if and only if there exist kernels $k_f \in K(f)$ and $k_g \in K(g)$ such that $k_{fg} = k_f \cap k_g$ is *nontrivial* (called nontrivial kernel intersection)
- Theorem 4 (*necessity*) : If f and g have a nontrivial kernel intersection, then they have a nontrivial common divisor.
  - ➢ <u>PROOF hints</u> *: Simply show that a nontrivial kernel intersection k_{fg} is the divisor for both f and g .*

Kernel Intersection for Optimizing Multiple Functions (2)

- Theorem 4 (*sufficiency*) : If f and g have a nontrivial common divisor, then they have a nontrivial kernel intersection.
  - <u>PROOF sketches</u>:
    - ✓ Let d be the nontrivial common divisor for f and g. Make d cube-free and call this e (If d is cube-free, then $e = d$. Otherwise, e is the kernel of d).
    - ✓ e is the common nontrivial (cube-free) divisor for f and g.
    - ✓ By Theorem 3, there exist $k_f \in P(f)$, $k_g \in P(g)$ such that $e \subseteq k_f$, $e \subseteq k_g$.
    - ✓ Since e is cube-free, such k_f and k_g are also cube-free. Therefore both k_f and k_g are kernels for f and g, respectively ($k_f \in K(f)$, $k_g \in K(g)$).
    - ✓ Since $e \subseteq k_f \cap k_g$, and e is nontrivial (since it is cube-free), $k_f \cap k_g$ is nontrivial.

Kernel Intersection for Optimizing Multiple Functions (3)

- Implications of Theorem 4 :
  - ➢ Nontrivial kernel intersections are nontrivial common divisors.

  → *Good divisor candidate*
  - ➢ If there are no nontrivial kernel intersections, this indicates that there are no common divisors.

  → *Search for other types of divisors (common cubes, kernels)*

Kernel Intersection Extraction by Rectangle Covering (1)

- Cokernel-cube matrix：
  - Assign a distinct index to each cube in each node (starting from 1)
  - Assign all cokernels to rows
  - Assign all cubes contained in each kernel (kernel-cube) to columns.
  - At each matrix element, assign the index of the cube which is formed by the product of the kernel-cube (row) and its corresponding cokernel (column). If there is no such pair, assign 0.

$$F = \overset{1}{abd}+\overset{2}{acd}+\overset{3}{bc}+\overset{4}{bf}+\overset{5}{cf}$$

$$G = \overset{6}{adf}+\overset{7}{bf}+\overset{8}{cf}+\overset{9}{ef}$$

$$H = \overset{10}{ade}+\overset{11}{bc}+\overset{12}{ce}$$

	cokernel	kernel
F	ad	$b+c$
F	b	$ad+c+f$
F	c	$ad+b+f$
F	f	$b+c$
G	f	$ad+b+c+e$
H	c	$b+e$
H	e	$ad+c$

	ad	b	c	d	e	f
$F{:}ad$	0	1	2	0	0	0
$F{:}b$	1	0	3	0	0	4
$F{:}c$	2	3	0	0	0	5
$F{:}f$	0	4	5	0	0	0
$G{:}f$	6	7	8	0	9	0
$H{:}c$	0	11	0	0	12	0
$H{:}e$	10	0	12	0	0	0

Kernel Intersection Extraction by Rectangle Covering (2)

- A rectangle (R, C) in the cokernel-cube matrix M is a subset of rows R and subset of columns C such that $M(i, j) \neq 0$ for all $i \in R$ and $j \in$ C.
- A nontrivial kernel intersection is a rectangle (R, C) in cokernel-cube matrix such that $|R| \geq 2$ and $|C| \geq 2$.
- Also allow rectangle cover overlap at "don't-care" cubes.

	ad	*b*	*c*	*d*	*e*	*f*
F:ad	0	1	2	0	0	0
F:b	1	0	3	0	0	4
F:c	2	3	0	0	0	5
F:f	0	4	5	0	0	0
G:f	6	7	8	0	9	0
H:c	0	11	0	0	12	0
H:e	10	0	12	0	0	0

$$F = \overset{1}{abd}+\overset{2}{acd}+\overset{3}{bc}+\overset{4}{bf}+\overset{5}{cf}$$

$$G = \overset{6}{adf}+\overset{7}{bf}+\overset{8}{cf}+\overset{9}{ef}$$

$$H = \overset{10}{ade}+\overset{11}{bc}+\overset{12}{ce}$$

When changing rectangle elements to don't-care, other elements with the same index needs to be changed

	ad	*b*	*c*	*d*	*e*	*f*
F:ad	0	*	*	0	0	0
F:b	*	0	3	0	0	*
F:c	*	3	0	0	0	*
F:f	0	*	*	0	0	0
G:f	6	*	*	0	9	0
H:c	0	11	0	0	12	0
H:e	10	0	12	0	0	0

$$X = b+c$$

$$F_X = ad+f$$

$$F = XF_X\underline{+abd+acd}+bc\underline{+bf+cf}$$

$$G = Xf+adf\underline{+bf+cf}+ef$$

$$H = ade+bc+ce$$

don't-care cubes

Kernel Intersection Extraction by Rectangle Covering (3)

	ad	b	c	d	e	f
F:ad	0	1	2	0	0	0
F:b	1	0	3	0	0	4
F:c	2	3	0	0	0	5
F:f	0	4	5	0	0	0
G:f	6	7	8	0	9	0
H:c	0	11	0	0	12	0
H:e	10	0	12	0	0	0

