Technology Mapping

Example:

\[ t_1 = a + bc; \]
\[ t_2 = d + e; \]
\[ t_3 = ab + d; \]
\[ t_4 = t_1 t_2 + fg; \]
\[ t_5 = t_4 h + t_2 t_3; \]
\[ F = t_5'; \]

This shows an unoptimized set of logic equations consisting of 16 literals
Optimized Equations

Using technology independent optimization, these equations are optimized using only 14 literals:

\[
\begin{align*}
t_1 &= d + e; \\
t_2 &= b + h; \\
t_3 &= a t_2 + c; \\
t_4 &= t_1 t_3 + fgh; \\
F &= t_4';
\end{align*}
\]
Optimized Equations

Implement this network using a set of gates which form a library. Each gate has a cost (i.e. its area, delay, etc.)
Technology Mapping

Two approaches:
• Rule-Based [LSS]
• Algorithmic [DAGON, MISII]
  • Represent each function of a network using a set of base functions. This representation is called the subject graph.
  • Typically the base is 2-input NANDs and inverters [MISII].
  • The set should be functionally complete.
- Each gate of the library is likewise represented using the base set. This generates pattern graphs
  • Represent each gate in all possible ways
Subject graph

Subject graph of 2-input NANDs and invertors
Algorithmic Approach

A **cover** is a collection of pattern graphs such that

- every node of the subject graph is contained in one (or more) pattern graphs
- each input required by a pattern graph is actually an output of some other graph (i.e. the inputs of one gate must exists as outputs of other gates.)

For minimum area, the cost of the cover is the sum of the areas of the gates in the cover.

Technology mapping problem: *Find a minimum cost covering of the subject graph by choosing from the collection of pattern graphs for all the gates in the library.*
Subject Graph

\[ F = t_4' \]

\[ t_1 = d + e; \]
\[ t_2 = b + h; \]
\[ t_3 = a t_2 + c; \]
\[ t_4 = t_3 t_3 + f g h; \]
Pattern Graphs for the IWLS Library

- inv(1)
- nand2(2)
- nand3 (3)
- nor(2)
- nor3 (3)
- aoi21 (3)
- oai22 (4)
- xor (5)
Subject graph covering

\[ t_1 = d + e; \]
\[ t_2 = b + h; \]
\[ t_3 = a t_2 + c; \]
\[ t_4 = t_1 t_3 + f g h; \]
\[ F = t_4'; \]

Total cost = 23
Better Covering

\[ t_1 = d + e, \]
\[ t_2 = b + h, \]
\[ t_3 = a t_2 + c, \]
\[ t_4 = t_1 t_3 + f g h, \]
\[ F = t_4'. \]

Total area = 19
Alternate Covering

\[ t_1 = d + e; \]
\[ t_2 = b + h; \]
\[ t_3 = at_2 + c; \]
\[ t_4 = t_1t_3 + fgh; \]
\[ F = t_4'; \]

Total area = 15
Technology mapping using DAG covering

Input
- Technology independent, optimized logic network
- Description of the gates in the library with their cost

Output
- Netlist of gates (from library) which minimizes total cost

General Approach
- Construct a subject DAG for the network
- Represent each gate in the target library by pattern DAG's
- Find an optimal-cost covering of subject DAG using the collection of pattern DAG's
Technology mapping using DAG covering

Complexity of DAG covering:

- NP-hard
- Remains NP-hard even when the nodes have out degree \( \leq 2 \)
- If subject DAG and pattern DAG's are trees, an efficient algorithm exists
DAG covering as a binate covering problem

- Compute all possible matches \( \{m_k\} \) (ellipses in fig.) for each node
- Using a variable \( m_i \) for each match of a pattern graph in the subject graph, \( (m_i = 1 \text{ if match is chosen}) \)
- Write a clause for each node of the subject graph indicating which matches cover this node. Each node has to be covered.
  - e.g., if a subject node is covered by matches \( \{m_2, m_5, m_{10}\} \), then the clause would be \( (m_2 + m_5 + m_{10}) \).
- Repeat for each subject node and take the product over all subject nodes. (CNF)
DAG covering as a binate covering problem

Any satisfying assignment guarantees that all subject nodes are covered, but does not guarantee that other matches chosen create outputs needed as inputs needed for a given match.

