Module 1

Objective

- Libraries
- Problem formulation and analysis
- Algorithms for library binding based on structural methods
Library binding

◆ Given an unbound logic network and a set of library cells
  ▲ Transform into an interconnection of instances of library cells
  ▲ Optimize delay
    ▼ (under area or power constraints)
  ▲ Optimize area
    ▼ Under delay and/or power constraints
  ▲ Optimize power
    ▼ Under delay and/or area constraints

◆ Library binding is called also technology mapping
  ▲ Redesigning circuits in different technologies
Major approaches

◆ Rule-based systems
  ▲ Generic, handle all types of cells and situations
  ▲ Hard to obtain circuit with specific properties
  ▲ Data base:
    ▼ Set of pattern pairs
    ▼ Local search: detect pattern, implement its best realization

◆ Heuristic algorithms
  ▲ Typically restricted to single-output combinational cells
  ▲ Library described by cell functionality and parameters

◆ Most systems use a combination of both approaches:
  ▲ Rules are used for I/Os, high buffering requirements, ...
Library binding: issues

Matching:

- A cell matches a sub-network when their terminal behavior is the same
- Tautology problem
- Input-variable assignment problem

Covering:

- A cover of an unbound network is a partition into sub-networks which can be replaced by library cells.
- Binate covering problem
Assumptions

◆ Network granularity is fine
  ▲ Decomposition into base functions:
    ▲ 2-input AND, OR, NAND, NOR

◆ Trivial binding
  ▲ Use base cells to realize decomposed network
  ▲ There exists always a trivial binding:
    ▼ Base-cost solution…
Example

\[ z = a + w \]
\[ w = x + y \]
\[ y = d + u \]
\[ x = b + c \]
\[ u = e + f \]
Example

\[
\begin{array}{|c|c|}
\hline
\text{Library} & \text{Cost} \\
\hline
\text{AND2} & 4 \\
\text{OR2} & 4 \\
\text{OA21} & 5 \\
\hline
\end{array}
\]

\[
x = b + c \\
y = ax \\
z = xd
\]

\[
m_1: \{v_1, \text{OR2}\} \\
m_2: \{v_2, \text{AND2}\} \\
m_3: \{v_3, \text{AND2}\} \\
m_4: \{v_1, v_2, \text{OA21}\} \\
m_5: \{v_1, v_3, \text{OA21}\}
\]
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 covering 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 approach to library binding

◆ Split problem into various stages:
  ▲ 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 networks
  ▲ Covering
    ▼ Cover each sub-network by library cells

◆ Most tools use this strategy
  ▲ Sometimes stages are merged
Decomposition
Partitioning
Covering
Heuristic algorithms

◆ Structural approach
  ▲ Model functions by patterns
    ▼ Example: tree, dags
  ▲ Rely on pattern matching techniques

◆ Boolean approach
  ▲ Use Boolean models
  ▲ Solve the tautology problem
    ▼ Use BDD technology
  ▲ More powerful
Example

- **Boolean vs. 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 exist
Structural matching and covering

- Expression patterns
  - Represented by dags

- Identify pattern dags in network
  - Sub-graph isomorphism

- Simplification:
  - Use tree patterns

- Typical problems with EXORs and MAJority functions
Example
Tree-based matching

Network:
- Partitioned and decomposed
  - NOR2 (or NAND2) + INV
  - Generic base functions
    - Not much used
  - Subject tree

Library
- Represented by trees
- Possibly more than one tree per cell

Pattern recognition
- Simple binary tree match
- Aho-Corasik automaton
Example

SUBJECT TREE

PATTERN TREES

cost = 2
INV

cost = 3
NAND

cost = 4
AND

cost = 5
OR
Example: Lib

Match of s: t1
  cost = 2

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

Match of t: t3
  cost = 4

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

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

(c) Giovanni De Micheli
Tree covering

- Dynamic programming
  - Visit subject tree bottom up
- At each vertex
  - Attempt to match:
    - Locally rooted subtree to all library cell
    - Find best match and record
  - There is always a match when the base cells are in the library
- Bottom-up search yields and optimum cover
- Caveat:
  - Mapping into trees is a distortion for some cells
  - Overall optimality is weakened by the overall strategy of splitting into several stages
Different covering problems

