Library binding

Given an unbound logic network and a set of library cells:
- Transform network into an interconnection of instances of library cells.
- Optimize area, possible under delay constraints.
- Optimize delay, possible under area constraints.

Called also technology mapping:
- Method used for re-designing circuits in different technologies.

Library models

- Combinational elements:
  - Single-output functions: e.g. AND, OR, AOI.
  - Compound cells: e.g. adders, encoders.

- Sequential elements:
  - Registers, counters.

- Miscellaneous:
  - Schmitt triggers.
Major approaches

- Rule-based systems:
  - Mimic designer activity.
  - Handle all types of cells.

- Heuristic algorithms:
  - Restricted to single-output combinational cells.

- Most tools use a combination of both.

Rule-based library binding

- Binding by stepwise transformations.

- Data-base:
  - Set of patterns associated with best implementation.

- Rules:
  - Select subnetwork to be mapped.
  - Handle high-fanout problems, buffering, etc.

Example

- [Diagram of logic gates]

- Search for a sequence of transformations.

- Search space:
  - *Breadth* (options at each step).
  - *Depth* (look-ahead).

- *Meta-rules* determine dynamically breadth and depth.
Rule-based library binding

- Advantages:
  - Applicable to all kinds of libraries.
- Disadvantages:
  - Large rule data-base:
    * Completeness issue.
    * Formal properties of bound network.
  - Data-base updates.

Algorithms for library binding

- Restricted to s-output combinational cells.
- Fast and efficient:
  - Quality comparable to rule-based systems.
- Library description/update is straightforward:
  - Each cell modeled by its function or equivalent pattern.

Problem analysis

- Matching:
  - A cell matches a sub-network if their terminal behavior is the same.
  - Function \( f(x) \) matches \( g(y) \),
    * if there exists a permutation matrix \( P \), such that \( f(x) = g(P \cdot x) \) is a tautology.
- Covering:
  - A cover of an unbound network is a partition into subnetworks which can be replaced by library cells.

Assumptions

- Network granularity is fine.
  - Decomposition into simple base functions.
  - Example: 2-input AND, OR, NAND, NOR.
- Trivial binding:
  - Replacement of each vertex by base cell.
Example

- Vertex covering:
  - Covering v1: \((m_1 + m_4 + m_5)\).
  - Covering v2: \((m_2 + m_4)\).
  - Covering v3: \((m_3 + m_5)\).
- Input compatibility:
  - Match \(m_2\) requires \(m_1\):
    \((m_2^2 + m_1)\).
  - Match \(m_3\) requires \(m_1\):
    \((m_3^2 + m_1)\).
- Overall binate clause:
  \((m_1 + m_4 + m_5)(m_2^2 + m_4)(m_3^2 + m_5)(m_3^2 + m_1) = 1\).

Heuristic algorithms

- Decomposition:
  - Cast network and library is standard form.
  - Decompose into base functions.
  - Example: NAND2 and INV.
- Partitioning:
  - Break network into cones.
  - Reduce to many multi-input single-output subnetworks.
- Covering:
  - Cover each subnetwork by library cells.
Heuristic algorithms

- Structural approach:
  - Model functions by *patterns*.
    - Example: trees, dags.
  - Rely on pattern matching techniques.

- Boolean approach:
  - Use Boolean models.
  - Solve *tautology* problem.
  - More powerful.
Example

**Boolean versus structural matching**

- \( f = xy + x'y' + y'z \)

- \( g = xy + x'y' + xz \)

- Function equality is a tautology:
  - Boolean match.

- Patterns may be different:
  - Structural match may not be found.

---

**Structural matching and covering**

- Expression patterns:
  - Represented by dags.

- Identify pattern dags in network:
  - Sub-graph isomorphism.

- Simplification:
  - Use tree patterns.
Tree-based matching

- Network:
  - Partitioned and decomposed:
    * NOR2 (or NAND2) + INV.
    * Generic base functions.
  - Subject tree.
- Library:
  - Represented by trees.
  - Possibly more than one tree per cell.
- Pattern recognition:
  - Simple binary tree match.
  - Aho-Corasick automaton.