Rectify this by adding additional clauses.
DAG covering as a binate covering problem

- Let match $m_i$ have subject nodes $s_{i1},...,s_{in}$ as $n$ inputs. If $m_i$ is chosen, one of the matches that realizes $s_{ij}$ must also be chosen for each input $j$ ($j$ not a primary input).

- Let $S_{ij}$ be the disjunctive expression in the variables $m_k$ giving the possible matches which realize $s_{ij}$ as an output node. Selecting match $m_i$ implies satisfying each of the expressions $S_{ij}$ for $j = 1 \ldots n$.

This can be written

$$(m_i \Rightarrow (s_{i1} \ldots s_{in})) \iff (\overline{m}_i + (s_{i1} \ldots s_{in})) \iff ((\overline{m}_i + s_{i1}) \ldots (\overline{m}_i + s_{in}))$$
DAG covering as a binate covering problem

- Also, one of the matches for each primary output of the circuit must be selected.
- An assignment of values to variables $m_i$ that satisfies the above covering expression is a legal graph cover
- For area optimization, each match $m_i$ has a cost $c_i$ that is the area of the gate the match represents.
- The goal is a satisfying assignment with the least total cost.
  - Find a least-cost prime:
    - if a variable $m_i = 0$ its cost is 0, else its cost in $c_i$
    - $m_i = 0$ means that match $i$ is not chosen
Binate Covering

This problem is more general than unate-covering for two-level minimization because
- variables are present in the covering expression in both their true and complemented forms.

The covering expression is a binate logic function, and the problem is referred to as the \textit{binate-covering problem}.

This problem has appeared before:
- As state minimization of incompletely specified finite-state machines [Grasselli-65]
- As a problem in the design of optimal three level NAND-gate networks [Gimpel-67]
- In the optimal phase assignment problem [Rudell-86]
Binate Covering: Example

<table>
<thead>
<tr>
<th>Gate</th>
<th>Cost</th>
<th>Inputs</th>
<th>Produces</th>
<th>Covers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>inv</td>
<td>1</td>
<td>$b$</td>
<td>$g_1$</td>
</tr>
<tr>
<td>$m_2$</td>
<td>inv</td>
<td>1</td>
<td>$a$</td>
<td>$g_2$</td>
</tr>
<tr>
<td>$m_3$</td>
<td>nand2</td>
<td>2</td>
<td>$g_1,g_2$</td>
<td>$g_3$</td>
</tr>
<tr>
<td>$m_4$</td>
<td>nand2</td>
<td>2</td>
<td>$a, b$</td>
<td>$g_4$</td>
</tr>
<tr>
<td>$m_5$</td>
<td>nand2</td>
<td>2</td>
<td>$g_3,g_4$</td>
<td>$g_5$</td>
</tr>
<tr>
<td>$m_6$</td>
<td>inv</td>
<td>1</td>
<td>$g_4$</td>
<td>$g_6$</td>
</tr>
<tr>
<td>$m_7$</td>
<td>nand2</td>
<td>2</td>
<td>$g_6,c$</td>
<td>$g_7$</td>
</tr>
<tr>
<td>$m_8$</td>
<td>inv</td>
<td>1</td>
<td>$g_7$</td>
<td>$g_8$</td>
</tr>
<tr>
<td>$m_9$</td>
<td>nand2</td>
<td>2</td>
<td>$g_8,d$</td>
<td>$g_9$</td>
</tr>
<tr>
<td>$m_{10}$</td>
<td>nand3</td>
<td>3</td>
<td>$g_6,c, d$</td>
<td>$g_9$</td>
</tr>
<tr>
<td>$m_{11}$</td>
<td>nand3</td>
<td>3</td>
<td>$a,b,c$</td>
<td>$g_7$</td>
</tr>
<tr>
<td>$m_{12}$</td>
<td>xnor2</td>
<td>5</td>
<td>$a,b$</td>
<td>$g_5$</td>
</tr>
<tr>
<td>$m_{13}$</td>
<td>nand4</td>
<td>4</td>
<td>$a,b,c,d$</td>
<td>$g_9$</td>
</tr>
<tr>
<td>$m_{14}$</td>
<td>oai21</td>
<td>3</td>
<td>$a,b,g_4$</td>
<td>$g_5$</td>
</tr>
</tbody>
</table>
Binate Covering: Example

