Introduction to multi-level logic synthesis (automatic factoring)

Lecture 2

Optimization

- Combinational optimization
  - 2-level
  - Multi-level
- Sequential optimization
Two-Level Representation
- Applicable in only special situations
- Serve as the basis for multi-level optimization

Multi-Level Circuit
- More realistic view in most cases
Representation

• In order to develop a method for multi-level logic optimization, we need a representation for the algorithm to work on
  - Representation of a circuit to optimize
  - Independent of technology (65nm vs. 28nm, etc.)
  - Result of optimization can be evaluated effectively

• Two (circuit) representations
  - Network representation (network graph)
  - Boolean function representation
    • Factored Forms
    • OBDD (Ordered Binary Decision Diagram)

Representation Example

- A circuit is represented as a graph
- Each node is represented as a 2-level SOP
Example of Optimization – Node Elimination

Example of Optimization – Factoring
Example of Optimization – 2-level Minimization

- We see a SOP as a set of cubes
  - $ab + bc + ef = \{ab, bc, ef\}$

- We see a cube as a set of literals
  - $abc = \{a, b, c\}$

- For example
  - $F = ac + bc + ad$
  - $F$ contains 3 cubes $\{c1, c2, c3\}$
  - Where $c1 = \{a, c\}$, $c2 = \{b, c\}$, $c3 = \{a, d\}$
Assumption

• In 2-level logic synthesis, we assume that our final implementation is the same as how the function is represented
  - Literals are inputs
  - Use multi-input AND and 1 big OR
  - So, minimizing formula = minimizing implementation

• In multi-level logic synthesis, we assume that a node can be an arbitrary function
  - In one way, we may try to minimize the total number of "literals" used in the network
  - We do not consider what are available in the cell library
  - This allows us to develop a general theory here

In general, optimization is
But here, we only look for

- An efficient way to transform one representation into another
  - So that the cost (e.g. the # of literals used in the circuit) can be evaluated efficiently
  - So that we can apply local search to find the near-optimal solution
  - See 156A Lecture 8 for “local search”

- You can also look into here for more information
  - http://www.ece.cmu.edu/~ee760/760docs/lec07.pdf
  - Where it teaches you how to use SIS

- And this efficient way is “Factoring”

Factored Form

\[
\begin{align*}
& a \\
& a' \\
& ab'c \\
& ab + c'd \\
& (a + b)(c + a' + de) + f
\end{align*}
\]

- Where \(a, a', b', c\), are called literals
- Factored form can be derived from a SOP

\[
ace + ade + bce + bde + e'
\]

\[
\rightarrow e(a + b)(c + d) + e'.
\]
Definition

- A factored form is
  - A product or a sum, where
  - A product is
    - Either a single literal
    - Or a product of factored form
  - A sum is
    - either a single literal
    - Or a sum of factored form

\[
\begin{align*}
  a + b'c & \quad \text{Yes} \\
  ((a' + b)cd + e)(a + b') + e' & \quad \text{No}
\end{align*}
\]

Note:
- literal count $\propto$ transistor count $\propto$ area
- (however, area also depends on wiring)
Algebraic and Boolean

- $F = \{Ci\}$ is called an algebraic expression if in $F$, no cube contain another and no cube contains the form $xx$ or $xx'$
  - Otherwise, it is called Boolean expression (or absorptive)
  - For examples
    - $a+bc$ is algebraic
    - $a+ab$ is Boolean
- The support of $F$, denoted as $\text{supp}(F)$ is the set of variables used in $F$
  - Two expressions $F,G$ are called orthogonal if they have disjoint supports (denoted as $F \perp G$)
  - For examples
    - “$ab+c$” $\perp$ “$d'e+f$”
    - “$ab+c$” and “$c'+de$” are not orthogonal

Factored form is not unique

- There are 12 literals in the first form
- There is only 8 literals in the 3rd one
- Take the first one and multiply out, we can get the original expression
  - Without using $xx'=0$ and $xx=x$
- Take the 3rd one and multiply out, we get a different expression
  - Because we have $afag$
Algebraic factored form

- A factored form is said to be Algebraic if the SOP expression can be obtained by multiplying \( F \) out directly (without using \( xx' = 0 \) and \( xx = x \) and single cube containment)
  - Otherwise, it is called Boolean

\[
\begin{align*}
    a + bc & \quad (a + b + c + d)(a' + b' + e' + d') \\
    (a + b)(c + d) & \quad (af + b + c)(ag + d + e) \\
    (b + c)(d + e + ag) + (d + e + g)af, (a + b)(c + d)(e + f) + g + b(e + h).
\end{align*}
\]

Factoring tree

- A sub-tree is called a factor
  - \((a + b')\)
  - \(cd(a' + b)\)
- Two trees are equivalent if they represent the same function
- Two trees are syntactically equivalent if they are isomorphic
  - \((a + b)(c + d)e = (a + b)e(c + d)\)
Maximally factored

- A factored form is maximally factored if
  - For every sum of products, there are NO 2 syntactically equivalent factors in the products
  - For every product of sums, there are NO 2 syntactically equivalent factors in the sums

- Not maximally factored
  - \( ab + ac = a(b+c) \)
  - \( (a+b)(a+c) \)
  - Note that "\( \cdot \)" distributes over "\( + \)" and vice versa
    - \( (a+b)(a+c) = a+bc \)