◆ Covering for minimum area:
  ▲ Each cell has a fixed area cost (label)
  ▲ Area is additive:
    ▼ Add area of match to cost of sub-trees

◆ Covering for minimum delay:
  ▲ Delay is fanout independent
    ▼ Delay computed with (max, +) rules
    ▼ Add delay of match to highest cost of sub-trees
  ▲ Delay is fanout dependent
    ▼ Look-ahead scheme is required
Simple library

INV

NAND2

AND2

NOR2

OR2

AOI21

AOI22
Example – minimum area cover

Area cost: INV:2 NAND2:3 AND2:4 AOI21:6

<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>3+3 = 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>w</td>
<td>t2</td>
<td>NAND2(y,z)</td>
<td>3+6+2 = 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o</td>
<td>t1</td>
<td>INV(w)</td>
<td>2+11 = 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t3</td>
<td></td>
<td>AND2(y,z)</td>
<td>6 + 4 + 2 = 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t6B</td>
<td></td>
<td>AOI21(x,d,a)</td>
<td>6 + 3 = 9</td>
</tr>
</tbody>
</table>
**Example – minimum delay cover**

- **Fixed delays:** INV: 2  NAND2: 4  AND2: 5  AOI21: 10
- **All inputs are stable at time 0, except for** $t_d = 6$

<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><strong>t2</strong></td>
<td>NAND2(b,c)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td><strong>t1</strong></td>
<td>INV(a)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z</td>
<td><strong>t2</strong></td>
<td>NAND2(x,d)</td>
<td>6 + 4 = 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>w</td>
<td><strong>t2</strong></td>
<td>NAND2(y,z)</td>
<td>10 + 4 = 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o</td>
<td><strong>t1</strong></td>
<td>INV(w)</td>
<td>14 + 2 = 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t3</td>
<td></td>
<td>AND2(y,z)</td>
<td>10 + 5 = 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t6B</td>
<td></td>
<td>AOI21(x,d,a)</td>
<td>10 + 6 = 16</td>
</tr>
</tbody>
</table>
Minimum-delay cover for load-dependent delays

◆ Model

▲ Gate delay is \( d = \alpha + \beta \text{cap}_\text{load} \)
▲ Capacitive load depends on the driven cells (fanout cone)
▲ There is a finite (possibly small) set of capacitive loads

◆ Algorithm

▲ Visit subject tree bottom up
▲ Compute an array of solutions for each possible load
▲ For each input to a matching cell, the best match for the corresponding load is selected

◆ Optimality

▲ Optimum solution when all possible loads are considered
▲ Heuristic: group loads into bins
Example – minimum delay cover

- Delays: **INV**: 1+load  **NAND2**: 3+load  **AND2**: 4+load  **AOI21**: 9+load
- All inputs are stable at time 0, except for $t_d = 6$
- All loads are 1

Same as before!

<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>a</td>
<td></td>
<td>x</td>
<td>t2</td>
<td>NAND2(b,c)</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>y</td>
<td>t1</td>
<td>INV(a)</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>z</td>
<td>t2</td>
<td>NAND2(x,d)</td>
<td>6+4 = 10</td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>w</td>
<td>t2</td>
<td>NAND2(y,z)</td>
<td>10 + 4 = 14</td>
</tr>
<tr>
<td>e</td>
<td></td>
<td>o</td>
<td>t1</td>
<td>INV(w)</td>
<td>14 + 2 = 16</td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>t3</td>
<td></td>
<td>AND2(y,z)</td>
<td>10 + 5 = 15</td>
</tr>
<tr>
<td>g</td>
<td></td>
<td>t6B</td>
<td></td>
<td>AOI21(x,d,a)</td>
<td>10 + 6 = 16</td>
</tr>
</tbody>
</table>
Example – minimum delay cover

- Delays: **INV**: 1+load  **NAND2**: 3+load  **AND2**: 4+load  **AOI21**: 9+load
- All inputs are stable at time 0, except for $t_d = 6$
- All loads are 1 (for cells seen so far)
- Add new cell **SINV** with delay $1 + \frac{1}{2} \text{ load}$ and load 2
- The sub-network drives a load of 5
Example – minimum delay cover