Generate constraints that each node $g_i$ be covered by some match.

$$(m_1 + m_{12} + m_{14})(m_2 + m_{12} + m_{14})(m_3 + m_{12} + m_{14})$$
$$(m_4 + m_{11} + m_{12} + m_{13})(m_5 + m_{12} + m_{13})$$
$$(m_6 + m_{11} + m_{13})(m_7 + m_{10} + m_{11} + m_{13})$$
$$(m_8 + m_{10} + m_{13})(m_9 + m_{10} + m_{13})$$

To ensure that a cover leads to a valid circuit, extra clauses are generated.

- For example, selecting $m_3$ requires that
  - a match be chosen which produces $g_2$ as an output, and
  - a match be chosen which produces $g_1$ as an output.

The only match which produces $g_1$ is $m_1$, and the only match which produces $g_2$ is $m_2$
Binate Covering: Example

The primary output nodes $g_5$ and $g_9$ must be realized as an output of some match.

- The matches which realize $g_5$ as an output are $m_5, m_{12}, m_{14}$.
- The matches which realize $g_9$ as an output are $m_9, m_{10}, m_{13}$

- **Note:**
  - A match which requires a primary input as an input is satisfied trivially.
  - Matches $m_1, m_2, m_4, m_{11}, m_{12}, m_{13}$ are driven only by primary inputs and do not require additional clauses.
Binate Covering: Example

- Finally, we get
  \[(\bar{m}_3 + m_1)(\bar{m}_3 + \bar{m}_2)(m_3 + \bar{m}_5)(\bar{m}_5 + m_4)(m_6 + m_4)
  (m_7 + m_6)(m_8 + m_7)(m_8 + m_9)(m_{10} + m_6)
  (m_{14} + m_4)(m_5 + m_{12} + m_{14})(m_9 + m_{10} + m_{13})\]
- The covering expression has 58 implicants
- The least cost prime implicant is
  \[\bar{m}_3 \bar{m}_5 \bar{m}_6 \bar{m}_7 \bar{m}_8 \bar{m}_9 \bar{m}_{10} \bar{m}_{12} \bar{m}_{13} \bar{m}_{14}\]
- This uses two gates for a cost of nine gate units. This corresponds to a cover which selects matches \(m_{12}\) (xor2) and \(m_{13}\) (nand4).
Binate Covering: Example

