## Logic Synthesis and Verification

## Technology Mapping

Jie－Hong Roland Jiang江介宏

Department of Electrical Engineering National Taiwan University

Fall 2014

Reading：
Logic Synthesis in a Nutshell Section 4

Technology Independent Optimization

## $\square$ Example

$t_{1}=a+b c$
$t_{2}=d+e$
$t_{3}=a b+d$
$t_{4}=t_{1} t_{2}+f g$
$t_{5}=t_{4} h+t_{2} t_{3}$
$\mathrm{F}=\mathrm{t}_{5}$ ，


An unoptimized set of logic equations consisting of 17 literals

Technology Independent Optimization
$\square$ Example（cont＇d）

$$
\begin{aligned}
& t_{1}=d+e \\
& t_{2}=b+h \\
& t_{3}=a t_{2}+c \\
& t_{4}=t_{1} t_{3}+f g h \\
& F=t_{4}^{\prime}
\end{aligned}
$$



An optimized set of logic equations consisting of 13 literals

## Technology Mapping

$\square$ Implement an optimized Boolean network using a set of pre-designed and pre-characterized gates from a library

- Each gate has a cost (e.g. area, delay, power, etc.)
library of primitive gates



## Technology Mapping

Two approaches:

1. Rule based: LSS
2. Algorithmic: DAGON
$\square$ Represent the netlist to be mapped in terms of a selected set of base functions, e.g., \{NAND2, INV\}

- Base functions from a functionally complete set
- Such a netlist is called the subject graph
$\square$ Each gate in the library is likewise represented using the base functions
- Represent each gate in all possible ways
" This generates pattern graphs


## Algorithmic Technology Mapping

## Subject Graph

$\square$ A cover is a collection of pattern graphs such that
$\square$ 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 pattern graph (i.e. the inputs of one gate must exist as outputs of other gates)
$\square$ For area minimization, the cost of the cover is the sum of the areas of the gates in the coverTechnology 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
$\square$ Example
$t_{1}=d+e$
$t_{2}=b+h$
$t_{3}=a t_{2}+c$
$\mathrm{t}_{4}=\mathrm{t}_{1} \mathrm{t}_{3}+\mathrm{fgh}$
$\mathrm{F}=\mathrm{t}_{4}$,



## Pattern Graphs

$\square$ Example

- (IWLS library)






## Subject Graph Covering (1)



10

Subject Graph Covering (2)
$\square$ Example
$t_{1}=d+e$
$t_{2}=b+h$
$t_{3}=a t_{2}+c$
$t_{4}=t_{1} t_{3}+f g h$

Total cost $=18$


## Subject Graph Covering (3)

$\square$ Example

Total cost $=15$
$t_{1}=d+e$
$t_{2}=b+h$
$t_{4}=t_{1} t_{3}+f g h$


## DAG Covering

I nput:- Logic network after technology independent optimization
- A library of gates with their costsOutput:
A netlist of gates (from library) which minimizes total cost
$\square$ General Approach:
- Construct a subject DAG (directed acyclic graph) 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


## DAG Covering

Complexity- NP-hard
$\square$ Remains NP-hard even when the nodes have out-degree $\leq 2$
■ If subject DAG and pattern DAG's are trees, efficient algorithms exist


## DAG Covering Binate Covering Approach

$\square$ Compute all possible matches $\left\{m_{k}\right\}$ of pattern graphs for each node in the subject graph

- Using a variable $m_{i}$ for each match of a pattern graph in the subject graph, ( $m_{i}=1$ if match is chosen)
$\square$ 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 $\left\{m_{2}, m_{5}\right.$, $\left.m_{10}\right\}$, then the clause would be $\left(m_{2}+m_{5}+m_{10}\right)$
$\square$ Repeat for each subject node and take the product over all subject nodes (CNF)



## DAG Covering <br> Binate Covering Approach

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

$\square$ Resolve this problem by adding additional clauses

## DAG Covering <br> Binate Covering Approach

$\square$ Let match $m_{j}$ have subject nodes $v_{i 1}, \ldots, v_{i n}$ as its $n$ inputs. If $m_{i}$ is chosen, one of the matches that realizes $v_{i j}$ must also be chosen for each input $j$ (if j not a primary input).
$\square$ Let $S_{i j}$ be the disjunctive expression in the variables $m_{k}$ giving the possible matches which realize $v_{i j}$ as an output node. Selecting match $m_{i}$ implies satisfying each of the expressions $S_{i j}$ for $\mathrm{j}=1 \ldots \mathrm{n}$. This can be written
$\left(m_{i} \Rightarrow\left(S_{i 1} \ldots S_{i n}\right)\right) \Leftrightarrow\left(\bar{m}_{i}+\left(S_{i 1} \ldots S_{i n}\right)\right) \Leftrightarrow\left(\left(\bar{m}_{i}+S_{i 1}\right) \ldots\left(\bar{m}_{i}+S_{i n}\right)\right)$

