Two-level Logic Synthesis and Optimization

Giovanni De Micheli

Integrated Systems Centre

EPF Lausanne
Module 1

◆ Objectives

- Fundamentals of logic synthesis
- Mathematical formulation
- Definition of the problems
Combinational logic design
Background

◆ Boolean Algebra

  ▪ Quintuple \( (B, +, \cdot, 0, 1) \)
  
  ▪ Binary Boolean algebra \( B = \{ 0, 1 \} \)

◆ Boolean function

  ▪ Single output \( f : B^n \rightarrow B \)
  
  ▪ Multiple output \( f : B^n \rightarrow B^m \)
  
  ▪ Incompletely-specified:
    
    ◆ \textit{Don’t care} symbol: *
    
    ◆ \( f : B^n \rightarrow \{ 0, 1, * \}^m \)
The *don’t care* conditions

- We do not care about the value of a function
- Related to the environment
  - Input patterns that never occur
  - Input patterns such that some output is never observed
- Very important for synthesis and optimization
Definitions

◆ Scalar function:

- **ON-set**
  - Subset of the domain such that $f$ is true

- **OFF-set**
  - Subset of the domain such that $f$ is false

- **DC-set**
  - Subset of the domain such that $f$ is a *don’t care*

◆ Multiple-output function:

- ON, OFF, DC-sets defined for each component
Cubical representation
Definitions

- **Boolean variables**
- **Boolean literals:**
  - Variables and their complement
- **Product or cube:**
  - Product of literals
- **Implicant:**
  - Product implying a value of the function (usually 1)
  - Hypercube in the Boolean space
- **Minterm:**
  - Product of all input variables implying a value of the function (usually 1)
  - Vertex in the Boolean space
Tabular representations

- **Truth table**
  - List of all minterms of a function

- **Implicant table or cover**
  - List of implicants sufficient to define a function

- **Note**
  - Implicant tables are smaller in size as compared to truth tables
Example of truth table

- \( x = ab + a'c \);  \( y = ab + bc + ac \)

<table>
<thead>
<tr>
<th>abc</th>
<th>xy</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
</tr>
<tr>
<td>001</td>
<td>10</td>
</tr>
<tr>
<td>010</td>
<td>00</td>
</tr>
<tr>
<td>011</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td>00</td>
</tr>
<tr>
<td>101</td>
<td>01</td>
</tr>
<tr>
<td>110</td>
<td>11</td>
</tr>
<tr>
<td>111</td>
<td>11</td>
</tr>
</tbody>
</table>
**Example of implicant table**

- $x = ab + a'c$;  
  $y = ab + bc + ac$

<table>
<thead>
<tr>
<th>abc</th>
<th>xy</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>10</td>
</tr>
<tr>
<td>*11</td>
<td>11</td>
</tr>
<tr>
<td>101</td>
<td>01</td>
</tr>
<tr>
<td>11*</td>
<td>11</td>
</tr>
</tbody>
</table>
Cubical representation of minterms and implicants

\[ f_1 = a'b'c' + a'b'c + ab'c + abc + abc' \]

\[ f_2 = a'b'c + ab'c \]
Representations

- Visual representations
  - Cubical notation
  - Karnaugh maps

- Computer-oriented representations
  - Matrices
    - Sparse
    - Various encoding
  - Binary-decision diagrams
    - Address sparsity and efficiency
Module 2

Objectives

- Two-level logic optimization
- Motivation
- Models
- Exact algorithms for logic optimization
Two-level logic optimization
motivation

◆ Reduce size of the representation

◆ Direct implementation
  ◆ PLAs reduce size and delay

◆ Other implementation styles
  ◆ Reduce amount of information
  ◆ Simplify local functions and connections
Programmable logic arrays

- Macro-cells with rectangular structure
  - Implement any multi-output function
  - Layout generated by module generators
  - Fairly popular in the seventies/eighties

- Advantages
  - Simple, predictable timing

- Disadvantages
  - Less flexible than cell-based realization
  - Dynamic operation

