EE6094
CAD for VLSI Design

Chapter 5
Logic Optimization

Spring 2024
Andy, Yu-Guang Chen
Assistant Professor, Department of EE
National Central University
andyygchen@ee.ncu.edu.tw

Outline

- Logic optimization overview
- Two-level logic optimization
- Multi-level logic optimization

Outline

- Logic optimization overview
- Two-level logic optimization
- Multi-level logic optimization

HDL Synthesis

Synthesis = Domain Translation + Optimization + Mapping

Domain translation

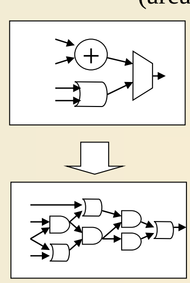
Optimization (area, timing, power...)

```
--VHDL
if(A='1') then
  Y<=C + D;
elseif (B='1') then
  Y<=C or D;
else Y<=C;
endif
```

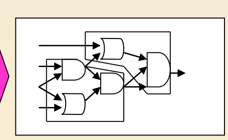
```
//Verilog
if(A==1)
   Y=C + D;
else if(B==1)
   Y=C | D;
else Y=C;
```

Behavioral domain

RTL synthesis



Structural domain



Two-Level Logic Optimization

- Two-level logic representations
  - Sum-of-product form
  - Product-of-sum form
- Two-level logic optimization
  - Key technique in logic optimization
  - Many efficient algorithms to find a near minimal representation in a practical amount of time
  - In commercial use for several years
  - Minimization criteria: **number of product terms**
- Example: F = XYZ + XY'Z' + XY'Z + X'YZ+XYY'Z
  = XY' + YZ

Multi-Level Logic Optimization

- Translate a combinational circuit to meet performance or area constraints
  - Two-level minimization
  - Common factors or kernel extraction
  - Common expression resubsitution
- In commercial use for several years
- Example:

$$\begin{aligned} f1 &= bcd + \bar{b}c\bar{d} + c\bar{d}e + \\ &\quad \bar{a}c + cdf + ab\bar{c}\bar{d}\bar{e} + a\bar{b}\bar{c}d\bar{f} \\ f2 &= bdg + \bar{b}dfg + \bar{b}\bar{d}g + b\bar{d}eg \end{aligned} \Rightarrow \begin{aligned} f1 &= c(\bar{a} + x) + a\bar{c}\bar{x} \\ f2 &= gx \\ x &= d(b + f) + \bar{d}(\bar{b} + e) \end{aligned}$$

Technology Mapping

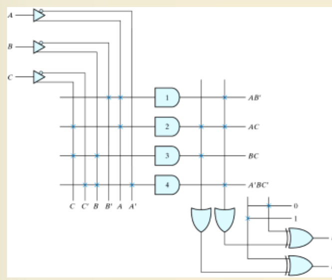
- Goal: translation of a technology independent representation (e.g. Boolean networks) of a circuit into a circuit in a given technology (e.g. standard cells) with optimal cost
- Optimization criteria:
  - Minimum area
  - Minimum delay
  - Meeting specified timing constraints
  - Meeting specified timing constraints with minimum area
- Usage:
  - Technology mapping after technology independent logic optimization
  - Technology translation

Outline

- Logic optimization overview
- Two-level logic optimization
- Multi-level logic optimization

Two-Level Logic Optimization

- **Goals**
  - Primary goal is to reduce the number of product terms
    - All product terms have the same cost
    - Implicants correspond to PLA rows
  - Secondary goal is to reduce the number of literals
    - Literals correspond to transistors

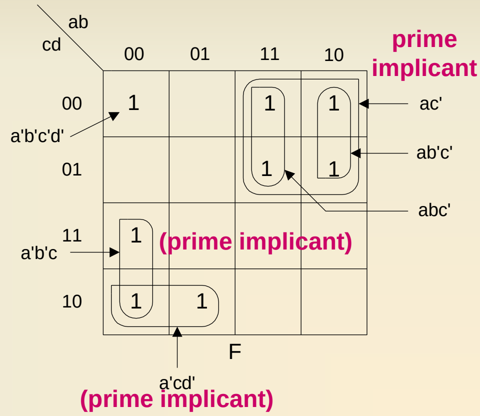


Two-Level Logic Optimization

- **Basic idea**: Boolean law $x+x'=1$ allows for grouping $x_1x_2+x_1\,x'_2= x_1$
- Approaches to simplify logic functions:
  - Karnaugh maps [Kar53]
  - Quine-McCluskey [McC56]