<table>
<thead>
<tr>
<th>Network</th>
<th>Subject graph</th>
<th>Vertex</th>
<th>Match</th>
<th>Gate</th>
<th>Cost (Load=1)</th>
<th>Cost (Load=2)</th>
<th>Cost (Load=5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y,z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a,x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b,c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>t2</td>
<td>NAND2(b,c)</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>t1</td>
<td>INV(a)</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>t2</td>
<td>NAND2(x,d)</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>t2</td>
<td>NAND2(y,z)</td>
<td>14</td>
<td>15</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>t1</td>
<td>INV(w)</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td>AND2(y,z)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>t6B</td>
<td>AOI21(x,d,a)</td>
<td>SINV(w)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18.5</td>
</tr>
</tbody>
</table>
Library binding and polarity assignment

- Search for lower cost solutions allowing signals to be generated with inverted polarity
  - More cells to choose from
- Polarities can be adjusted at register and/or I/O boundaries
- Within structural covering:
  - Polarity assignment is handled by a smart trick
- Within structural covering
  - Polarity assignment is built into the formulation
Structural covering and polarity assignment

- Assume subject network is decomposed into base functions such as NAND2 and INV
- Pre-process subject network by adding inverter pairs between NANDs
- Provide I/Os with both polarity
- Add to library inverter-pair cell 2INV to the library:
  - ▲ Cell corresponds to a connection and has 0 cost
  - ▲ Unnecessary 2INV will be removed by the algorithm
- Apply bottom-up dynamic programming cover algorithm
Example
General issues with covering

- Decomposition and covering yield solutions that are locally sub-optimal at multiple fanout points
  - Covering in various logic cones is independent
  - High capacitive loads skews the solution
- Approaches
  - Implicit decomposition methods
    - Compute and map all possible decompositions of a network
  - Cell cloning and buffering at cone vertices
  - Wavefront mapping
Wavefront mapping

◆ A *wavefront* is a sub-network
  ▲ Cutting all I/O paths
  ▲ With width $W$

◆ When $W$ is the circuit depth, then covering for load-dependent delays is optimum
  ▲ The entire network is covered

◆ For reasonably smaller values of $W$, optimality is preserved
  ▲ The wavefront can be dynamically decomposed
  ▲ The wavefront can be mapped (with duplications) for delay or other objectives
Buffering

◆ Drive multiple sinks:
  ▲ Possibly with +/- polarities
  ▲ Possibly with different loads
  ▲ Possibly with different criticality

◆ Logic/electrical effort theory
  ▲ delay = p + q gain
  ▲ gain = load/C_{in} is a design parameter

◆ Issues:
  ▲ Determine minimal buffer tree delay
  ▲ Determine tree with minimal load
    ▼ satisfying all timing constraint

◆ Solutions:
  ▲ Make electrical effort uniform in chains
    ▼ Closed form solution
  ▲ Adapt to non-continuous cell sizes and tree structure
    ▼ Gain based algorithm
Module 2

Objectives

- Boolean covering
- Boolean matching
- Simultaneous optimization and binding
- Extensions to Boolean methods
Boolean covering

- Decompose network into base functions
- Partition network into cones
- Apply bottom-up covering to each cone

▲ When considering vertex v:
  ▼ Construct clusters by local elimination
  ▼ Limit the depth of the cluster by limiting the support of the function
  ▼ Associate several functions with vertex v
  ▼ Apply matching and record cost

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

- **Cluster function** \(f(x)\)
  - Sub-network behavior

- **Pattern function** \(g(y)\)
  - Cell behavior

- **\(P\)-equivalence**
  - Is there a permutation operator \(P\), such that \(f(x) = g(P \times x)\) is a tautology?

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

◆ **NPN** classification of logic functions

◆ **NPN-equivalence**

▲ There exist a permutation operator $P$ and complementation operators $N_i$ and $N_o$, such that $f(x) = N_o \circ g \circ (P \circ N_i \circ x)$ is a tautology

◆ **Variations:**

▲ $N$-equivalence

▲ $PN$-equivalence
Boolean matching

◆ Pin assignment problem:
  ▲ Map cluster variables $x$ to pattern variables $y$
  ▲ Characteristic equation: $A(x, y) = 1$

◆ Pattern function under variable assignment:
  ▲ $g_A (x) = S_y \left( A(x, y) g(y) \right)$

