



Metodologie di progetto HW

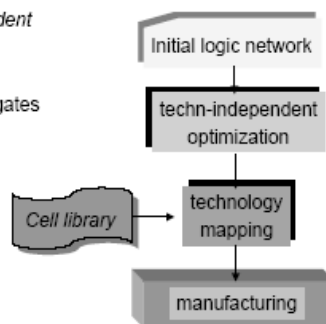
Technology Mapping

Last update: 19/03/09



Technology Mapping

- Where is it used
 - After *technology independent optimization*
- Role
 - Assign logic functions to gates from custom library
 - Optimize for area, delay, power, etc.
- Also called *library binding*

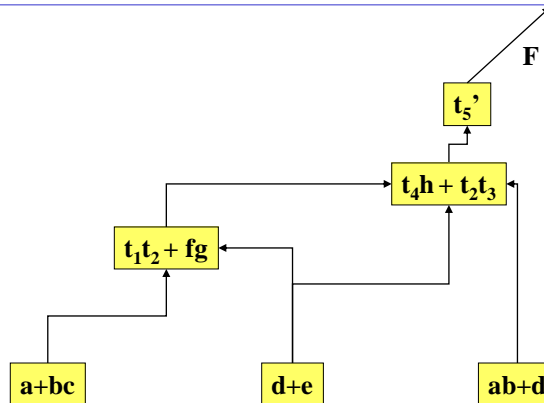




Technology Mapping

Example:

$$\begin{aligned}
 t_1 &= a + bc; \\
 t_2 &= d + e; \\
 t_3 &= ab + d; \\
 t_4 &= t_1 t_2 + fg; \\
 t_5 &= t_4 h + t_2 t_3; \\
 F &= t_5';
 \end{aligned}$$



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

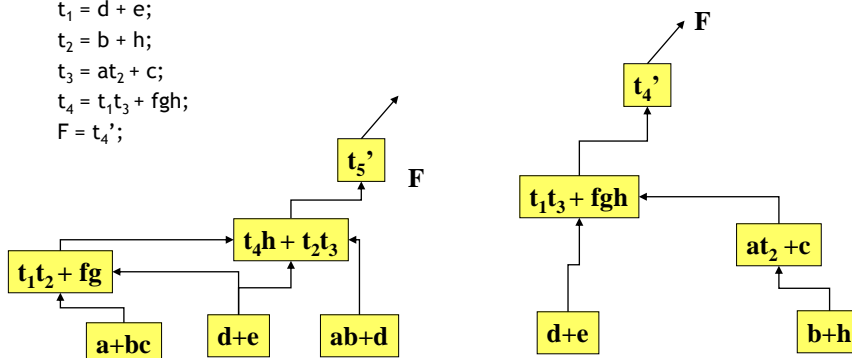
3



Optimized Equations

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

$$\begin{aligned}
 t_1 &= d + e; \\
 t_2 &= b + h; \\
 t_3 &= at_2 + c; \\
 t_4 &= t_1 t_3 + fgh; \\
 F &= t_4';
 \end{aligned}$$

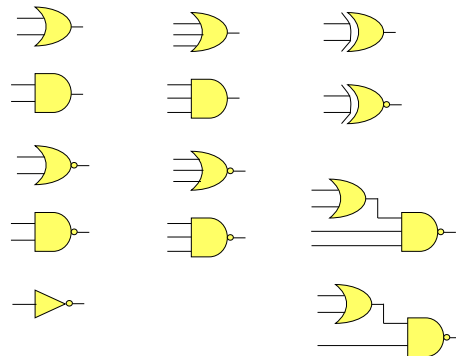


4



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.)