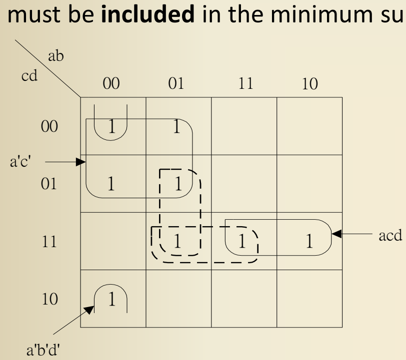
Implicant

- **Implicant**:
  any single 1 or any group of 1’s combined together on a map of the function F
  - ab’c’, abc’
- **Prime Implicant**:
  an implicant that cannot be combined with another terms to eliminate a variable
  - a’b’c, a’cd’, ac’



Essential Prime Implicant

- If a minterm is covered by **only one** prime implicant, that prime implicant is **ESSENTIAL**
  - must be **included** in the minimum sum-of-products



Note: 1's in red color are covered by only one prime implicant. All other 1's are covered by at least two prime implicants

Minimum Sum-of-Products

- Minimum number of prime implicants which cover all of the 1's
  - Minimum cover (global optimum)
- A sum-of-products expression containing a non-prime implicant cannot be minimum
  - Could be simplified by combining the non-prime term with additional minterm
- To find the minimum sum-of-products
  - Not every prime implicant is needed
  - If prime implicants are selected in the wrong order, a non-minimum solution may result
  - Essential prime implicants must be included

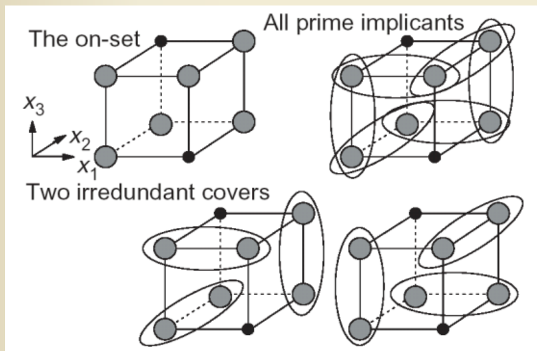
Minimal Cover or Irredundant cover

- A set of prime implicants that together cover all points in the on-set (and some or all points of the dc-set) is called a prime cover
- A prime cover is irredundant when none of its prime implicants can be removed from the cover
  - Minimal cover (local optimum)

- Different from minimum cover (possibly same)

Cover Examples

- $f = x_1' x_3' + x_2' x_3 + x_1 x_2$
- $f = x_1' x_2' + x_2 x_3' + x_1 x_3$



Quine, McCluskey

- Two-step process
  - 1. Generation of all prime implicants
  - 2. Extraction of a minimum cover (covering problem)

Prime Implicant Generation

- Utilize AB+AB'=A(B+B')=A
- F = ∑m (4,5,6,8,9,10,13)+d(0,7,15)

Primary Implicant Generation

- F = ∑m (4,5,6,8,9,10,13)+d(0,7,15)

Implication Table

	Column I	
zero "1"	0000	
one "1"	0100 1000	
two "1"	0101 0110 1001 1010	
three "1"	0111 1101	
four "1"	1111	

