LIBRARY BINDING

© Giovanni De Micheli

Stanford University
Outline

- Modeling and problem analysis.
- Rule-based systems for library binding.
- Algorithms for library binding:
  - Structural covering/matching.
  - Boolean covering/matching.
- Concurrent optimization and binding.
Library binding

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

\[\text{Diagram 1} \Rightarrow \text{Diagram 2}\]

\[\text{Diagram 3} \Rightarrow \text{Diagram 4}\]
Strategies

- 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

- Mainly for single-output combinational cells.

- Fast and efficient:
  - Quality comparable to rule-based systems.

- Library description/update is simple:
  - 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.
  - Input-variable assignment problem.

- 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 base functions.
    * 2-input $AND, OR, NAND, NOR$.

- Trivial binding:
  - Replacement of each vertex by base cell.
Example

(a) $z = a + w$
$w = x + y$
$y = d * u$
$x = b + c$
$u = ef$

(b)  

c

(c)  

c

(d)  

c
Example

<table>
<thead>
<tr>
<th>Library</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND2</td>
<td>4</td>
</tr>
<tr>
<td>OR2</td>
<td>4</td>
</tr>
<tr>
<td>OA21</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ x = b + c \]
\[ y = a \times x \]
\[ z = x \times d \]

(a)  
(b)  
(c)  

m1: \{v1, OR2\}  
m2: \{v2, AND2\}  
m3: \{v3, AND2\}  
m4: \{v1, v2, OA21\}  
m5: \{v1, v3, OA21\}  

(d)  
(e)  
(f)
Example

- Vertex covering:
  - Covering $v_1$: $(m_1 + m_4 + m_5)$.
  - Covering $v_2$: $(m_2 + m_4)$.
  - Covering $v_3$: $(m_3 + m_5)$.

- Input compatibility:
  - Match $m_2$ requires $m_1$:
    * $(m'_2 + m_1)$.
  - Match $m_3$ requires $m_1$:
    * $(m'_3 + m_1)$.

- Overall $binate$ clause:
  - $(m_1 + m_4 + m_5)(m_2 + m_4)(m_3 + m_5)(m'_2 + m_1)(m'_3 + m_1) = 1$
Heuristic algorithms

- Decomposition:
  - Cast network and library in 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.
Decomposition
Partitioning
Covering
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.
Example

Boolean versus structural matching

\[ f = xy + x'y' + y'z \]

\[ g = xy + x'y' + xz \]

*Patterns do not match.*
Structural matching and covering

- Expression patterns:
  - Represented by dags.

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

- Simplification:
  - Use tree patterns.
Example
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.
Simple library

INV
NAND2
AND2
NOR2
OR2
AOI21
AOI22

(a) (b) (c) (d)

N1v N2v t1.1
t2.1
t2.2
t3.1
t3.2
t4.1
t4.2
t5.1
t5.2
I1v
I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v 16A.1 16A.2 16A.3 16B.1 16B.2 16B.3

I1v
I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v I1N1v I1N2v 17.1 17.2 17.3 17.4
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

**SUBJECT TREE**

- r
  - s
  - t
  - u

**PATTERN TREES**

- t1 (cost = 2, INV)
- t2 (cost = 3, NAND)
- t3 (cost = 4, AND)
- t4 (cost = 5, OR)
Example

Match of s: t1
   cost = 2
Match of u: t2
   cost = 3

Match of t: t1
   cost = 2+3=5

Match of r: t2
   cost = 3+2+4 =9

Match of r: t4
   cost = 5+3=8
Example

- Minimum-area cover.

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

- Best choice:
  - AOI21 fed by a NAND2 gate.
### 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>x</td>
<td>t2</td>
<td>NAND2(b,c)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>t1</td>
<td>INV(a)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z</td>
<td>t2</td>
<td>NAND2(x,d)</td>
<td>2*3=6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>w</td>
<td>t2</td>
<td>NAND2(y,z)</td>
<td>3*3+2=11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o</td>
<td>t1</td>
<td>INV(w)</td>
<td>3<em>3+2</em>2=13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t3</td>
<td></td>
<td>AND2(y,z)</td>
<td>2*3+4+2=12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t6B</td>
<td></td>
<td>AOI21(x,d,a)</td>
<td>3+6=9</td>
</tr>
</tbody>
</table>
Minimum delay cover

- 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:
  - \( \text{INV}:2; \text{NAND2}:4; \text{AND2}:5; \text{AOI21}:10 \).