5



Technology Mapping

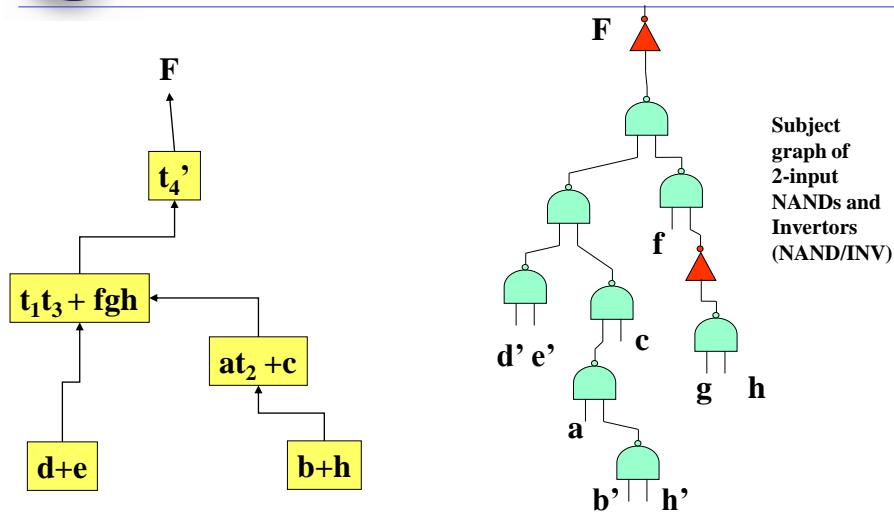
Two approaches:

- Rule-Based [LSS]
 - Same structure as rule based techniques for technology independent optimization
 - Useful for local optimizations
 - Large execution times
- 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 (Best choice)
 - 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

6



Subject Graph



7



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 exist 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.

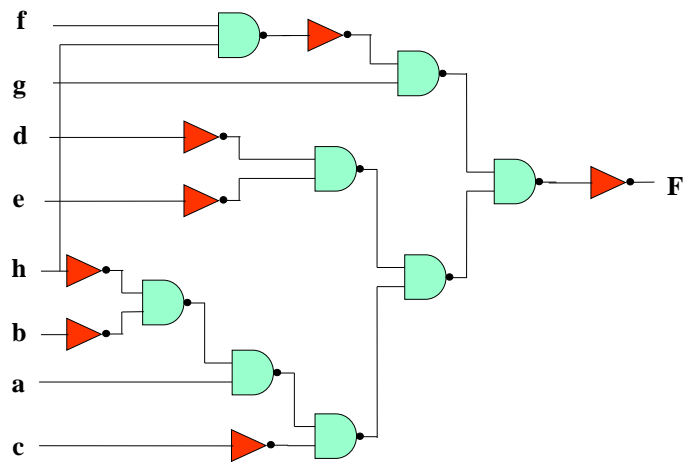
Technology mapping problem: Find a *minimum cost covering* of the subject graph by choosing from the collection of pattern graphs for all the gates in the library.

8



Subject Graph

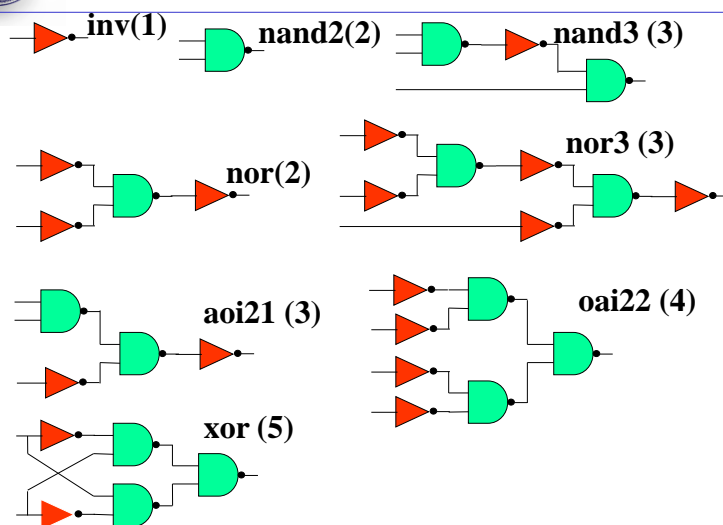
$t_1 = d + e;$
 $t_2 = b + h;$
 $t_3 = at_2 + c;$
 $t_4 = t_1t_3 + fgh;$
 $F = t_4';$