Primary Implicant Generation

<table>
<tr><th colspan="3">Implication Table</th></tr>
<tr><th>Column I</th><th>Column II</th><th></th></tr>
<tr><td>0000 |</td><td>0-00</td><td></td></tr>
<tr><td></td><td>-000</td><td></td></tr>
<tr><td>0100 |</td><td></td><td></td></tr>
<tr><td>1000 |</td><td>010-</td><td></td></tr>
<tr><td></td><td>01-0</td><td></td></tr>
<tr><td>0101 |</td><td>100-</td><td></td></tr>
<tr><td>0110 |</td><td>10-0</td><td></td></tr>
<tr><td>1001 |</td><td></td><td></td></tr>
<tr><td>1010 |</td><td>01-1</td><td></td></tr>
<tr><td></td><td>-101</td><td></td></tr>
<tr><td>0111 |</td><td>011-</td><td></td></tr>
<tr><td>1101 |</td><td>1-01</td><td></td></tr>
<tr><td>1111 |</td><td>-111</td><td></td></tr>
<tr><td></td><td>11-1</td><td></td></tr>
</table>

Primary Implicant Generation

<table>
<tr><th colspan="3">Implication Table</th></tr>
<tr><th>Column I</th><th>Column II</th><th>Column III</th></tr>
<tr><td>0000 |</td><td>0-00 *</td><td>01-- *</td></tr>
<tr><td></td><td>-000 *</td><td></td></tr>
<tr><td>0100 |</td><td></td><td>-1-1 *</td></tr>
<tr><td>1000 |</td><td>010- |</td><td></td></tr>
<tr><td></td><td>01-0 |</td><td></td></tr>
<tr><td>0101 |</td><td>100- *</td><td></td></tr>
<tr><td>0110 |</td><td>10-0 *</td><td></td></tr>
<tr><td>1001 |</td><td></td><td></td></tr>
<tr><td>1010 |</td><td>01-1 |</td><td></td></tr>
<tr><td></td><td>-101 |</td><td></td></tr>
<tr><td>0111 |</td><td>011- |</td><td></td></tr>
<tr><td>1101 |</td><td>1-01 *</td><td></td></tr>
<tr><td>1111 |</td><td>-111 |</td><td></td></tr>
<tr><td></td><td>11-1 |</td><td></td></tr>
</table>

Primary Implicant Generation

F = ∑m (4,5,6,8,9,10,13)+d(0,7,15)

cd \ ab	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

a, b, c, d

Prime Implicants:
- 0-00 = a'c'd'
- 100- = ab'c'
- 1-01 = ac'd
- -1-1 = bd
- -000 = b'c'd'
- 10-0 = ab'd'
- 01-- = a'b

Column Covering

	4	5	6	8	9	10	13
0,4 (0-00)	×						
0,8 (-000)				×			
8,9 (100-)				×	×		
8,10 (10-0)				×		×	
9,13 (1-01)					×		×
4,5,6,7 (01- -)	×	×	×				
5,7,13,15 (-1-1)		×					×

rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element
is covered by the prime implicant

Column Covering

	4	5	6	8	9	10	13
0,4 (0-00)	X						
0,8 (-000)				X			
8,9 (100-)				X	X		
8,10 (10-0)				X		X	
9,13 (1-01)					X		X
4,5,6,7 (01- -)	X	X	X				
5,7,13,15 (-1-1)		X					X

If column has a single X, then the implicant associated with the row is essential. It must appear in minimum cover

Column Covering

	4	5	6	8	9	10	13
0,4 (0-00)	X						
0,8 (-000)				X			
8,9 (100-)				X	X		
8,10 (10-0)				X		X	
9,13 (1-01)					X		X
4,5,6,7 (01- -)	X	X	X				
5,7,13,15 (-1-1)		X					X

Eliminate all columns covered by essential primes

Column Covering

	4	5	6	8	9	10	13
0,4 (0-00)	×						
0,8 (-000)				×			
8,9 (100-)				×	×		
$\boxed{\text{8,10 (10-0)}}$				×		×	
$\boxed{\text{9,13 (1-01)}}$					×		×
$\boxed{\text{4,5,6,7 (01- -)}}$	×	×	×				
5,7,13,15 (-1-1)		×					×

Find minimum set of rows that cover the remaining columns

f = ab'd' + ac'd + a'b

Petrick's Method

- Solve the satisfiability problem of the following function
  P = (P1+P6)(P6+P7)P6(P2+P3+P4)(P3+P5)P4(P5+P7)=1