## DAG Covering

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

## DAG Covering

Binate Covering Approach
$\square$ Binate covering is more general than unate covering
$\square$ Unlike unate covering, variables are present in both their true and complemented forms in the covering expression

- The covering expression is a binate function, and the problem is referred to as the binatecovering problem

DAG Covering
Binate Covering Approach

| $\square$ Example | Match | Gate | Cost | Inputs | Root | Covers |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | $\mathrm{m}_{1}$ | inv | 1 | b | $\mathrm{g}_{1}$ | $\mathrm{g}_{1}$ |
|  | $\mathrm{m}_{2}$ | inv | 1 | a | $\mathrm{g}_{2}$ | $\mathrm{g}_{2}$ |
|  | $\mathrm{m}_{3}$ | nand2 | 2 | $\mathrm{g}_{1}, \mathrm{~g}_{2}$ | $\mathrm{g}_{3}$ | $\mathrm{g}_{3}$ |
|  | $\mathrm{m}_{4}$ | nand2 | 2 | a, b | $\mathrm{g}_{4}$ | $\mathrm{g}_{4}$ |
|  | $\mathrm{m}_{5}$ | nand2 | 2 | $\mathrm{g}_{3}, \mathrm{~g}_{4}$ | $\mathrm{g}_{5}$ | $\mathrm{g}_{5}$ |
|  | $\mathrm{m}_{6}$ | inv | 1 | $\mathrm{g}_{4}$ | $\mathrm{g}_{6}$ | $\mathrm{g}_{6}$ |
|  | $\mathrm{m}_{7}$ | nand2 | 2 | $\mathrm{g}_{6}, \mathrm{c}$ | $\mathrm{g}_{7}$ | $\mathrm{g}_{7}$ |
|  | $\mathrm{m}_{8}$ | inv | 1 | $\mathrm{g}_{7}$ | $\mathrm{g}_{8}$ | $\mathrm{g}_{8}$ |
|  | $\mathrm{m}_{9}$ | nand2 | 2 | $\mathrm{g}_{8}$, d | $\mathrm{g}_{9}$ | $\mathrm{g}_{9}$ |
|  | $\mathrm{m}_{10}$ | nand3 | 3 | $\mathrm{g}_{6}, \mathrm{c}, \mathrm{d}$ | $\mathrm{g}_{9}$ | $\mathrm{g}_{7}, \mathrm{~g}_{8}, \mathrm{~g}_{9}$ |
|  | $\mathrm{m}_{11}$ | nand3 | 3 | a,b,c | $\mathrm{g}_{7}$ | $\mathrm{g}_{4}, \mathrm{~g}_{6}, \mathrm{~g}_{7}$ |
|  | $\mathrm{m}_{12}$ | xnor2 | 5 | a,b | $\mathrm{g}_{5}$ | $\mathrm{g}_{1}, \mathrm{~g}_{2}, g_{3}, \mathrm{~g}_{4}, \mathrm{~g}_{5}$ |
|  | $\mathrm{m}_{13}$ | nand4 | 4 | a, b, c, d | $\mathrm{g}_{9}$ | $\mathrm{g}_{4}, \mathrm{~g}_{6}, \mathrm{~g}_{7}, \mathrm{~g}_{8}, \mathrm{~g}_{9}$ |
|  | $\mathrm{m}_{14}$ | oai21 | 3 | $a, b, g_{4}$ | $\mathrm{g}_{5}$ | $\mathrm{g}_{1}, \mathrm{~g}_{2}, \mathrm{~g}_{3}, \mathrm{~g}_{5}$ |

## DAG Covering <br> Binate Covering Approach

## DAG Covering

Binate Covering Approach

## $\square$ Example (cont'd)

- Generate constraints that each node $g_{i}$ be covered by some match


## $\square$ Example (cont'd)

$\left(m_{1}+m_{12}+m_{14}\right)\left(m_{2}+m_{12}+m_{14}\right)\left(m_{3}+m_{12}+m_{14}\right)$
$\square$ 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 $m_{5}$, $\mathrm{m}_{12}, \mathrm{~m}_{14}$