$X = b+c$

$F_X = ad+f$

$F = XF_X+abd+acd+bc+bf+cf$

$G = Xf+adf+bf+cf+ef$

$H = ade+bc+ce$

	ad	b	c	d	e	f
F:ad	0	*	*	0	0	0
F:b	*	0	3	0	0	*
F:c	*	3	0	0	0	*
F:f	0	*	*	0	0	0
G:f	6	*	*	0	9	0
H:c	0	11	0	0	12	0
H:e	10	0	12	0	0	0

$X = b+c$

$Y = ad + c$

$F_X = ad+f$

$F = XF_X+Yb+abd+acd+bc+bf+cf$

$G = Xf+Yf+adf+bf+cf+ef$

$H = Ye+ade+bc+ce$

Cost Function of Rectangles on Cokernel-Cube Matrix

- $w_r(i)$: # literals in cokernel in row i.
- $w_c(j)$: # literals in kernel-cube in row j.
- $b(i, j)$: Boolean flag on element (i, j)
  - 0 (if $M(i, j) = 0$ || $M(i, j) =$ ‘*’)
  - 1 (otherwise)
- $Weight(R, C) = \Sigma_{i\in R,\, j\in C}\, b(i, j)\,(w_r(i) + w_c(j))$
- $Overhead(R, C) = \Sigma_{i\in R}\, w_r(i) + \Sigma_{j\in C}\, w_c(j)$
- $Merge(R, C) = 2 \times$ (# nodes containing cube C)
- $\Delta Literal(R, C) = Overhead(R, C) + Merge(R, C) - Weight(R, C)$
- $\Delta Node(R, C) =$ (# nodes containing cube C) + 1
- $\Delta Gate(R, C) = \Delta Literal(R, C) - \Delta Node(R, C)$

  $= -\,\Sigma_{i\in R,\, j\in C}\, b(i, j)\,(w_r(i) + w_c(j))$

  $+\, \Sigma_{i\in R}\, w_r(i) + \Sigma_{j\in C}\, w_c(j)$ + (# nodes containing cube C) – 1

$X = b+c$

$F_X = ad+f$

$F = XF_X+abd+acd+bc+bf+cf$

$G = Xf+adf+bf+cf+ef$

$H = ade+bc+ce$

Kernel Intersection Extraction by Rectangle Covering

$F = \overset{1}{abd}+\overset{2}{acd}+\overset{3}{bc}+\overset{4}{bf}+\overset{5}{cf}$

$G = \overset{6}{adf}+\overset{7}{bf}+\overset{8}{cf}+\overset{9}{ef}$

$H = \overset{10}{ade}+\overset{11}{bc}+\overset{12}{ce}$

	ad	*b*	*c*	*d*	*e*	*f*
F:ad	0	1	2	0	0	0
F:b	1	0	3	0	0	4
F:c	2	3	0	0	0	5
F:f	0	4	5	0	0	0
G:f	6	7	8	0	9	0
H:c	0	11	0	0	12	0
H:e	10	0	12	0	0	0

Weight(*R*, *C*) = 14
Overhead(*R*, *C*) = 6
Merge(*R*, *C*) = 4
Δ*Literal*(*R*, *C*) = 4 + 6 – 14 = –4
Δ*Node*(*R*, *C*) = 3
Δ*Gate*(*R*, *C*) = – 4 – 3 = – 7

$X = b+c$
$F_X = ad+f$
$F = XF_X+bc$
$G = Xf+adf+ef$
$H = ade+bc+ce$

	ad	*b*	*c*	*d*	*e*	*f*
F:ad	0	*	*	0	0	0
F:b	*	0	3	0	0	*
F:c	*	3	0	0	0	*
F:f	0	*	*	0	0	0
G:f	6	*	*	0	9	0
H:c	0	11	0	0	12	0
H:e	10	0	12	0	0	0

Weight(*R*, *C*) = 10
Overhead(*R*, *C*) = 6
Merge(*R*, *C*) = 6
Δ*Literal*(*R*, *C*) = 6 + 6 – 10 = 2
Δ*Node*(*R*, *C*) = 4
Δ*Gate*(*R*, *C*) = 2 – 4 = – 2

$X = b+c$
$Y = ad+c$
$F_X = ad+f$
$F = XF_X+Yb$
$G = Xf+Yf+ef$
$H = Ye+bc$

	ad	*b*	*c*	*d*	*e*	*f*
F:ad	0	*	*	0	0	0
F:b	*	0	*	0	0	*
F:c	*	*	0	0	0	*
F:f	0	*	*	0	0	0
G:f	*	*	*	0	9	0
H:c	0	11	0	0	*	0
H:e	*	0	*	0	0	0

Multi-Level Logic Optimization Flow

Optimization schemes on a set of functions $\{f_i\}$

A) Nontrivial common divisor extraction :

- generate the kernel sets for each f_i.
- Select a pair of kernels $k_i \in K(f_i)$ and $k_j \in K(f_j)$ where $i \neq j$ such that $k_i \cap k_j$ is nontrivial. If such kernel intersection exists, use $k_i \cap k_j$ as divisors to divide all functions divisible by $k_i \cap k_j$.

B) Common cube extraction :

- Select a pair of cubes $c_i \in f_i$ and $c_j \in f_j$ where $i \neq j$ such that $c_i \cap c_j$ has 2 or more literals. If such common cube exists, use $c_i \cap c_j$ as divisors to divide all functions divisible by $c_i \cap c_j$.

✓ NOTE : intersections on cubes assumes cubes as a set of literals

C) Division decomposition on each function f_i :

- Select a kernel $k_i \in K(f_i)$ and divide f_i by k_i.