	4	5	6	8	9	10	13
0,4 (0-00)	×						
0,8 (-000)				×			
8,9 (100-)				×	×		
8,10 (10-0)				×		×	
9,13 (1-01)					×		×
4,5,6,7 (01- -)	×	×	×				
5,7,13,15 (-1-1)		×					×

- Each term represents a corresponding column
- Each column must be chosen at least once
- All columns must be covered

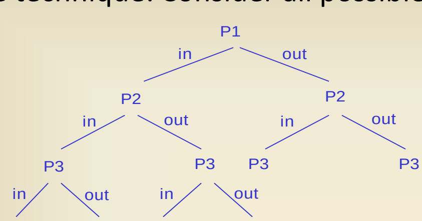
ROBDDs and Satisfiability

- A Boolean function is **satisfiable** if an assignment to its variables exists for which the function becomes '1'
- Any Boolean function whose ROBDD is unequal to '0' is satisfiable.
- Suppose that choosing a Boolean variable x_i to be '1' costs c_i. Then, the **minimum-cost satisfiability** problem asks to minimize: $\sum_{i=1}^{n} c_i \mu(x_i)$

  where $\mu(x_i) = 1$ when x_i = '1' and $\mu(x_i) = 0$ when x_i = '0'.
- Solving minimum-cost satisfiability amounts to computing the shortest path in an ROBDD, which can be solved in linear time.
  - Weights: *$w(v, \eta(v)) = c_i$, $w(v, \lambda(v)) = 0$, variable $x_i = \phi(v)$.*

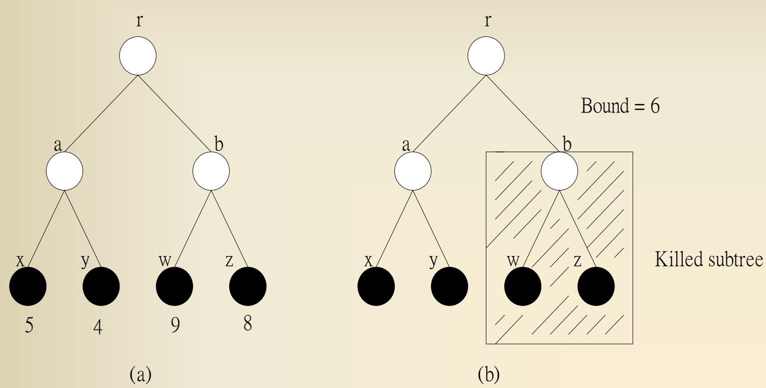
Brute Force Technique

- Brute force technique: Consider all possible elements



- Complete branching tree has $2^{|P|}$ leaves!!
  - Need to prune it
- Complexity reduction
  - Essential primes can be included right away
    - If there is a row with a single "1" for the column
  - Keep track of best solution seen so far
    - Classic ***branch and bound***

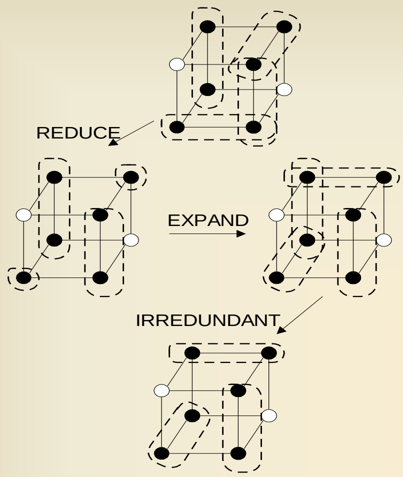
Branch and Bound Algorithm



Heuristic Optimization

- Generation of ***all*** prime implicants is impractical
  - The number of prime implicants for functions with *n* variables is in the order of $3^n/n$
- Finding an *exact* minimum cover is NP-hard
  - Cannot be finished in polynomial time
- Heuristic method: avoid generation of all prime implicants
- Procedure
  - A minterm of ON(f) is selected, and expanded until it becomes a prime implicant
  - The prime implicant is put in the final cover, and all minterms covered by this prime implicant are removed
  - Iterated until all minterms of the ON(f) are covered
- "ESPRESSO" developed by UC Berkeley
  - The kernel of synthesis tools, and computes minimal cover