<table>
<thead>
<tr>
<th>Gate</th>
<th>Cost</th>
<th>Inputs</th>
<th>Produces</th>
<th>Covers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>inv 1</td>
<td>$b$</td>
<td>$g_1$</td>
<td>$g_1$</td>
</tr>
<tr>
<td>$m_2$</td>
<td>inv 1</td>
<td>$a$</td>
<td>$g_2$</td>
<td>$g_2$</td>
</tr>
<tr>
<td>$m_3$</td>
<td>nand2 2</td>
<td>$g_1, g_2$</td>
<td>$g_3$</td>
<td>$g_3$</td>
</tr>
<tr>
<td>$m_4$</td>
<td>nand2 2</td>
<td>$a, b$</td>
<td>$g_4$</td>
<td>$g_4$</td>
</tr>
<tr>
<td>$m_5$</td>
<td>nand2 2</td>
<td>$g_3, g_4$</td>
<td>$g_5$</td>
<td>$g_5$</td>
</tr>
<tr>
<td>$m_6$</td>
<td>inv 1</td>
<td>$g_4$</td>
<td>$g_6$</td>
<td>$g_6$</td>
</tr>
<tr>
<td>$m_7$</td>
<td>nand2 2</td>
<td>$g_6, c$</td>
<td>$g_7$</td>
<td>$g_7$</td>
</tr>
<tr>
<td>$m_8$</td>
<td>inv 1</td>
<td>$g_7$</td>
<td>$g_8$</td>
<td>$g_8$</td>
</tr>
<tr>
<td>$m_9$</td>
<td>nand2 2</td>
<td>$g_8, d$</td>
<td>$g_9$</td>
<td>$g_9$</td>
</tr>
<tr>
<td>$m_{10}$</td>
<td>nand3 3</td>
<td>$g_6, c, d$</td>
<td>$g_9$</td>
<td>$g_7, g_8, g_9$</td>
</tr>
<tr>
<td>$m_{11}$</td>
<td>nand3 3</td>
<td>$a, b, c$</td>
<td>$g_7$</td>
<td>$g_4, g_6, g_7$</td>
</tr>
<tr>
<td>$m_{12}$</td>
<td>xnor2 5</td>
<td>$a, b$</td>
<td>$g_5$</td>
<td>$g_4, g_5, g_6, g_7, g_8, g_9$</td>
</tr>
<tr>
<td>$m_{13}$</td>
<td>nand4 4</td>
<td>$a, b, c, d$</td>
<td>$g_9$</td>
<td>$g_4, g_6, g_7, g_8, g_9$</td>
</tr>
<tr>
<td>$m_{14}$</td>
<td>oai21 3</td>
<td>$a, b, g_4$</td>
<td>$g_5$</td>
<td>$g_4, g_2, g_3, g_5$</td>
</tr>
</tbody>
</table>

Note that the node $g_4$ is covered by both matches
Complexity of DAG covering

More general than unate covering
- Finding least cost prime of a \textit{binate} function.
  - Even finding a feasible solution is NP-complete (SAT).
  - For unate covering, finding a feasible solution is easy.
- DAG-covering: covering + implication constraints

\textit{Given a subject graph}, the binate covering provides the exact solution to the technology-mapping problem.

- However, better results may be obtained with a different initial decomposition into 2-input NANDS and inverters

\textbf{Methods to solve the binate covering formulation:}
- Branch and bound [Thelen]
- BDD-based [Lin and Somenzi]

Even for moderate-size networks, these are expensive.
Optimal Tree Covering by Trees

- If the subject DAG and primitive DAG’s are trees, then an efficient algorithm to find the best cover exists
- Based on dynamic programming
- First proposed for optimal code generation in a compiler
Optimal Tree Covering by Trees

Partition subject graph into forest of trees
Cover each tree optimally using dynamic programming

Given:
- Subject trees (networks to be mapped)
- Forest of patterns (gate library)

- Consider a node $N$ of a subject tree
- Recursive Assumption: for all children of $N$, a best cost match (which implements the node) is known
- Cost of a leaf of the tree is 0.
- Compute cost of each pattern tree which matches at $N$, $\text{Cost} = \text{SUM of best costs of implementing each input of pattern plus the cost of the pattern}$
- Choose least cost matching pattern for implementing $N$
Optimum Area Algorithm

Optimal_area_cover(node) {
    /* find optimal cover for all nodes below "node" */
    foreach input of node {
        optimal_area_cover(input); /* thus satisfies recursive assumption */
    }
    /* Using these, find the best cover at "node" */
    node→area = INFINITY;
    node→match = 0;
    foreach match at node {
        area = match→area;
        foreach pin of match {
            area = area + pin→area;
        }
        if (area < node→area) {
            node→area = area;
            node→match = match;
        }
    }
}
Tree Covering in Action

library

nand2 = 3
inv = 2
nand3 = 4
nand4 = 5
and2 = 4
aio21 = 4
oai21 = 4
Complexity of Tree Covering

- Complexity is controlled by finding all sub-trees of the subject graph which are isomorphic to a pattern tree.
- Linear complexity in both size of subject tree and size of collection of pattern trees
Partitioning the subject DAG into trees