9



Pattern Graphs for the IWLS Library



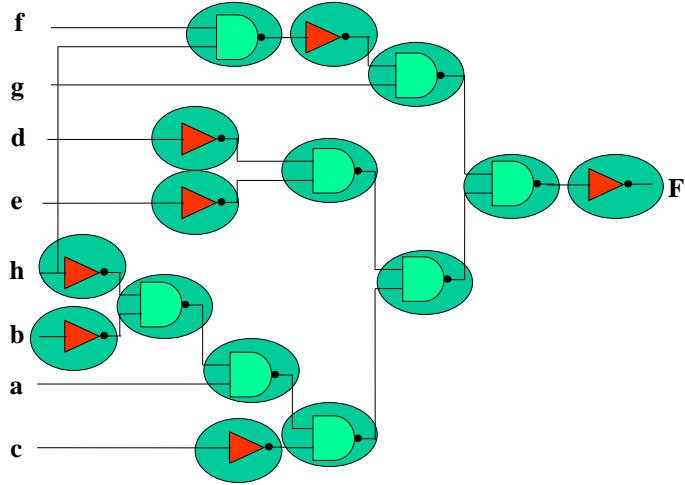
10



Subject graph covering

$$\begin{aligned}
 t_1 &= d + e; \\
 t_2 &= b + h; \\
 t_3 &= at_2 + c; \\
 t_4 &= t_1t_3 + fgh; \\
 F &= t_4';
 \end{aligned}$$

Total cost = 23



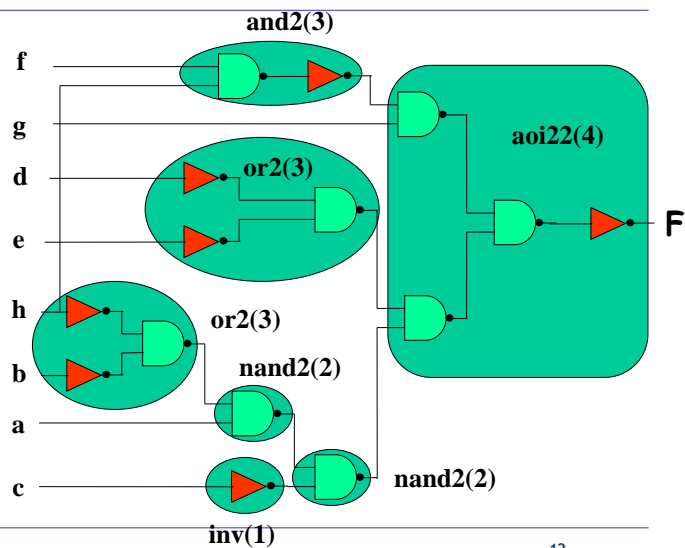
11



Better Covering

$$\begin{aligned}
 t_1 &= d + e; \\
 t_2 &= b + h; \\
 t_3 &= at_2 + c; \\
 t_4 &= t_1t_3 + fgh; \\
 F &= t_4';
 \end{aligned}$$