ESPRESSO

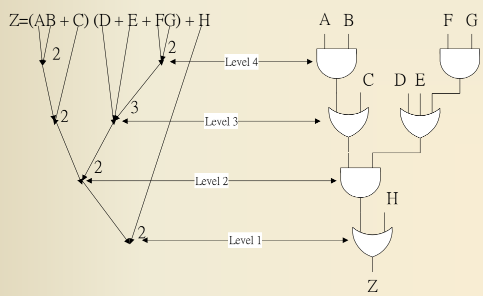


Outline

- Logic optimization overview
- Two-level logic optimization
- Multi-level logic optimization

Multi-Level Logic

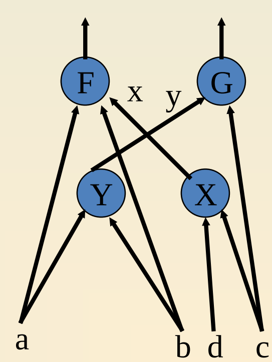
- Multi-level logic:
  - A set of logic equations with no cyclic dependencies
- Example: Z = (AB + C)(D + E + FG) + H
  - 4-level, 6 gates, 13 gate inputs



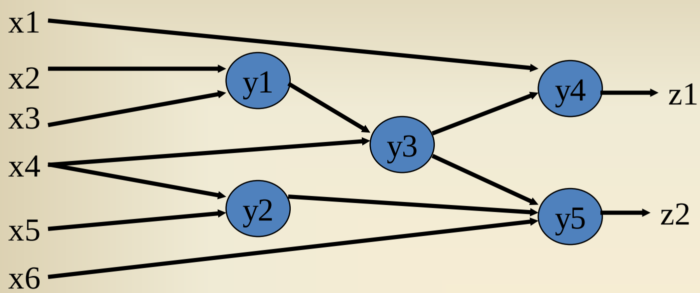
Boolean Network

- Directed acyclic graph (DAG)
- Each source node is a primary input
- Each sink node is a primary output
- Each internal node represents an equation
- Arcs represent variable dependencies

fanin of y : a, b
fanout of x : F



Boolean Network : An Example



y1 = f_1(x2, x3) = x2' + x3'

y2 = f_2(x4, x5) = x4' + x5'

y3 = f_3(x4, y1) = x4'y1'

y4 = f_4(x1, y3) = x1 + y3'

y5 = f_5(x6, y2, y3) = x6y2 + x6'y3'

Multi-Level v.s. Two-Level

Two-level:

- Often used in control logic design

  $f_1 = x_1x_2 + x_1x_3 + x_1x_4$

  $f_2 = x_1'x_2 + x_1'x_3 + x_1x_4$

- Only x_1x_4 shared
- Sharing restricted to common cube

Multi-level:

- Datapath or control logic design
- Can share $x_2 + x_3$ between the two expressions
- Can use complex gates

  $g_1 = x_2 + x_3$

  $g_2 = x_1x_4$

  $f_1 = x_1y_1 + y_2$

  $f_2 = x_1'y_1 + y_2$

  (y_i is the output of gate g_i)

Multi-Level Logic Optimization

- Technology independent
- Decomposition/Restructuring
  - Algebraic
    - Polynomials
  - Functional
    - Don't cares
- Node optimization
  - Two-level logic optimization techniques are used

Decomposition / Restructuring

- Goal : given initial network, find best network
- Two problems:
  - Find good **common sub-functions**
  - How to perform **division**
- Example:

$f_1 = bcd + b'cd' + cd'e + a'c + cdf + abc'd'e' + ab'c'df'$

$f_2 = bdg + b'dfg + b'd'g + bd'eg$

decompose:

$f_1 = c(a' + x) + ac'x'$ $\quad x = d(b + f) + d'(b' + e)$

$f_2 = gx$ $\quad f_{dividend} = f_{divisor} f_{quotient} + f_{remainder}$

Basic Operations

1. decomposition

(single function)