Trivial partition: break the graph at all multiple-fanout points
- leads to no “duplication” or “overlap” of patterns
- drawback - sometimes results in many of small trees

Leads to 3 trees
Partitioning the subject DAG into trees

Single-cone partition:
- from a single output, form a large tree back to the primary inputs;
- map successive outputs until they hit match output formed from mapping previous primary outputs.
  - Duplicates some logic (where trees overlap)
  - Produces much larger trees, potentially even better area results
Logic Decomposition and Technology Mapping
Lehman-Watanabe Method

Common Approach:
• Phase 1: Technology independent optimization
  - commit to a particular Boolean network
  - algebraic decomposition used
• Phase 2: AND2/INV decomposition
  - commit to a particular decomposition of a general Boolean network using 2-input ANDs and inverters
• Phase 3: Technology mapping (tree-mapping)
Logic Decomposition and Technology Mapping - Lehman-Watanabe Method

Drawbacks:

Procedures in each phase are disconnected:
- Phase 1 and Phase 2 make critical decisions without knowing much about constraints and library
- Phase 3 knows about constraints and library, but solution space is restricted by decisions made earlier
Lehman-Watanabe Technology Mapping

Incorporate technology independent procedures (Phase 1 and Phase 2) into technology mapping

Key Idea:
- Efficiently encode a set of AND2/INV decompositions into a single structure called a mapping graph
- Apply a modified tree-based technology mapper while dynamically performing algebraic logic decomposition on the mapping graph
Outline

• **Mapping Graph**
  - Encodes a set of AND2/INV decompositions

• **Tree-mapping on a mapping graph: graph-mapping**

• **Δ-mapping:**
  - without dynamic logic decomposition
  - solution space: Phase 3 + Phase 2

• **Δ-mapping:**
  - with dynamic logic decomposition
  - solution space: Phase 3 + Phase 2 + Algebraic decomposition (Phase 1)

• **Experimental results**
End of lecture 18
A set of AND2/INV Decompositions

\( f = abc \) can be represented in various ways

\[ \begin{array}{c}
\text{Diagram 1} \\
\text{Diagram 2} \\
\text{Diagram 3}
\end{array} \]
A set of AND2/INV Decompositions

Combine them using a choice node
Encoding AND2/INV Decompositions

These decompositions can be represented more compactly as:

This representation encodes even more decompositions, e.g.
Mapping Graph

A Boolean network containing 4 modifications:
- Choice node: choices on different decompositions
- Cyclic: functions written in terms of each other, e.g. inverter chain with an arbitrary length
- Reduced: No two choice nodes with same function. No two AND2s with same fanin. (like BDD node sharing)
- Ugates: just for efficient implementation - do not explicitly represent choice nodes and inverters

For χ η τ benchmark (MCNC’91), there are $2.2 \times 10^{93}$ AND2/INV decompositions. All are encoded with only 400 ugates containing 599 AND2s in total.
Tree-mapping on a Mapping Graph

Graph-Mapping on Trees**: Apply dynamic programming from primary inputs:
- find matches at each AND2 and INV, and
- retain the cost of a best cover at each node
  - a match may contain choice nodes
  - the cost at a choice node is the minimum of fanin costs
  - fixed-point iteration on each cycle, until costs of all the nodes in the cycle become stable

Run-time is typically linear in the size of the mapping graph

** mapping graph may not be a tree, but any multiple fanout node just represents several copies of same function.
Example: Tree-mapping

Delay: best choice if c is later than a and b.

subject graph

library pattern graph
Graph-Mapping: Theory

Graph-mapping( \( \mu \) ) = \( \min_{\theta \in \mu} \) (tree-mapping(\( \theta \)))

\( \mu \): mapping graph
\( \theta \): AND2/INV decomposition encoded in \( \mu \)

- Graph-mapping finds an optimal tree implementation for each primary output over all AND2/INV decompositions encoded in \( \mu \)
- Graph-mapping is as powerful as applying tree-mapping exhaustively, but is typically exponentially faster
Λ-Mapping