In order to Factor, we need "Division" 

- Let \( f = x'z' + yz + xz \)
- Let \( p = x'+z \)
- We can have \( f = \)
  - \( (x'+z)(x+z') + x'y = \)
  - \( (x'+z)(y+z') + xz \)

- Where \( (x+z') \) is called a quotient
- And \( x'y \) is called a remainder
Factor VS. divisor

• (factor = \( g \)) Let \( f \) and \( g \) are Boolean functions satisfying
  - \( f \subseteq g \)
  - \( f \) can be written as \( g \cdot h \) (\( h \) is not unique), where
  - \( f \subseteq h \subseteq f + g' \)
• (divisor = \( g \)) If \( f \cdot g \neq 0 \) (not containment as above), \( f \) can be written as \( f = g \cdot h + r \) (not unique)
  - Where \( f \cdot g' \subseteq r \subseteq f \)
  - For a given \( r \), \( h \) can be chosen so that
    • \( f \cdot r' \subseteq h \subseteq f + g' \)
• \( f \cdot g \) is an algebraic product if they have disjoint support sets
  - \((a+b)(c+d)\) is an algebraic product

Division

• Given \( F \) and \( P \), a division generates \( Q \) and \( R \)
  - Such that \( F = PQ + R \)
• If \( PQ \) is an algebraic product, this is an algebraic division
  - \( P \) is an algebraic factor
• We can perform weak division, such that
  - \( PQ \) is an algebraic product
  - \( R \) has as few cubes as possible
  - \( PQ + R \) and \( F \) have the same set of cubes
  - See textbook for the algorithm
Example

• $F = ad + abc + bcd$, $P = a+bc$

• (1) for “a”, look into $F$ to collect all cubes that contains “a”
  - They are “ad” and “abc”
  - So we have $(p_1)d$ and $(p_1)bc$

• (2) for “bc”, we have “abc” and “bcd”
  - So we have “$(p_2)a$” and $(p_2)d$

• Observe that d multiply both $(p_1)$ and $(p_2)$
  - So we get $(p_1+p_2)d + abc$
  - $= (a+bc)d + abc$

Kernel – to find good divisors

Definition 10.5.1 An expression is cube-free if no cube divides the expression evenly, that is,

$$\neg \exists C \text{ such that } F = QC$$

(no remainder), and $C$ is a cube.

For instance, $ab + c$ is cube-free; $ab + ac$ and $abc$ are not cube-free. A cube-free expression must have more than one cube.

Definition 10.5.2 The primary divisors of an algebraic expression $F$ are the set of expressions

$$D(F) = \{ F/c | c \text{ is a cube} \}. \quad (10.5)$$

The kernels of an expression $F$ are the set of expressions

$$K(F) = \{ g | g \in D(F) \text{ and } g \text{ is cube-free} \}. \quad (10.6)$$

In other words, the kernels of an expression $F$ are the cube-free primary divisors of $F$.

“c” is called the co-kernel of $K$
Example

\[ F = adf + aef + bdf + bef + cdf + cef + bfg + h \]
\[ = (a + b + c)(d + e)f + bfg + h \]

<table>
<thead>
<tr>
<th>kernel</th>
<th>co-kernel</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d + e )</td>
<td>( af, cf )</td>
<td>0</td>
</tr>
<tr>
<td>( d + e + g )</td>
<td>( bf )</td>
<td>0</td>
</tr>
<tr>
<td>( a + b + c )</td>
<td>( df, ef )</td>
<td>0</td>
</tr>
<tr>
<td>( (a + b + c)(d + e) + bg )</td>
<td>( f )</td>
<td>1</td>
</tr>
<tr>
<td>( ((a + b + c)(d + e) + bg)f + h )</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

F/a = df+ef is not a cube-free divisor
A kernel may have more than 1 co-kernel
If the original expression is cube free, a co-kernel can be the “1”

The usage of kernels

- The fundamental theorem is used to detect if two or more expressions have any common algebraic divisor than just the single cubes

- **Fundamental Theorem**
  - If two expressions F and G have the property that for any pair of kernels, \( K_F \) and \( K_G \), they have at most 1 term in common,
  - Then, F and G have no common non-trivial algebraic divisors other than just a single cube
**In another words**

- If F and G have a common *more-than-1-cube* divisor
  - You can find $K_F$ and $K_G$, such that **intersections of these two kernels give an expression with more than just 1 cube**
  - And that expression is your common divisor

- $F = ae+be+cde+ab$, $G = ad+ae+bd+be+bc$
- $F/e = a+b+cd$, $G/e$ or $G/d = a+b$
- $\{a,b,cd\} \cap \{a,b\} = \{a,b\}$
- $(a+b)$ is your common divisor for F and G

**Remaining questions**

- How to compute all kernels efficiently?
  - See book
- How to choose a particular divisor for factoring?
  - Apply heuristics
    - Very much like expansion/reduction iterations
- You can look into here for more information
  - [http://www.ece.cmu.edu/~ee760/760docs/lec07.pdf](http://www.ece.cmu.edu/~ee760/760docs/lec07.pdf)
  - Where it teaches you how to use SIS
- You can download SIS and play with it
  - It has everything for you to understand the synthesis process