- Open issue
  - Will PLA structures be useful with new nanotechnologies? (e.g., nanowires)
Programmable logic array

\[ f_1 = a'b' + b'c + ab; \quad f_2 = b'c \]
Two-level minimization

◆ Assumptions

  ♦ Primary goal is to reduce the number of implicants
  ♦ All implicants have the same cost
  ♦ Secondary goal is to reduce the number of literals

◆ Rationale

  ♦ Implicants correspond to PLA rows
  ♦ Literals correspond to transistors
Definitions

- **Minimum cover**
  - Cover of a function with minimum number of implicants
  - Global optimum

- **Minimal cover or irredundant cover**
  - Cover of the function that is not a proper superset of another cover
  - No implicant can be dropped
  - Local optimum

- **Minimal w.r.to 1-implicant containment**
  - No implicant contained by another one
  - Weak local optimum
Example

\[ f_1 = a'b'c' + a'b'c + ab'c + abc + abc'; \quad f_2 = a'b'c + ab'c \]

Minimum cover

Irredundant cover

Minimal cover w.r. to single implicant containment

(c) Giovanni De Micheli
Definitions

◆ Prime implicant
  ➤ Implicant not contained by any other implicant

◆ Prime cover
  ➤ Cover of prime implicants

◆ Essential prime implicant
  ➤ There exist some minterm covered only by that prime implicant
  ➤ Needs to be included in the cover
Two-level logic minimization

◆ Exact methods
  ◆ Compute minimum cover
  ◆ Often difficult/impossible for large functions
  ◆ Based on Quine-McCluskey method

◆ Heuristic methods
  ◆ Compute minimal covers (possibly minimum)
  ◆ Large variety of methods and programs
    ♦ MINI, PRESTO, ESPRESSO
Exact logic minimization

**Quine’s theorem:**
- There is a minimum cover that is prime

**Consequence**
- Search for minimum cover can be restricted to prime implicants

**Quine-McCluskey method**
- Compute prime implicants
- Determine minimum cover
Prime implicant table

- **Rows:** minterms
- **Columns:** prime implicants

**Exponential size**

- $2^n$ minterms
- Up to $3^n / n$ prime implicants

**Remarks**

- Some functions have much fewer primes
- Minterms can be grouped together
- Implicit methods for implicant enumeration
Example

- \( f = a'b'c' + a'b'c + ab'c + abc + abc' \)

- Primes:

  \[
  \begin{array}{c|cccc}
  \alpha & 00 & \ast & 1 \\
  \beta & \ast & 01 & 1 \\
  \gamma & 1 & *1 & 1 \\
  \delta & 11 & * & 1 \\
  \end{array}
  \]

- Table:

  \[
  \begin{array}{cccc}
  \alpha & \beta & \gamma & \delta \\
  \hline
  000 & 1 & 0 & 0 & 0 \\
  001 & 1 & 1 & 0 & 0 \\
  101 & 0 & 1 & 1 & 0 \\
  111 & 0 & 0 & 1 & 1 \\
  110 & 0 & 0 & 0 & 1 \\
  \end{array}
  \]
Minimum cover
early methods

- **Reduce table**
  - Iteratively identify essentials, save them in the cover.
  - Remove covered minterms

- **Petrick’s method**
  - Write covering clauses in *pos* form
  - Multiply out pos form into *sop* form
  - Select cube of minimum size

- **Remark**
  - Multiplying out clauses has exponential cost
Example

- pos clauses
  - $(\alpha) (\alpha + \beta) (\beta + \gamma) (\gamma + \delta) (\delta) = 1$

- sop form:
  - $\alpha\beta\delta + \alpha\gamma\delta = 1$

- Solutions:
  - $\{ \alpha \beta \delta \}$
  - $\{ \alpha \gamma \delta \}$
Matrix representation

◆ View table as Boolean matrix: \( A \)

◆ Selection Boolean vector for primes: \( x \)

◆ Determine \( X \) such that
  - \( A \ x \geq 1 \)
  - Select enough columns to cover all rows