$f = abc + abd + c'd' + a' + b'$

$f = xy + (xy)'$
$x = ab$
$y = c + d$

2. extraction

(multiple functions)

$f = (az + bz')cd + e$
$g = (az + bz')e'$
$h = cde$

$f = xy + e$
$g = xe'$
$h = ye$
$x = az + bz'$
$y = cd$

Basic Operations

3. Factoring

(series-parallel decomposition)

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

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

4. substitution

(with complement)

$g = a + b$
$f = a + bc + b'c'$

$f = g(a + c) + g'c'$

5. elimination

$f = ga + g'b$
$g = c + d$

$f = ac + ad + bc'd'$

"Division" plays a key role !!

Division

- Division: p is a Boolean divisor of f if $q \neq \phi$ and r exist such that $f = pq + r$
  - p is said to be a factor of f if in addition $r = \phi$:

    $$f = pq$$
  - q is called the **quotient**
  - r is called the **remainder**
  - q and r are **not unique**
- ***Weak division***: the unique algebraic division such that r has as few cubes as possible
  - The quotient q resulting from weak division is denoted by f / p (it is ***unique***)

Weak Division Algorithm

Weak_div(f, p):

U = Set $\{u_j\}$ of cubes in f with literals not in p deleted

V = Set $\{v_j\}$ of cubes in f with literals in p deleted

/* note that $u_j v_j$ is the j-th cube of f */

$V^i = \{v_j \in V : u_j = p_i\}$

$q = \cap V^i$

$r = f - pq$

return(q, r)

Weak Division Algorithm

- Example

common expressions

$$f = \boxed{acg + adg} + ae + \boxed{bc + bd + be + a'b}$$
$$p = ag + b$$
$$U = \boxed{ag + ag} + a + \boxed{b + b + b + b}$$
$$V = \boxed{c + d} + e + \boxed{c + d + e + a'}$$
$$V^1 = \boxed{c + d}$$
$$V^2 = \boxed{c + d} + e + a'$$
$$q = c + d = f/p$$

Algebraic Divisor

- Example:

  X = (a + b + c)de + f

  Y = (b + c + d)g + aef

  Z = aeg + bc
- Single-cube divisor: ae
- Multiple-cube divisor: b + c
- Extraction of common sub-expression is a global area optimization effort

Some Definitions about Kernels

- Definition: An expression is ***cube-free*** if no cube divides the expression evenly (i.e., cannot be factored)
  - *ab + c* is cube-free
  - *ab + ac = a (b + c)* is not cube-free
- Note: a cube-free expression must have more than one cube
  - *abc* is not cube-free
- Definition: The ***primary divisors*** of an expression f are the set of expressions

  D(f) = {f/c | c is a cube}

Kernels

- Definition: The ***kernels*** of an expression *f* are the set of expressions

  K(f) = {*g* | *g* ∈ *D(f)* and *g* is cube free}
- The kernels of an expression f are K(f) = {f/c}, where
  - / denote algebraic polynomial division
  - c is a cube
  - No cube divide f/c evenly (without any remainder)
- Naïve kernel computation method
  - Divide function by the elements of the power set of its support set
- The cube c used to obtain the kernel is the ***co-kernel*** for that kernel

Co-Kernels

- Definition: A cube *c* used to obtain the kernel $k = f/c$ is called a ***co-kernel*** of *k*. *C(f)* is used to denote the set of co-kernels of *f*.
- Example

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

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

Kernel	***Co-kernel***
a + b + c	df, ef
d + e	af, bf, cf
(a + b + c)(d + e)f + g	1

Kernels of Expressions

- Example:

  $f = x_1x_2x_3 + x_1x_2x_4 + x_3'x_2$

  $K = \{x_1x_3 + x_1x_4 + x_3',\ x_3 + x_4\}$

  - x_1x_2 is the co-kernel for the kernel $x_3 + x_4$
- Kernels can be used to factor an expression

  $f = x_2(x_1(x_3 + x_4) + x_3')$
- Key in finding ***common divisors*** between expressions

Common Divisor

- Theorem (Brayton & McMullen):

  f and *g* have a multiple-cube common divisor if and only if the intersection of a kernel of *f* and a kernel of *g* has more than one cube