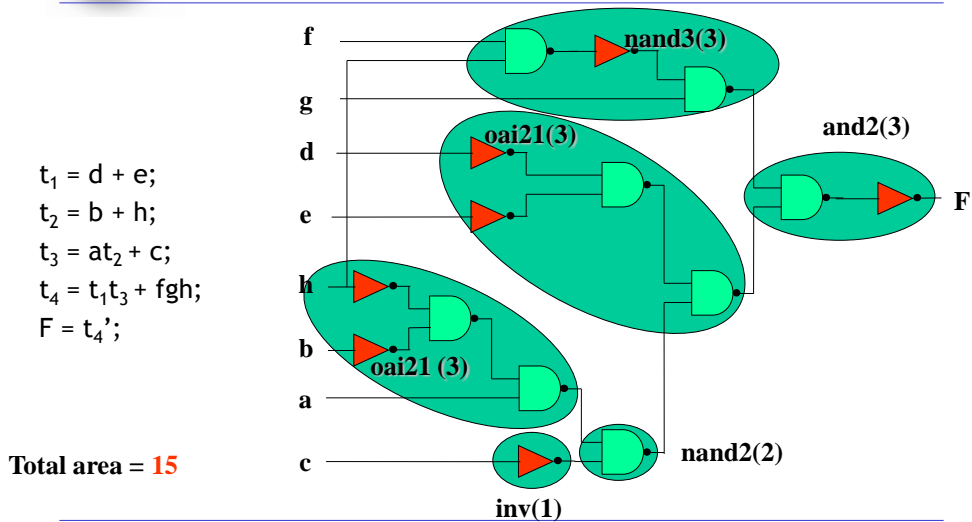
Total area = 19



12



Alternate Covering



13



Tech. 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:

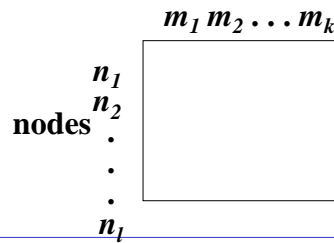
- Construct a subject DAG for the network
- Represent each gate in the target library by pattern DAG's
- Find an optimal-cost covering of subject DAG using the collection of pattern DAG's

14



DAG covering as 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, ($m_i = 1$ 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)

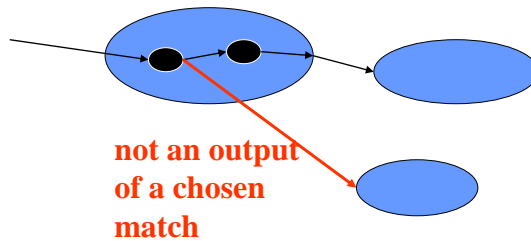


15



DAG covering as 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.

16



DAG covering as binate covering problem

- Let match m_i have subject nodes s_{i1}, \dots, 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 $j = 1 \dots n$.
This can be written

$$(m_i \Rightarrow (s_{i1} \dots s_{in})) \Leftrightarrow (\bar{m}_i + (s_{i1} \dots s_{in})) \Leftrightarrow ((\bar{m}_i + s_{i1}) \dots (\bar{m}_i + s_{in}))$$

17



DAG covering as 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 is c_i
 - $m_i = 0$ means that match i is not chosen

18



Binate Covering

This problem is **more general** than unate-covering for two-level minimization because

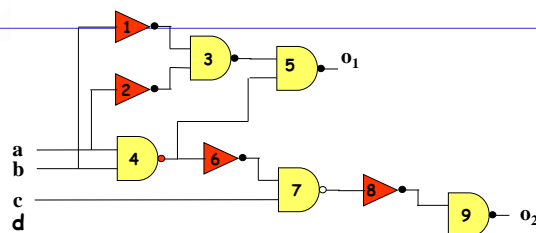
- variables are present in the covering expression in both their **true** and **complemented** forms.

The covering expression is a binate logic function, and the problem is referred to as the **binate-covering problem**.

19



Binate Covering: Example



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

20



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

21



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

22



Binate Covering: Example

- Finally, we get

$$(\bar{m}_3 + m_1) (\bar{m}_3 + m_2) (m_3 + \bar{m}_5) (\bar{m}_5 + m_4) (\bar{m}_6 + m_4)$$

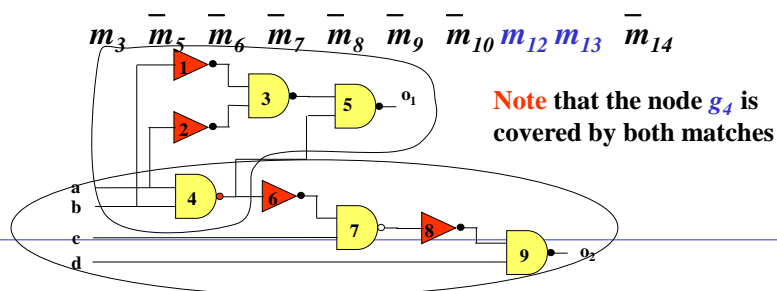
$$(\bar{m}_7 + m_6) (\bar{m}_8 + m_7 + m_{11}) (m_8 + m_9) (m_{10} + m_6)$$