◆ Tautology problem
  ▲ $f(x) = g_A (x)$
  ▲ $\forall x \ f(x) = S_y \left( A(x, y) g(y) \right)$
Cluster terminals: x -- cell terminals: y

Assign x₁ to y’₂ and x₂ to y₁

Characteristic equation

\[ A(x₁, x₂, y₁, y₂) = (x₁ \oplus y₂)(x₂ \oplus y₁) \]

AND pattern function

\[ g = y₁ y₂ \]

Pattern function under assignment

\[ S_{y₁y₂} A \ g = S_{y₁y₂} ((x₁ \oplus y₂)(x₂ \oplus y₁)y₁y₂) = x₂x’₁ \]
Signatures and filters

- Capture some properties of Boolean functions
- If signatures do not match, there is no match
- Signatures are 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 variables in $f(x)$ with unate variables in $g(y)$
  ▲ Binate variables in $f(x)$ with binate variables in $g(y)$

◆ Variables or group of variables:
  ▲ That are interchangeable in $f(x)$ must be interchangeable in $g(y)$
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 and 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 \( h \), where:

\[
f f'_{\text{DC}} \leq h \leq f + f_{\text{DC}}
\]

Matching condition:

\[
\forall x \ ( f_{\text{DC}}(x) + f(x) \oplus S_y ( A(x,y) g(y) ) )
\]
Example

◆ Assume $v_x$ is bound to an $\text{OR3}(c',b,e)$
◆ Don’t care set includes $x \oplus (c' + b + e)$
◆ Consider $f_j = x(a+c)$ with $\text{CDC} = x' c'$
◆ No simplification.
  ▲ Mapping into AOI gate.

◆ Matching with DCs.
  ▲ Map to a MUX gate.
Example
Extended matching

◆ Motivation:
  ▲ Search implicitly for best pin assignment
  ▲ Make a single test, determining matching and assignment

◆ Technique:
  ▲ Construct BDD model of cell and assignments

◆ Visual intuition:
  ▲ Imagine to place \texttt{MUX} function at cell inputs
  ▲ Each cell input can be routed to any cluster input (or voltage rail)
  ▲ Input polarity can be changed:
    ▼ NP-equivalence (extensible to NPN)
  ▲ Cell and cluster may differ in size

◆ Cell and multiplexers are described by a composite function $G(x,c)$
  ▲ Pin assignment is determining $c$
Example

\( g = y_1 + y_2 y'_3 \)

\( y_1(c,x) = (c_0c_1x_1 + c_0c'_1x_2 + c'_0c_1x_3) \oplus c_2 \)

\( G = y_1(c,x) + y_2(c,x) y_3(c,x)' \)

An EXOR gate can be placed at the gate output to support NPN-equivalence check.
Extended matching modeling

◆ Model composite functions with 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 \( c \): \( m( \lceil \log_2 n \rceil + 1 ) \)

▲ One additional variable for output polarity

◆ A match exists if there is at least one value of \( c \) satisfying
\[
M(c) = \forall_x [ G(x,c) \oplus f(x) ]
\]
Example

- **Cell:** \( g = x' \cdot y \)
- **Cluster:** \( f = wz' \)

- \( G(a,b,c,d) = (c \oplus (za+wa'))' \cdot (d \oplus (zb+wb')) \)

- \( F \oplus G = (wz) \oplus (c \oplus (za+wa'))' \cdot (d \oplus zb+wb') \)

- \( M(c) = ab' \cdot c' \cdot d' + a' \cdot bcd \)
Extended matching

- 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 representation
- Extensions:
  - Support multiple-output matching
  - Full library representation
Full library model

- Represent full library with $L(x,c)$
  - One single (large) BDD

- Visual intuition
  - All composite cells connected to a MUX

- Compare cluster to library $L(x,c)$
  - $M(c) = \forall_x [ L(x,c) \oplus f(x) + f_{DC}(x) ]$

  Vector $c$ determines:
  - Feasible cell matches
  - Feasible pin assignments
  - Feasible output polarity
Summary

◆ Library binding is a key step in synthesis

◆ Most systems use some rules together with heuristic algorithms that concentrate on combinational logic
  ▲ Best results are obtained with Boolean matching
  ▲ Sometimes structural matching is used for speed

◆ Library binding is tightly linked to buffering and to physical design