Synthesis of 2-level Logic (Logic Optimization)

Lecture 6

Quine’s Theorem

• **Theorem (Quine):** A minimal SOP implementation of a function must always consist of a sum of *prime implicants*
  – Apply to 2-level logic only
  – So finding prime implicants are crucial

• **Implicant:** a product term \( p \) that is included in the function \( f \)
  – \( F = xy' + yz \Rightarrow xy' \) and \( xyz \) are implicant

• **Prime Implicant:** an implicant that is not included in any other implicant
  – \( xy' \) is prime, but \( xyz \) is not
Consensus

- X ⇒ Y = X' + Y
- (S ⇒ H) ∧ (H ⇒ M) ⇒ (S ⇒ M)
- (S' + H) ∧ (H' + M) ⊇ (S' + M)
  - H and H' can be cancelled
- Similarly
  - S'H + MH' ⊇ S'M
  - H and H' can be cancelled as well

Tabular Method

- Given an f is SOP form, we try to compute all prime implicants for f
- First, we get minterm canonical form

f = x'y' + wxy + x'y'z + wy'z
wx'y'z, w'x'y'z, wx'y'z', w'x'y'z'
wxyz + wxyz'
wxy'z' + w'x'y'z'
f = w'x'y'z' + w'x'y'z + w'x'y'z' + wxyz' + wxy'z' + wxyz + wxyz'
wx'y'z + wx'yz + wxy'z + wxyz
Tabular Method

<table>
<thead>
<tr>
<th></th>
<th>1: w'x'y'z'</th>
<th>1,2: w'x'y'</th>
<th>1,3,4: w'x'z'</th>
<th>1,4,5: x'y'z'</th>
</tr>
</thead>
<tbody>
<tr>
<td>2: w'x'y'z'</td>
<td>2,5: x'y'z'</td>
<td>3,6: x'yz'</td>
<td>4,5: wx'y'</td>
<td>4,6: wx'z'</td>
</tr>
<tr>
<td>3: w'x'yz'</td>
<td>3,6: x'yz'</td>
<td>4,5: wx'y'</td>
<td>4,6: wx'z'</td>
<td></td>
</tr>
<tr>
<td>4: wx'y'z'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5: wx'y'z'</td>
<td>5,8: wy'z'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6: wx'yz'</td>
<td>6,7: wzy'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: wxyz'</td>
<td>7,9: wxy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8: wxy'z</td>
<td>8,9: wxz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9: wxyz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Pls: wy'z, wzy', wxy, wxz, x'y', x'z'

Find a cover

• Pls: wy'z, wzy', wxy, wxz, x'y', x'z'
• \( f = x'y' + wxy + x'yz' + wy'z \)

\[
\begin{array}{cccccc}
\text{wy'z, wzy', wxy, wxz, x'y', x'z'} & 0 & 0 & 0 & 1 & 0 \\
x'y' & 0 & 0 & 0 & 0 & 1 \\
wxy & 0 & 0 & 1 & 0 & 0 \\
x'yz' & 0 & 0 & 0 & 0 & 1 \\
wyz' & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\( F = x'z' + x'y' + wxy + wy'z \)
Covering Problem

- In general, finding a minimal cover is a hard problem
  - This is a typical NP-complete problem
  - Typical solution
    - Branch and bound algorithms
    - Various heuristics to prune search space

\[
\begin{array}{cccc}
  x'y' & x'z' & y'z' & yz \\
  x'y'z' & 0 & 1 & 1 & 0 \\
x'yz' & 1 & 0 & 0 & 1 \\
x'yz & 1 & 0 & 0 & 1 \\
xyz & 0 & 0 & 0 & 1 \\
x'y'z & 0 & 0 & 1 & 0 \\
\end{array}
\]

Solutions:
\[x'y+y'z'+yz,\]
or
\[x'z'+y'z'+yz\]

Problem Re-Formulation

\[
\begin{array}{cccc}
  x'y' & x'z' & y'z' & yz \\
  p1 & p2 & p3 & p4 \\
x'y'z' & 0 & 1 & 1 & 0 \\
x'yz' & 1 & 1 & 0 & 0 \\
x'yz & 1 & 0 & 0 & 1 \\
xyz & 0 & 0 & 0 & 1 \\
x'y'z & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[(p2+p3)(p1+p2)(p1+p4)p3p4=1\]

- Problem: Given a boolean formula \( f \), find a minimal assignment to satisfy \( f \) (to make \( f=1 \))
  - This is a **Min-SAT** problem (minimal satisfiability)
  - There are many SAT tools for EDA applications
Quine-McCluskey