$$(\bar{m}_{14} + m_4) (m_5 + m_{12} + m_{14}) (m_9 + m_{10} + m_{13})$$
- The covering expression has 58 implicants
- The least cost prime implicant is

$$\bar{m}_3 \bar{m}_5 \bar{m}_6 \bar{m}_7 \bar{m}_8 \bar{m}_9 \bar{m}_{10} m_{12} m_{13} \bar{m}_{14}$$
- 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).

23

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



24



Complexity of DAG covering

More general than unate covering

- 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.
- DAG-covering: **covering + implication** constraints

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:

- Branch and bound [Thelen]
- BDD-based [Lin and Somenzi]

Even for moderate-size networks, these are expensive.

25



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
 - (KEUTZER)

26



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 ,
Cost = SUM of best costs of implementing each **input** of pattern
plus the cost of the pattern
 - Choose least cost matching pattern for implementing N
-

27



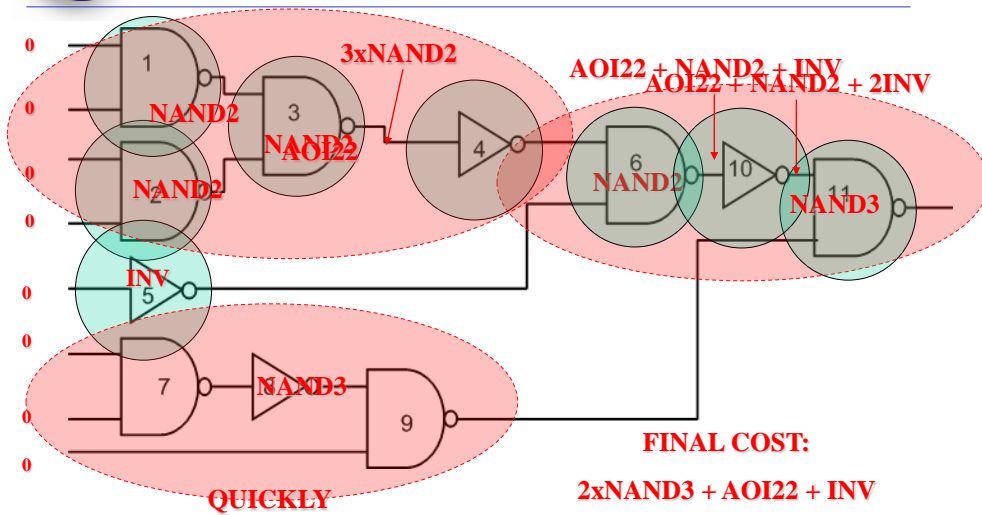
Optimum Area Algorithm

```
Algorithm OPTIMAL_AREA_COVER(node) {  
  foreach input of node {  
    OPTIMAL_AREA_COVER(input); // satisfies recurs. 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;  
    }  
  }  
}
```