- The matches which realize $g_{9}$ as an output are $m_{9}$, $\mathrm{m}_{10}, \mathrm{~m}_{13}$
- Note:
$\square$ 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


## DAG Covering <br> Binate Covering Approach

$\square$ Example (cont'd)

- Finally, we get
$\left(\bar{m}_{3}+m_{1}\right)\left(\underline{m}_{3}+m_{2}\right)\left(m_{3}+\bar{m}_{5}\right)\left(\bar{m}_{5}+m_{4}\right)\left(\bar{m}_{6}+m_{4}\right)$
$\left(m_{7}+m_{6}\right)\left(m_{8}+m_{7}\right)\left(m_{8}+m_{9}\right)\left(m_{10}+m_{6}\right)$
$\left(m_{14}+m_{4}\right)\left(m_{5}+m_{12}+m_{14}\right)\left(m_{9}+m_{10}+m_{13}\right)$
- The covering expression has 58 implicants
- The least cost prime implicant is

$$
\mathrm{m}_{3} \mathrm{~m}_{5} \mathrm{~m}_{6} \mathrm{~m}_{7} \mathrm{~m}_{8} \mathrm{~m}_{9} \mathrm{~m}_{10} \mathrm{~m}_{12} \mathrm{~m}_{13} \overline{\mathrm{~m}}_{14}
$$

- This uses two gates for a cost of 9 gate units. This corresponds to a cover which selects matches $\mathrm{m}_{12}$ (xor2) and $\mathrm{m}_{13}$ (nand4).


## DAG Covering <br> Binate Covering Approach

ㅁ Example (cont'd)
$m_{3} \bar{m}_{5} \bar{m}_{6} \bar{m}_{7} \bar{m}_{8} \bar{m}_{9} \bar{m}_{10} m_{12} m_{13} \bar{m}_{14}$


Note: $g_{4}$ is covered by both matches

## DAG Covering <br> Binate Covering Approach

## $\square$ Complexity

■ DAG-covering: covering + implication constraints

- More general than unate covering
$\square$ Finding least cost prime of a binate function
- Even finding a feasible solution is NP-complete (SAT)
- For unate covering, finding a feasible solution is easy
- 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
- Methods to solve the binate covering formulation: $\square$ Branch and bound, BDD-based $\square$ Expensive even for moderate-size networks


## Tree Covering

$\square$ When the subject graph and pattern graphs are trees, an efficient algorithm to find the best cover exists
$\square$ Solvable with dynamic programming

## Tree Covering

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

- Given:
$\square$ Subject trees (networks to be mapped)
$\square$ Forest of patterns (gate library)
- For each node $N$ of a subject tree
- Recursive Assumption: for all children of N , a best cost match (which implements the node) is known
- Compute cost of each pattern tree which matches at N , Cost $=$ SUM of best costs of implementing each
input of pattern plus the cost of the pattern - Cost of a leaf of the tree is 0
$\square$ Choose least cost matching pattern for implementing $N$


## Tree Covering

```
\square Algorithm OPTIMAL_AREA_COVER(node) {
    foreach input of node {
        OPTIMAL_AREA_COVER(input);//satisfies recur. assumption
    |
    // Using these, find the best cover at node
    node->area = INFINITY;
    node->match = 0
    foreach match at node {
        area = match }->\mathrm{ area;
            foreach pin of match {
            area = area + pin->area;
            }
            if (area < node->area) {
            node->area = area;
            node }->\mathrm{ match = match
        }
}
}
```


## Tree Covering

- Example

Library: nand2 $=3$ inv $=2$ nand3 $=4$ nand $4=5$ $\begin{aligned} & \text { and2 } \\ & \text { aoi21 }\end{aligned}=4$ aoi21 $=4$
oai21 $=4$

nand4


## Tree Covering

$\square$ Complexity
■ Complexity is controlled by finding all subtrees 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


## Tree Covering

$\square$ Partition subject DAG into trees

- Trivial partition: break the graph at all multiple-fanout points
$\square$ no duplication or overlap in the resulting trees $\square$ drawback - sometimes results in many small trees



## Tree Covering

$\square$ Partition 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 better area results



## Min-Delay Technology Mapping

- For trees:
- identical to min-area covering
- use optimal delay values within the dynamic programming paradigm
$\square$ For DAGs:
- if delay does not depend on number of fanouts: use dynamic programming as presented for trees
- leads to optimal solution in polynomial time $\square$ Assume logic replication is okay
$\square$ Combined objective
- e.g. apply delay as first criteria, then area as second
- combine with static timing analysis to focus on critical paths