- Compute \textit{data-ready} times bottom-up:
  - \( t_x = 4, t_y = 2; t_z = 10t_w = 14 \).

- Best choice:
  - \( \text{AND2}, \) two NAND2 and an INV gate.
### 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>o</td>
<td></td>
<td>x</td>
<td>t2</td>
<td>NAND2(b,c)</td>
<td>4</td>
</tr>
<tr>
<td>w</td>
<td></td>
<td>y</td>
<td>t1</td>
<td>INV(a)</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td></td>
<td>z</td>
<td>t2</td>
<td>NAND2(x,d)</td>
<td>6 + 4 = 10</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td>w</td>
<td>t2</td>
<td>NAND2(y,z)</td>
<td>10 + 4 = 14</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>o</td>
<td>t1</td>
<td>INV(w)</td>
<td>14 + 2 = 16</td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>t3</td>
<td>AND2(y,z)</td>
<td>10 + 5 = 15</td>
<td></td>
</tr>
<tr>
<td>t6B</td>
<td></td>
<td></td>
<td>AOI21(x,d,a)</td>
<td>10 + 6 = 16</td>
<td></td>
</tr>
</tbody>
</table>
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+1; NAND2:3+1; AND2:4+1; AOI21:9+1.

- 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:
  - $\text{INV:1} + 1$; $\text{NAND2:3} + 1$; $\text{AND2:4} + 1$; $\text{AOI21:9} + 1$; $\text{SINV:1} + 0.5I$.

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

- 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>x</td>
<td>t2</td>
<td>NAND2(b,c)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>t1</td>
<td>INV(a)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z</td>
<td>t2</td>
<td>NAND2(x,d)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>w</td>
<td></td>
<td>NAND2(y,z)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o</td>
<td>t1</td>
<td>INV(w)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t3</td>
<td></td>
<td>AND2(y,z)</td>
<td>18.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t6B</td>
<td></td>
<td>AOI21(x,d,a)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SINV(w)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Load=1</th>
<th>Load=2</th>
<th>Load=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>z</td>
<td>10</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>w</td>
<td>14</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>t6B</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>18.5</strong></td>
</tr>
</tbody>
</table>
Library binding and polarity assignment

- Search for lower cost solution by not constraining the signal polarities.

- Most circuit allow us to choose 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 library:
  – To eliminate unneeded pairs.
  – Cell corresponds to a connection with zero cost.
Example
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

\[ f_{j,1} = xy; \]
\[ f_{j,2} = x(a + c); \]
\[ f_{j,3} = (e + z)y; \]
\[ f_{j,4} = (e + z)(a + c); \]
\[ f_{j,5} = (e + c' + d)y; \]
\[ f_{j,6} = (e + c' + d)(a + c); \]
Boolean matching
\( \mathcal{P} \)-equivalence

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

- **Pattern function** \( g(y) \): cell behavior.

- **\( \mathcal{P} \)-equivalence:**
  - Exists a permutation operator \( \mathcal{P} \), such that \( f(x) = g(\mathcal{P} x) \) is a tautology?

- **Approaches:**
  - Tautology check over all input permutations.
  - Multi-rooted pattern ROBDD capturing all permutations.
Input/output polarity assignment

- Allow for reassignment of input/output polarity.

- $\mathcal{NP}_N$ classification of Boolean functions.

- $\mathcal{NP}_N$-equivalence:
  - Exists a permutation matrix $\mathcal{P}$, and complementation operators $\mathcal{N}_i, \mathcal{N}_o$ such that $f(\mathbf{x}) = \mathcal{N}_o g(\mathcal{P} \mathcal{N}_i \mathbf{x})$ is a tautology?

- Variations:
  - $\mathcal{N}$-equivalence, $\mathcal{PN}$-equivalence
Boolean matching

- *Pin assignment* problem.
  - Map cluster variables $x$ to pattern vars $y$.
  - Characteristic equation: $A(x, y) = 1$.

- Pattern function under variable assignment:
  - $g_A(x) = S_y A(x, y) g(y)$

- *Tautology problem.*
  - $f(x) \oplus g_A(x)$
  - $\forall x (f(x) \oplus S_y (A(x, y) g(y)))$
Example

- Assign \( x_1 \) to \( y'_2 \) and \( x_2 \) to \( y_1 \).