28



Tree Covering in Action



29



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

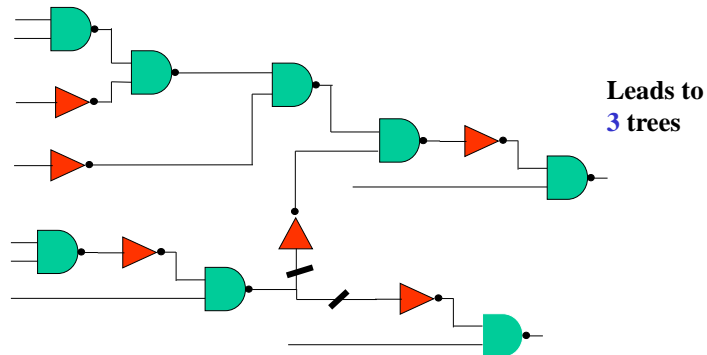
30



Partitioning the Subject DAG into Trees

Trivial partition: break the graph at all multiple-fanout points

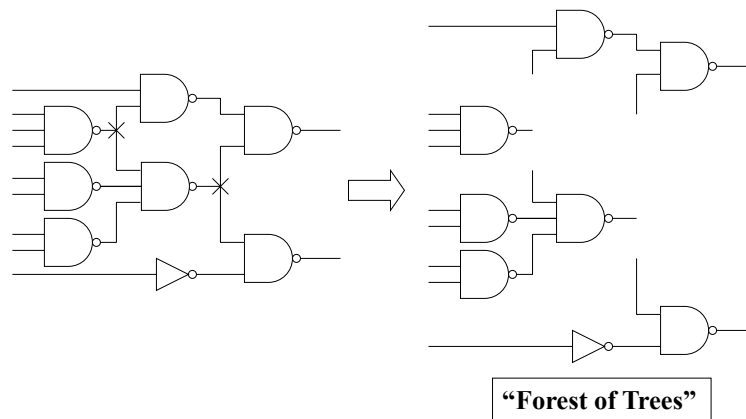
- leads to **no** “duplication” or “overlap” of patterns
- drawback - sometimes results in many of **small** trees



31



DAG-to-Tree Decomposition



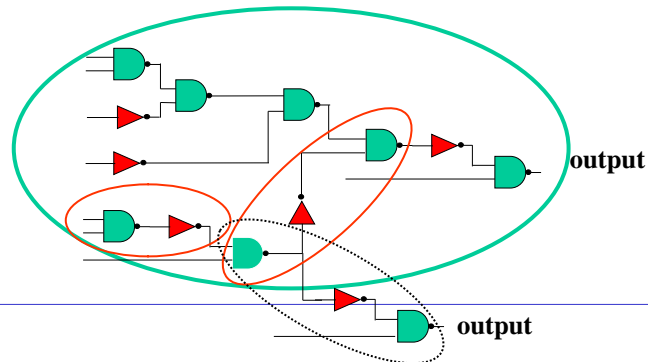
32



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 better area results



33



Min-Delay Covering

- For trees:
 - identical to min-area covering
 - use optimal delay values within the dynamic programming paradigm
- 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
- Combined objective
 - e.g. apply delay as first criteria, then area as second
 - combine with static timing analysis to focus on critical paths

34



Combined Decomposition and Technology Mapping

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**)

35



Combined Decomposition and Technology Mapping

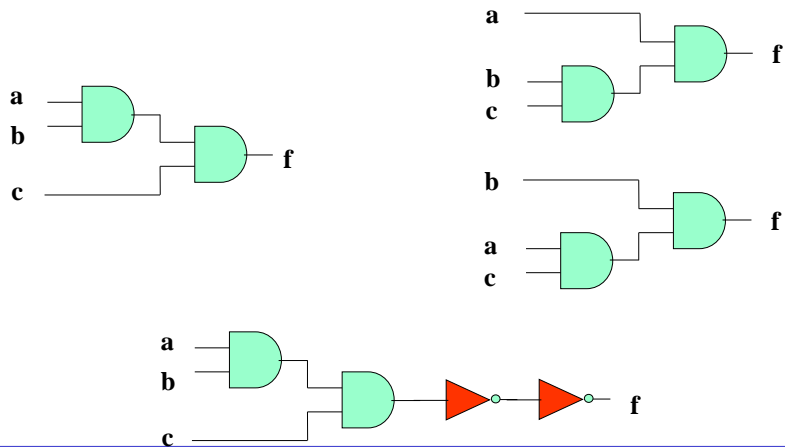
- Lehman - Watanabe - Algorithm:
- **Key Idea:**
 - Efficiently encode a **set** of AND2/INV decompositions into a **single** structure called a **mapping graph**
 - Apply a modified tree-based technology mapper while **dynamically performing algebraic logic decomposition** on the mapping graph