◆ Minimize cardinality of \( x \)
Example

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
0 \\
1 \\
\end{bmatrix}
=
\begin{bmatrix}
1 \\
2 \\
1 \\
1 \\
\end{bmatrix}
\]
Covering problem

◆ Set covering problem:
  ♦ A set $S$ -- minterm set
  ♦ A collection $C$ of subsets (implicant set)
  ♦ Select fewest elements of $C$ to cover $S$

◆ Computationally intractable problem

◆ Exact solution method
  ♦ Branch and bound algorithm

◆ Several heuristic approximation methods
Example
Edge-cover of a hypergraph
Branch and bound algorithm

- Tree search in the solution space
  - Potentially exponential

- Use bounding function:
  - If the lower bound on the solution cost that can be derived from a set of future choices exceeds the cost of the best solution seen so far, then kill the search
  - Bounding function should be fast to evaluate and accurate

- Good pruning may expedite the search
Example

Bound = 6
Kill sub-tree

(c) Giovanni De Micheli
Branch and bound for logic minimization
Reduction strategies

◆ Use matrix formulation of the problem

◆ Partitioning:
  ◆ If $A$ is block diagonal:
    ♦ Solve covering problems for the corresponding blocks

◆ Essentials
  ◆ Column incident to one (or more) rows with single 1
    ♦ Select column
    ♦ Remove covered row(s) from table
Branch and bound for logic minimization
Reduction strategies

◆ Column (implicant) dominance:
  ♦ If $a_{ki} \geq a_{kj}$ for all $k$
     ♦ Remove column $j$ (dominated)
  ♦ Dominated implicant ($j$) has its minterms already covered by dominant implicant ($i$)

◆ Row (minterm) dominance:
  ♦ If $a_{ik} \geq a_{jk}$ for all $k$
     ♦ Remove row $i$ (dominant)
  ♦ When an implicant covers the dominated minterm, it also covers the dominant one
Example

\[
A = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\end{bmatrix}
\]
Example

- Fourth column is essential
- Fifth column is dominated
- Fifth row is dominant
- Matrix after reductions:

\[
\mathbf{A} = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\end{bmatrix}
\]
Branch and bound covering algorithm

\[ EXACT\_COVER(A,x,b) \] {
Reduce matrix \( A \) and update corresponding \( x \);
if (current_estimate \( \geq \) \( |b| \)) return \( (b) \);
if (\( A \) has no rows) return \( (x) \);
select a branching column \( c \);
\[ x_c = 1; \]
\( \tilde{A} = A \) after deleting \( c \) and rows incident to it;
\[ x^\sim = EXACT\_COVER(\tilde{A},x,b); \]
if (\( |x^\sim| < |b| \) )
\[ b = x^\sim; \]
\[ x_c = 0; \]
\( \tilde{A} = A \) after deleting \( c \);
\[ x^\sim = EXACT\_COVER(\tilde{A},x,b); \]
if (\( |x^\sim| < |b| \) )
\[ b = x^\sim; \]
return \( (b) \);
}

(c) Giovanni De Micheli
Bounding function

- Estimate lower bound on covers that can be derived from current solution vector $x$

- The sum of the 1s in $x$, plus bound of cover for local $A$
  - Independent set of rows
    - No 1 in the same column
    - Require independent implicants to cover
  - Construct graph to show pairwise independence
  - Find clique number
    - Size of the largest clique
  - Approximation (lower) is acceptable
Example

- Row 4 independent from 1,2,3
- Clique number and bound is 2

\[
A = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\end{bmatrix}
\]
Example

◆ There are no independent rows
  • Clique number is 1 (one vertex)
  • Bound is 1+1= 2
    ♦ Because of the essential already selected

\[
A = \begin{bmatrix}
1 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1 \\
\end{bmatrix}
\]
Example
Branching on the cyclic core

◆ Select first column
  - Recur with $\tilde{A} = [11]$
    - Delete one dominated column
    - Take other column (essential)
  - New cost is 3