- The above process is called Quine-McCluskey procedure
  - Use tabular method to find prime implcants
  - Use matrix to find a minimal cover
- Problem 1:
  - Tabular method is too expensive
  - Require $O(2^n)$ rows
- Problem 2:
  - SAT is the first NP-complete problem
  - MIN-SAT is harder than SAT

In Reality

- In reality, many problems in EDA can be translated into SAT or MIN-SAT problems
- In theory, those are hard problems
- However, for practical use, some heuristics can usually make them run fast enough
  - Ex. For 2-level synthesis, the circuits are usually not too large
  - The functions are usually not too complicate
  - So the tools can work fine most of the time
Back To Tabular Method

• Tabular method is impractical because for most cases, it needs to start with $O(2^n)$ entries

• We need a method that can avoid the expansion of the table
  – Recursive Computation of Prime Implcants
    • Follow so-called Iterated Consensus

Iterated Consensus

• In essence, the tabular method is based on
  – $Xy + Xy’ = X$
  – This is called distance-1 merging

• Now we transform the problem of finding all Pis into a problem of finding a complete sum so we can apply iterated consensus

• (Complete Sum) A SOP is complete iff
  – No term includes any other term
  – The consensus of any two terms either does not exist or is contained in some other term
Example: Complete Sum

• $x_1 x_2 + x_2' x_3 + x_2 x_3 x_4$

• **Step 1: apply iterated consensus**
  – Add in $x_1 x_3$
  – Add in $x_3 x_4$

• **Step 2: remove contained term(s)**
  – $x_1 x_2 + x_2' x_3 + x_2 x_3 x_4 + x_1 x_3 + x_3 x_4$
  – Remove $x_2 x_3 x_4$
  – Result: $x_1 x_2 + x_2' x_3 + x_1 x_3 + x_3 x_4$
  – This is a complete sum

Recursive Procedure for CS

• **Theorem:**
  – The SOP obtained from two complete sums $F_1$ and $F_2$ by $F_1 \cdot F_2$ is a complete sum for $F_1 F_2$
    • Multiply $F_1$ and $F_2$ using $x x' = 0$
    • Remove all terms contained in other terms

• **Ex.** $(x_1+x_2)(x_2'+x_3)(x_3+x_4)$
  – Each is a complete sum
  – $(x_1 x_2' + x_1 x_3 + x_2 x_3)(x_3 + x_4)$
    • Two complete sums
  – $(x_1 x_2' x_3 + x_1 x_2' x_4 + x_1 x_3 + x_1 x_3 x_4 + x_2 x_3 + x_2 x_3 x_4)$
    • The complete sum
Recursive Procedure for CS

- Take any function \( f \), make it as POS form
  \[ f(x_1, x_2, \ldots, x_n) = [x_i + f(x_i=0)] [x_i' + f(x_i=1)] \]
  - This is called Shannon Expansion
  - Finding the CS for \( f \) is equivalent to
    - Finding the CS for \([x_i + f(x_i=0)]\)
    - Finding the CS for \([x_i' + f(x_i=1)]\)
    - Then, multiply the two and eliminate contained terms
  - “\( x_i \)” is called the splitting variable
  - A good choice of \( x_i \) should make both \([x_i + f(x_i=0)]\) and \([x_i' + f(x_i=1)]\) simpler
  - Heuristic: select the \( x_i \) with the most occurrences

Example

- \( F = v'xyz + v'w'z + v'x'z' + v'wxz + w'yz' + vwz' + vwx'z \)
- Find a splitting variable
  - \( v \) (6), \( w \) (5), \( x \) (2), \( y \) (2), \( z \) (7) ⇒ choose \( z \)
  - \( G_0 = F(z=0) = v'x' + w'y + vw \)
  - \( G_1 = F(z=1) = v'xy + v'w' + v'wx + vwx' \)
  - \( G_1 = v'xy + v'w' + v'wx + vwx' \)
  - Choose \( v \)
    - \( H_0 = G_1 (v=0) = xy + w' + wx \)
    - \( H_1 = G_1 (v=1) = wx' \)
  - \( H_1 = wx' \) is a complete sum
  - \( H_0 = xy + w' + wx = xy + w'(x+x') + wx = xy + x + w'x' = x + w'x' = x + w' \) is a complete sum
  - \( H_1 \cdot H_0 = (v' + wx') [v' + (x+w')] = v'x + v'w' + vwx' = G_1 \)
  - …… exercise!
ESPRESSO

• A 2-level synthesis and logic minimization tool
  – Step 1: Recursive procedure to find a complete sum for the given function
  – Step 2: find a minimal cover
  – Step 3: implement the result as SOP
    • AND-OR tree with Inverters placed properly

This is one of the earliest and most successful synthesis tool

NMOS NAND-NAND PLA

• This is a basic idea for FPGA implementation