36



A set of AND2/INV Decompositions

$f = abc$ can be represented in various ways

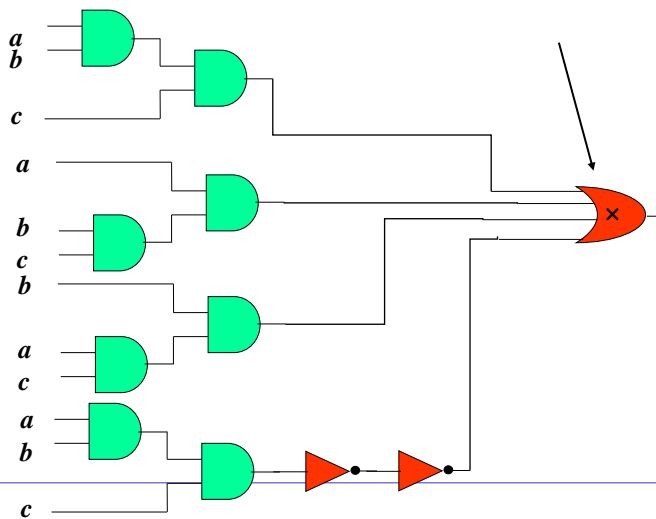


37



A set of AND2/INV Decompositions

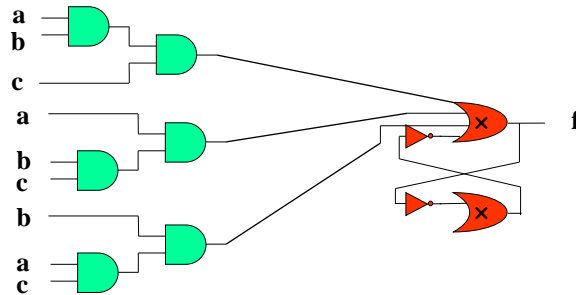
Combine them using a **choice node**



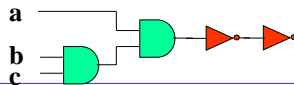
38



A set of AND2/INV Decompositions



These decompositions can be represented more compactly as:



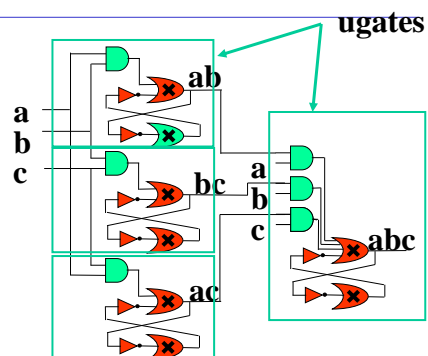
39



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



40



Tree-mapping on a Mapping Graph

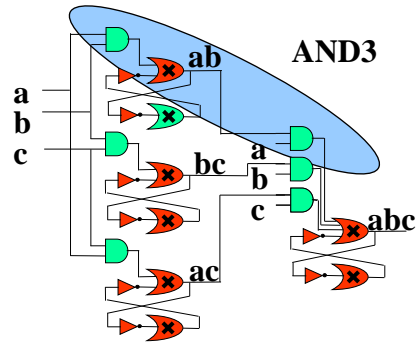
Graph-Mapping on Trees:

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



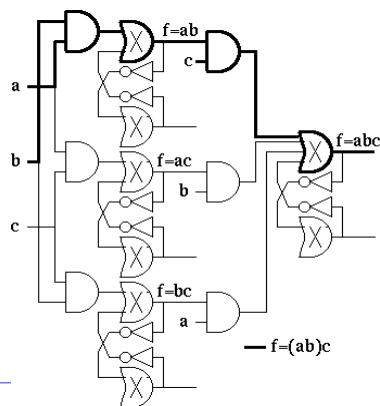
41



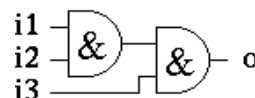
Example: Tree-mapping

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

subject graph



library pattern graph



42



THE END