- Characteristic equation:
  
  \[- A(x_1, x_2, y_1, y_2) = (x_1 \oplus y_2)(x_2 \overline{\oplus} y_1) \]

- AND pattern function:
  
  \[- g = y_1y_2 \]

- Pattern function under assignment:
  
  \[- S_{y_1,y_2}A g = \\
  \quad = S_{y_1,y_2}(x_1 \oplus y_2)(x_2 \overline{\oplus} y_1)y_1y_2 = x_2x'_1 \]
Signatures and filters

- Capture some properties of Boolean functions.

- If signatures do not match, there is no match.

- Used as filters to reduce computation.

- Signatures:
  - Unateness.
  - Symmetries.
  - Co-factor sizes.
  - Spectra.
Filters based on unateness and symmetries

• Any pin assignment must associate
  – unate (binate) variables in \( f(\mathbf{x}) \)
  with unate (binate) variables in \( g(\mathbf{y}) \).

• Variables or groups of variables
  – that are interchangeable in \( f(\mathbf{x}) \)
    must be interchangeable in \( g(\mathbf{y}) \).
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.
  - $g_3 = abc' + a'b'c$
    * Symmetries: $\{(a, b, c)\}$ – binate.
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.
  - Exploit don’t cares as degrees of freedom in matching.
Boolean matching with *don't care* conditions

- Given \( f(x), f_{DC}(x) \) and \( g(y) \):
  - \( g \) matches \( f \) if \( g \) is equivalent to \( \tilde{f} \), where \( f \cdot f'_{DC} \leq \tilde{f} \leq f + f_{DC} \)

- Matching condition:
  - \( \forall x(f_{DC}(x) + f(x) \oplus S_y (A(x, y) g(y))) \)
Example

- Assume $v_x$ is bound to $OR3(c', b, e)$.
- *Don’t care* set includes $x \oplus (c' + b + e)$.
- Consider $f_j = x(a + c)$ with $CDC = x'c'$.
- No simplification. Mapping into AOI gate.
- Matching with DC. Mapping into MUX gate.
Example
Example
Extended matching

- Augment pattern function with mux function.
  - Each cell input can be routed to any cluster input (or voltage rail).
  - Input polarity can be changed.
  - Cell and cluster may differ input size.

- Define composite function $G(x, c)$:
  - Pin assignment is determining $c$.

- Matching formula: $M(c) = \forall x [G(x, c) \oplus f(x)]$
Example

\[ g = y_1 + y_2 y'_3 \]

\[ y_1(c, x) = (c_0 c_1 x_1 + c_0 c'_1 x_2 + c'_0 c_1 x_3) \oplus c_2 \]

\[ G = y_1(c, x) + y_2(c, x) y_3(c, x)' \]
Extended matching modeling

- Model composite functions by ROBDDs.
  - Assume: \( n \)-input cluster and \( m \)-input cell.
  - For each cell input:
    * \( \lceil \log_2 n \rceil \) variables for pin permutation.
    * One variable for input polarity.
  - Total size of \( \mathbf{c} \): \( m(\lceil \log_2 n \rceil + 1) \).

- A match exists if there is at least one value of \( \mathbf{c} \) satisfying \( M(\mathbf{c}) = \forall \mathbf{x} [G(\mathbf{x}, \mathbf{c}) \oplus f(\mathbf{x})] \).
Example

- $g = x'y$, $f = wz'$
- $G(a, b, c, d, w, z) = (c \oplus (za + wa'))'(d \oplus (zb + wb'))$
- $f \oplus G = (wz') \oplus ((c \oplus (za + wa'))'(d \oplus (zb + wb')))$
- $M(a, b, c, d) = ab'c'd' + a'bcd$
Extended matching

- Captures implicitly all possible matches.

- No extra burden when exploiting don’t care sets.

\[- M(c) = \forall x [G(x, c) \oplus f(x) + f_{DC}(x)]\]

- Efficient BDD-based representation.

- Extensions to support multiple-output matching.
Summary

- Library binding is very important.

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

- Algorithmic approach:
  - Pattern-based: fast, but limited.
  - Boolean: more general and efficient.