◆ Exclude first column
  - Find another solution with cost equal to 3.
  - Discard

\[
A = \begin{bmatrix}
1 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1
\end{bmatrix}
\]
Espresso-exact

- Exact 2-level logic minimizer
- Exploits iterative reduction and branch and bound algorithm on cyclic core
- Compact implicant table
  - Rows represent groups of minterms covered by the same implicants
- Very efficient
  - Solves most benchmarks
### Example

After removing the essentials

<table>
<thead>
<tr>
<th></th>
<th>α</th>
<th>β</th>
<th>ε</th>
<th>ζ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000,0010</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>α</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 * * 0</td>
<td>1</td>
<td>0 * * 1</td>
<td>1</td>
</tr>
<tr>
<td>γ</td>
<td>0 1 **</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>δ</td>
<td>1 0 **</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ε</td>
<td>1 * 0 1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ζ</td>
<td>* 1 0 1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Giovanni De Micheli
Exact two-level minimization

There are two main difficulties:

- Storage of the implicant table
- Solving the cyclic core

Implicit representation of prime implicants

- Methods based on binary decision diagrams
- Avoid explicit tabulation

Recent methods make 2-level optimization solve exactly almost all benchmarks

- Heuristic optimization is just used to achieve solutions faster
Module 3

Boolean Relations

- Motivation of using relations
- Optimization of realization of Boolean relation
- Comparisons to two-level optimization
Boolean relations

- Generalization of Boolean functions
- More than one output pattern may correspond to an input pattern
  - Multiple-choice specifications
  - Model inner blocks of multi-level circuits
- Degrees of freedom in finding an implementation
  - More general than don’t care conditions
- Problem:
  - Given a Boolean relation, find a minimum cover of a compatible Boolean function that can implement the relation
Example

◆ Compare:
  - a + b > 4 ?
  - a + b < 3 ?

(c) Giovanni De Micheli
### Example

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$a_0$</th>
<th>$b_1$</th>
<th>$b_0$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>${ 000, 001, 010 }$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>${ 000, 001, 010 }$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>${ 000, 001, 010 }$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>${ 000, 001, 010 }$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>${ 000, 001, 010 }$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>${ 000, 001, 010 }$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>${ 011, 100 }$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>${ 101, 110, 111 }$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>${ 101, 110, 111 }$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>${ 101, 110, 111 }$</td>
</tr>
</tbody>
</table>
Example

Circuit is no longer an adder

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$a_0$</th>
<th>$b_1$</th>
<th>$b_0$</th>
<th>$\times$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>0</td>
<td>*</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>100</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>*</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>001</td>
</tr>
</tbody>
</table>
Minimization of Boolean relations

Since there are many possible output values (for any input), there are many logic functions implementing the relation.

- Compatible functions

Problem

- Find a minimum compatible function

Do not enumerate all compatible functions

- Compute the primes of the compatible functions
  - C-primes
- Derive a logic cover from the c-primes
Binate covering

◆ Covering problem is more complex
  ◦ As compared to minimizing logic functions.

◆ In classic Boolean minimization we just need enough implicants to cover the minterm
  ◦ Covering clause is *unate* in all variables
  ◦ Any additional implicant does not hurt

◆ In Boolean relation optimization, we need to pick implicants to realize a compatible function
  ◦ Some implicants cannot be taken together
  ◦ Covering clause is *binate* (implicant mutual exclusion)
  ◦ Non-compact Boolean space
Solving binate covering

◆ Binate cover can be solved with branch and bound
  ♦ In practice much more difficult to solve, because it is harder to bound effectively

◆ Binate cover can be reduced to min-cost SAT
  ♦ SAT solvers can be used

◆ Binate cover can be also modeled by BDDs

◆ Several approximation algorithms for binate cover
Boolean relations

- Generalization of Boolean functions
  - More degrees of freedom than don’t care sets
- Useful to represent multiple choice
- Useful to model internals of logic networks
- Elegant formalism, but computationally-intensive solution method