Given a Boolean network $\eta$,
• Generate a mapping graph $\mu$:
  • For each node of $\eta$,
    • encode all AND2 decompositions for each product term
      Ex: $abc \Rightarrow 3$ AND2 decompositions: $a(bc), c(ab), b(ca)$
    • encode all AND2/INV decompositions for the sum term
      Ex: $p+q+r \Rightarrow 3$ AND2/INV decompositions:
        $p+(q+r), r+(p+q), q+(r+p)$. 
  • Apply graph-mapping on $\mu$

In practice, $\eta$ is pre-processed so
• each node has at most 10 product terms and
• each term has at most 10 literals
**Λ-Mapping: Theory**

For the mapping graph $\mu$ generated for a Boolean network $\eta$, let

- $L_\eta$ be the set of AND2/INV decompositions encoded in $\mu$
- $\Lambda_\eta$ be the closure of the set of AND2/INV decompositions of $\eta$ under the associative and inverter transformations:

![Associative transform](image)

![Inverter transform](image)

**Theorem:** $\Lambda_\eta = L_\eta$
Dynamic Logic Decomposition

During graph-mapping, dynamically modify the mapping graph:
- find D-patterns and add F-patterns

D-pattern: $ab + ac$

F-pattern: $a(b + c)$
Dynamic Logic Decomposition

Note: Adding F-patterns may introduce new D-patterns which may imply new F-patterns.
Δ-Mapping

Given a boolean network $\eta$, generate a mapping graph $\mu$

Iteratively apply graph mapping on $\mu$, while performing dynamic logic decomposition until nothing changes in $\mu$
- Before finding matches at an AND2 in $\mu$, check if D-pattern matches at the AND2. If so, add the corresponding F-pattern

- In practice, terminate the procedure when a feasible solution is found
Δ-Mapping

For the mapping graph μ generated for a Boolean network η, let

- \( D_\eta \) be the set of AND2/INV decompositions encoded in the resulting mapping graph.
- \( \Delta_\eta \) be the closure of \( \Lambda_\eta \) under the distributive transformation:

\[
\begin{array}{c}
\text{b} \\
\text{a} \\
\text{c}
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{b} \\
\text{a} \\
\text{c}
\end{array}
\]

Theorem: \( \Delta_\eta = D_\eta \)
$\Delta$-Mapping

Theorem: If

- $\eta'$ is an arbitrary Boolean network obtained from $\eta$ by algebraic decomposition.
- $\theta$ is an arbitrary AND2/INV decomposition of $\eta'$

then $\theta \in D_\eta$

The resulting mapping graph encodes all the AND2/INV decompositions of all algebraic decompositions of $\eta$. 
Solution Space captured by Procedures

Phase 1: arbitrary algebraic decomposition

Phase 2: arbitrary AND2/INV decomposition

Mapping graph \( \mu \) associative transform

Dynamic decomposition distributive transform

\( \Lambda \)-mapping captures all AND2/INV decompositions of \( \eta \):

Phase 2 (subject graph generation) is subsumed

\( \Delta \)-mapping captures all algebraic decompositions:

Phase 2 and Phase 1 are subsumed.
Experiments

Examples: MCNC’91 combinational benchmarks
Library: From DEC ALPHA microprocessor
Cost: Delay (fastest possible implementation)

- Apply four different technology independent SIS scripts on the initial Boolean network.
- Encode each of the four optimized boolean network into a single mapping graph $\mu$. (Note: here we encode even more decompositions into $\mu$)
- Apply graph mapping on $\mu$ ($\Lambda$-mapping).
- Collapse the most timing critical region of a mapped network, and encode it in $\mu$.
- Apply graph-mapping with dynamic logic decomposition ($\Delta$-mapping)
## Experimental results