Binary tree match

Example: match $f = a + b'$ to library

- Decompose into $f_d = (a'b')'$.
  - Consider INV:
    * Number of children mismatch.
  - Consider NAND2:
    * Same number of children.
    * Recursive call:
      - Leaves found: MATCH!

```
match(u, v) {
  if (u is a leaf) return (true)
  else {
    if (v is a leaf) return (false);
    if (out_degree(v) != out_degree(u)) return (false);
    if (out_degree(v) == 1) {
      uc = child of u ; vc = child of v ;
      return (match(uc, vc));
    }
    else {
      ul = left-child of u ;
      ur = right-child of u ;
      vl = left-child of v ;
      vr = right-child of v ;
      return (match(ul, vl) + match(ur, vr) +
               match(ul, vl) + match(ur, vr));
    }
  }
}
```
Tree covering

• Dynamic programming:
  – Visit subject tree bottom-up.

• At each vertex:
  – Attempt to match:
    * Locally rooted subtree.
    * All library cells.

• Optimum solution, for the subtree.

Example

• The cell pattern tree and the rooted subtree are isomorphic.
  – The vertex is labeled with the cell cost.

• The cell tree is isomorphic to a subtree with leaves \( L \).
  – The vertex is labeled with the cell cost plus the labels of the vertices \( L \).

• There is no match:
  – Cannot happen when base functions are in the library.
Minimum-area cover.

Area costs:
- INV: 2; NAND2: 3; AND2: 4; AOI21: 6.

Best choice:
- AOI21 fed by a NAND2 gate.

Dynamic programming approach.

Cost related to gate delay.

Delay modeling:
- Constant gate delay.
  » Straightforward.
- Load-dependent delay:
  » Load fanout unknown.
  » Binning techniques.
Minimum delay cover
constant delays

- The cell pattern tree and the rooted subtree are isomorphic.
  - The vertex is labeled with the cell delay.
- The cell tree is isomorphic to a subtree with leaves $L$.
  - The vertex is labeled with the cell cost plus the maximum of the labels of $L$.

Example

- Inputs data-ready times are 0 except for $t_d = 6$.
- Constant delays:
  - INV:2; NAND2:4; AND2:5; AOI2:10.
- Compute data-ready times bottom-up:
  - $t_x = 4, t_y = 2; t_z = 10 t_d = 14$.
- Best choice:
  - AND2, two NAND2 and an INV gate.

Minimum delay cover
load-dependent delays

- Model:
  - Assume a finite set of load values.
- Dynamic programming approach:
  - Compute an array of solutions for each possible load.
    - For each input to a matching cell the best match for any load is selected.
  - Optimum solution, when all possible loads are considered.
Example

• Inputs data-ready times are 0 except for $t_d = 6$.

• Load-dependent delays:
  – INV:1+l; NAND2:3+l; AND2:4+l; AOI21:9+l.

• Loads:
  – INV:1; NAND2:1; AND2:1; AOI21:1.

• Same solution as before.

Example

• Inputs data-ready times are 0 except for $t_d = 6$.

• Load-dependent delays:
  – INV:1+l; NAND2:3+l; AND2:4+l; AOI21:9+l $\text{SINV}:1+0.5l$.

• Loads:
  – INV:1; NAND2:1; AND2:1; AOI21:1; $\text{SINV}:2$.

• Assume output load is 1:
  – Same solution as before.

• Assume output load is 5:
  – Solution uses SINV cell.

Example

<table>
<thead>
<tr>
<th>Network</th>
<th>Subject graph</th>
<th>Vertex</th>
<th>Match</th>
<th>Gate</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Loads1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>x</td>
<td>x</td>
<td>NAND2(b,c)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>y</td>
<td>INV(a)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z</td>
<td>Z</td>
<td>NAND2(x,a)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W</td>
<td>W</td>
<td>NAND2(y,z)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
<td>O</td>
<td>INV(w)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AND2(y,z)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AOI21(x,y,z)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\text{SINV}(w)$</td>
<td>20</td>
</tr>
</tbody>
</table>

Boolean covering

• Decompose network into base functions.

• When considering vertex $v_i$:
  – Construct clusters by local elimination.
  – Several functions associated with $v_i$.