## Decomposition and Technology <br> Mapping

## Common Approach:

■ Phase 1: Technology independent optimization $\square$ commit to a particular Boolean network $\square$ 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)

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


## Combined Decomposition and Technology Mapping

$\square$ Incorporate technology independent procedures
(Phase 1 and Phase 2) into technology mappingLehman-Watanabe Algorithm: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


## Combined Decomposition and Technology Mapping

Outline

- Mapping Graph
-Encodes a set of AND2/INV decompositions
- Tree-mapping on a mapping graph: graph-
mapping
- $\Lambda$-mapping:
$\square$ without dynamic logic decomposition
■solution space: Phase 3 + Phase 2
$\square \Delta$-mapping:
$\square$ with dynamic logic decomposition
$\square$ solution space: Phase $3+$ Phase $2+$ Algebraic decomposition (Phase 1)

Combined Decomposition and Technology Mapping
$\square$ AND2/INV decomposition
E.g., $\mathrm{f}=\mathrm{abc}$ can be represented in various ways


Combined Decomposition and Technology Mapping
-Combine different AND2/INV
decompositions with a choice node


Combined Decomposition and Technology Mapping

- The previous AND2/INV decompositions can be represented more compactly as:
This representation encodes even more decompositions, e.g.,


Combined Decomposition and Technology Mapping

- Mapping graph is a Boolean network containing the following our modifications.
- Choice node: choices on different
decompositions
- Cyclic: functions written in terms of each other, e.g. inverter chain
with an arbitrary length
mith arbirary leng
with same function. No two AND2s with same fanin.
- Ugates: just for efficient implementation - do not explicitly represent choice nodes and inverters
For CHT
- For CHT benchmark (MCNC'91), there are are encoded with only 40 Uugatites
containing 599 AND2s in total.


## Combined Decomposition and Technology Mapping

$\square$ 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
$\square$ 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.


## Combined Decomposition and Technology Mapping

## - Example

- Graph mapping on trees for min delay
$\square$ best choice if $c$ is later than $a$ and $b$.



## Combined Decomposition and Technology Mapping

$\square$ Graph mapping
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


## Combined Decomposition and Technology Mapping

$\square$ - -mapping
Given a Boolean network $\eta$,

- Generate a mapping graph $\mu$ :

■ For each node of $\eta$,
-encode all AND2 decompositions for each product term

- E.g., abc $\Rightarrow 3$ AND2 decompositions: $a(b c), c(a b), b(c a)$

Dencode all AND2/INV decompositions for the sum term

- E.g., $p+q+r \Rightarrow 3$ AND2/INV decompositions:
$p+(q+r), r+(p+q), q+(r+p)$
QIn practice, $\eta$ is preprocessed so each node has at most 10 product terms and each term has at most 10 literals

■ Apply graph-mapping on $\mu$

## Combined Decomposition and Technology Mapping

## Combined Decomposition and Technology Mapping

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

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

$\square$ Theorem: $\Lambda \eta=L \eta$


## -Dynamic logic decomposition

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



Combined Decomposition and
Technology Mapping
$\square$ Dynamic logic decomposition


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


Combined Decomposition and Technology Mapping

ㅁ $\Delta$-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 $\mu$
$\square$ Before finding matches at an AND2 in $\mu$, check if D-pattern matches at the AND2. If so, add the corresponding F-pattern $\square$ In practice, terminate the procedure when a feasible solution is



## Combined Decomposition and Technology Mapping

## $\square \Delta$-mapping

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

- D be the set of AND2/INV decompositions encoded in the resulting mapping graph.
- $\triangle$ be the closure of $\Lambda_{\eta}$ under the distributive transformation:

$\square$ Theorem: $\Delta_{\eta}=D_{\eta}$


## Combined Decomposition and Technology Mapping

Theorem: If1. $\eta^{*}$ is an arbitrary Boolean network obtained from $\eta$ by algebraic decomposition, and
2. $\theta$ is an arbitrary AND2/INV decomposition of $\eta^{*}$
then $\theta \in D_{n}$
$\square$ The resulting mapping graph encodes all the AND2/INV decompositions of all algebraic decompositions of $\eta$

## Combined Decomposition and Technology Mapping

$\square \Lambda$-mapping captures all AND2/INV decompositions of $\eta$ :
Phase 2 (subject graph generation) is subsumed
$\square \Delta$-mapping captures all algebraic decompositions:
Phase 2 and Phase 1 are subsumed


## Combined Decomposition and Technology Mapping

## $\square$ Summary

- 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:
$\square$ considers all retiming possibilities (implicitly) and algebraic factors across latches