<table>
<thead>
<tr>
<th>Name</th>
<th>SIS-1.2</th>
<th></th>
<th></th>
<th>SynFul</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delay</td>
<td>Area</td>
<td></td>
<td>Delay</td>
<td>Area</td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td>12.0</td>
<td>167.7</td>
<td>9.5</td>
<td>192.0</td>
<td>8.0</td>
<td>193.3</td>
</tr>
<tr>
<td>cm82a</td>
<td>13.2</td>
<td>86.0</td>
<td>10.5</td>
<td>58.3</td>
<td>7.0</td>
<td>78.7</td>
</tr>
<tr>
<td>pcle</td>
<td>15.0</td>
<td>255.7</td>
<td>12.0</td>
<td>156.7</td>
<td>10.5</td>
<td>176.7</td>
</tr>
<tr>
<td>pm1</td>
<td>13.5</td>
<td>135.0</td>
<td>9.8</td>
<td>135.0</td>
<td>9.2</td>
<td>131.3</td>
</tr>
<tr>
<td>x2</td>
<td>17.7</td>
<td>151.0</td>
<td>9.8</td>
<td>109.7</td>
<td>9.8</td>
<td>109.7</td>
</tr>
<tr>
<td>cm163a</td>
<td>15.5</td>
<td>130.7</td>
<td>12.0</td>
<td>111.3</td>
<td>9.7</td>
<td>121.0</td>
</tr>
<tr>
<td>comp</td>
<td>32.5</td>
<td>401.3</td>
<td>26.0</td>
<td>502.7</td>
<td>21.2</td>
<td>454.3</td>
</tr>
<tr>
<td>cu</td>
<td>16.0</td>
<td>145.0</td>
<td>11.8</td>
<td>147.3</td>
<td>9.8</td>
<td>149.0</td>
</tr>
<tr>
<td>il</td>
<td>12.7</td>
<td>183.0</td>
<td>11.5</td>
<td>132.3</td>
<td>9.8</td>
<td>111.0</td>
</tr>
<tr>
<td>cm151a</td>
<td>15.0</td>
<td>77.0</td>
<td>10.5</td>
<td>66.3</td>
<td>9.0</td>
<td>88.3</td>
</tr>
<tr>
<td>cht</td>
<td>9.2</td>
<td>459.7</td>
<td>9.2</td>
<td>320.7</td>
<td>8.2</td>
<td>323.7</td>
</tr>
<tr>
<td>ttt2</td>
<td>21.5</td>
<td>530.7</td>
<td>15.3</td>
<td>576.0</td>
<td>12.7</td>
<td>606.7</td>
</tr>
<tr>
<td>cm162a</td>
<td>12.7</td>
<td>164.7</td>
<td>12.0</td>
<td>111.3</td>
<td>9.7</td>
<td>122.7</td>
</tr>
</tbody>
</table>

- **Delay unit:** minimum input-output pin delay over all library gates
- **Area unit:** minimum gate-area over all library gates
- **SynFul’s CPU time:** ranged from 10 to 300 seconds on VAXstation 6000
Conclusions

• Logic decomposition during technology mapping
  - Efficiently encode a set on AND2/INV decompositions
  - Dynamically perform logic decomposition

• Two mapping procedures
  - \(\Lambda\)-mapping: optimal over all AND2/INV decompositions (associative rule)
  - \(\Delta\)-mapping: optimal over all algebraic decompositions (distributive rule)

• Was implemented and used for commercial design projects (in DEC/Compac alpha)

• Extended for sequential circuits:
  - considers all retiming possibilities (implicitly) and algebraic factors across latches
Tree-Covering Approximation

Principle of Optimality: An optimum sequence of decisions has the property that whatever the initial state and first decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.
Tree-Covering Approximation

**Dynamic Programming:** (Works for special types of graphs (e.g. trees)).

- An optimum solution can be obtained by computing the optimum solutions in order for each subgraph (e.g. sub-trees).
- The optimum decision at a node is composed of the optimum at the node plus the optimum costs of the subgraphs at the node. (basically can be proved by induction on the ordering)

**Proposition:** The minimum area cover for a tree $T$ can be derived from the minimum area covers for every node below the root of $T$