• Limit size and depth of clusters.
Example

\[
\begin{align*}
    f_{3,1} &= xy; \\
    f_{3,2} &= x(a + c); \\
    f_{3,3} &= (e + z)y; \\
    f_{3,4} &= (e + z)(a + c); \\
    f_{3,5} &= (e + c')(dy); \\
    f_{3,6} &= (e + c' + f)(a + c); \\
\end{align*}
\]

Boolean matching

- Pattern function \( g(y) \):
  - Cell behavior.

- Cluster function \( f(x) \):
  - Sub-network behavior.

- Tautology check:
  - Exists a permutation matrix \( P \), such that \( f(x) = g(P \cdot x) \) is a tautology?

- Tautology check must be done over all permutations of input variables.

Example

- Cluster function: \( f = abc \).
  - Symmetries: \( \{(a, b, c)\} \) – unate.

- Pattern functions:
  - \( g_1 = a + b + c \)
    * Symmetries: \( \{(a, b, c)\} \) – unate.
  - \( g_2 = ab + c \)
    * Symmetries: \( \{(a, b)(c)\} \) – unate.
  - \( g_3 = abc' + d' \)
    * Symmetries: \( \{(a, b, c)\} \) – binate.
Library binding and polarity assignment

- Search for lower cost solution by not constraining the signal polarities.
- Most circuit allow us to chose the input/output signal polarities.
- Approaches:
  - Structural covering.
  - Boolean covering.

Structural covering and polarity assignment

- Pre-process subject network:
  - Add inverter pairs between NANDs.
  - Provide signals with both polarity.
- Add inverter-pair cell to the network:
  - To eliminate unneeded pairs.
  - Cell corresponds to a connection with zero cost.

Example

Boolean covering and polarity assignment

- Extend definition of matching:
  - Exists a permutation matrix $P$, and complementation operators $N$ such that $f(x) = N g(PN x)$ is a tautology?
- $\mathcal{NP}$ classification of functions.
**Example**

- Cluster functions $f(a, b)$ in the set:
  - $\{a + b, a' + b, a + b', a' + b', ab, a'b, ab', a'b'\}$
- Match pattern function $g(x, y) = x + y$.

**Concurrent optimization and library binding**

- **Motivation:**
  - Logic simplification is usually done prior to binding.
  - Logic simplification/substitution can be combined with binding.

- **Mechanism:**
  - Binding induces some *don't care* conditions.
  - Boolean covering/matching can exploit *don't care* conditions.

---

**Concurrent logic optimization and binding**

- **Standard approach:**
  - Use *don’t care* information for simplification.
  - Bind cells after simplification.

- **Concurrent approach:**
  - Library binding and optimization.
  - Lower cost solutions.
• Assume $v_2$ is bound to $OR3(c', b, e)$.
• Don't care set includes $x \oplus (c' + b + e)$.
• Consider $f_j = x(a + c)$ with $CDC = x$.
• No simplification. Mapping into $AOI$ gate.
• Matching with DC. Mapping into $MUX$ gate.

Matching compatibility graph

• Vertices:
  - $N\mathcal{P}$ classes of functions.
  - Representative functions.

• Edges:
  - Representative functions differ in one minterm.

• Path:
  - Boolean cover (of the error).
Compatibility graph traversal

- Given cluster function $f$.
  - Find representative function (vertex).

- For all paths leading to a library element:
  - Check if error is contained in local don't care set.

Cluster function: $f = x(a + c)$ with $CDC = x'c'$.

Representative vertex $v_5$.

Vertices reachable from $v_5$:
- $\{v_9, v_{10}, v_{11}\}$.
  - Because error included in the don't care set.

Only vertex $v_9$ is a library cell.

Multiplexer gate:
- Representative function $a'b' + bc$ is in the same $N'P,A'$ class as $ab + \overline{b}c$ and thus can match $ce + \overline{c}a$. 
Summary

- Library binding is very important.

- Rule-based approach:
  - General, sometimes inefficient.

- Algorithmic approach:
  - Pattern-based: fast, but limited.
  - Boolean: slower, but promising.