$f_1 = x_1(x_2x_3 + x_2'x_4) + x_5$
$f_2 = x_1(x_2x_3 + x_2'x_5) + x_4$
$K(f_1) = \{x_2x_3 + x_2'x_4,$
$\quad x_1(x_2x_3 + x_2'x_4) + x_5\}$
$K(f_2) = \{x_2x_3 + x_2'x_5,$
$\quad x_1(x_2x_3 + x_2'x_5) + x_4\}$
$K_1 \cap K_2 = \{x_2x_3, x_1x_2x_3\}$

– f_1 and f_2 have no multiple-cube common divisor

$f_1 = x_1x_2 + x_3x_4 + x_5$
$f_2 = x_1x_2 + x_3'x_4 + x_5$
$K(f_1) = \{ x_1x_2 + x_3x_4 + x_5\}$
$K(f_2) = \{ x_1x_2 + x_3'x_4 + x_5\}$
$K_1 \cap K_2 = \{ x_1x_2 + x_5\}$

– f_1 and f_2 have multiple-cube common divisor

2024/4/2 Andy Yu-Guang Chen 49

Cube-Literal Matrix

- Cube-literal matrix
  - f = x1x2x3x4x7 + x1x2x3x4x8 + x1x2x3x5 + x1x2x3x6 + x1x2x9

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
$x_1x_2x_3x_4x_7$	1	1	1	1	0	0	1	0	0
$x_1x_2x_3x_4x_8$	1	1	1	1	0	0	0	1	0
$x_1x_2x_3x_5$	1	1	1	0	1	0	0	0	0
$x_1x_2x_3x_6$	1	1	1	0	0	1	0	0	0
$x_1x_2x_9$	1	1	0	0	0	0	0	0	1

2024/4/2 Andy Yu-Guang Chen 50

Cube-Literal Matrix & Rectangles

- A rectangle (R, C) of a matrix A is a subset of rows R and columns C such that

$$A_{ij} = 1 \forall\ i \in R,\ j \in C$$

  - Rows and columns need not to be continuous
- A prime rectangle is a rectangle not contained in any other rectangle
  - A prime rectangle indicates a {co-kernel, kernel} pair

Cube-Literal Matrix & Rectangles

- Example:

  R = {{1, 2, 3, 4},{1, 2, 3}}

  - co-kernel: $x_1x_2x_3$
  - kernel: $x_4x_7 + x_4x_8 + x_5 + x_6$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
$x_1x_2x_3x_4x_7$	1	1	1	1	0	0	1	0	0
$x_1x_2x_3x_4x_8$	1	1	1	1	0	0	0	1	0
$x_1x_2x_3x_5$	1	1	1	0	1	0	0	0	0
$x_1x_2x_3x_6$	1	1	1	0	0	1	0	0	0
$x_1x_2x_9$	1	1	0	0	0	0	0	0	1

Rectangles and Logic Synthesis

- **Single cube extraction**

F = abc + abd + eg
G = abfg
H = bd + ef
({1,2,4},{1,2}) <=> ab
({2,5},{2,4}) <=> bd

⇒

F = Xc + XY + eg
G = Xfg
H = Y + ef
X = ab
Y = bd

	a 1	b 2	c 3	d 4	e 5	f 6	g 7
abc 1	1	1	1	0	0	0	0
abd 2	1	1	0	1	0	0	0
eg 3	0	0	0	0	1	0	1
abfg 4	1	1	0	0	0	1	1
bd 5	0	1	0	1	0	0	0
ef 6	0	0	0	0	1	1	0

Summary

- Two-level logic optimization
  - Minimization criteria: number of product terms
  - Karnaugh maps [Kar53]
  - Quine-McCluskey [McC56]
- Multi-level logic optimization
  - Goal : given initial network, find best network
  - Two problems:
    - Find good **common sub-functions**
    - How to perform **division**
  - Weak Division Algorithm

Q&A

Andy, Yu-Guang Chen
Assistant Professor, Department of EE, NCU
Email: andyygchen@ee.ncu.edu.tw