Motivation

- Algebraic manipulation is not systematic.
- This chapter presents methods that can be automated in CAD tools.
- Although tools used for logic optimization, designers must understand the process.

### Chapter 4

#### Motivation

- Algebraic manipulation is not systematic.
- This chapter presents methods that can be automated in CAD tools.
- Although tools used for logic optimization, designers must understand the process.

#### Figure 4.1

The function \( f = \sum m(0, 2, 4, 5, 6) \)

<table>
<thead>
<tr>
<th>( K_m )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Figure 4.2

(a) Truth table  
(b) Karnaugh map

#### Figure 2.15

A function to be synthesized

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( f(x_1, x_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Figure 4.4

(a) Truth table  
(b) Karnaugh map

#### Figure 4.4

Location of three-variable minterms
Figure 6: A four-variable Karnaugh map

<table>
<thead>
<tr>
<th>Row number</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>( f(x_1, x_2, x_3, x_4) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row number</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row number</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row number</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row number</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ x_3 \times x_4 \]

\[ x_1 \times x_2 \]

\[ 00 \quad 01 \quad 11 \quad 10 \]

\[ 00 \quad 01 \quad 11 \quad 10 \]

\[ 00 \quad 01 \quad 11 \quad 10 \]

\[ 00 \quad 01 \quad 11 \quad 10 \]
Terminology

- A variable either uncomplemented or complemented is called a *literal*.
- A product term that indicates when a function is equal to 1 is called an *implicant*.
- An implicant that cannot have any literal deleted and still be a valid implicant is called a *prime implicant*.

Minimization Procedure

- Generate all prime implicants.
- Find all essential prime implicants.
- If essential primes do not form a cover, then select minimal set of non-essential primes.
Figure 4.10  Four-variable function $f = \Sigma m(2, 3, 5, 6, 7, 10, 11, 13, 34)$

Figure 4.11  The function $f = \Sigma m(4, 8, 10, 11, 12, 13, 15)$

Figure 4.12  The function $f = \Sigma m(0, 2, 4, 5, 10, 11, 13, 15)$

Minimization of POS Forms

- Find a cover of the 0’s and form maxterms.

Figure 4.13  POS minimization of $f = \Pi M(0, 1, 4, 8, 9, 12, 15)$

Incompletely Specified Functions

- Often certain input conditions cannot occur.
- Impossible inputs are called don’t cares.
- A function with don’t cares is called an incompletely specified function.
- Don’t cares can be used to improve the quality of the logic designed.
Multiple-Output Circuits

- Necessary to implement multiple functions.
- Circuits can be combined to obtain lower cost solution by sharing some gates.

(a) Function $f_1$

(b) Function $f_2$

(c) Combined circuit for $f_1$ and $f_2$
(c) Optimal realization of $f_3$ and $f_4$ together

(d) Combined circuit for $f_3$ and $f_4$

(a) Function $f_1$

(b) Function $f_2$

(a) Optimal realization of $f_3$ (b) Optimal realization of $f_4$

Figure 4.17 An example of multiple-output synthesis

Figure 4.18 DeMorgan’s theorem in terms of logic gates

Figure 4.19 Using NAND gates to implement a sum-of-products
Multilevel Synthesis

- SOP or POS circuits have 2-levels of gates.
- Only efficient for functions with few inputs.
- Many inputs can lead to fan-in problems.
- Multilevel circuits can also be more area efficient.
Impact on Wiring Complexity

- Space on chip is used by gates and wires.
- Wires can be a significant portion.
- Each literal corresponds to a wire.
- Factoring reduces literal count, so it can also reduce wiring complexity.

Functional Decomposition

- Multilevel circuits often require less area.
- Complexity is reduced by decomposing 2-level function into subcircuits.
- Subcircuit implements function that may be used in multiple places.
Example 4.8

\[ f = x_1 \oplus x_2 = x_1 \overline{x_2} + \overline{x_1} x_2 = \overline{x_1} + x_2 + \overline{x_1} x_2 + x_1 \overline{x_2} \]

(a) Sum-of-products implementation

(b) NAND gate implementation

(c) Optimal NAND gate implementation

Figures 4.27–4.29: Implementation of XOR
Practical Issues

- Functional decomposition can be used to implement general logic functions in circuits with built-in constraints.
- Enormous numbers of possible subfunctions leads to necessity for heuristic algorithms.

Figure 4.30 Conversion to a NAND-gate circuit

(a) Circuit with AND and OR gates

(b) Inversions needed to convert to NANDs

Figure 4.31 Conversion to a NOR-gate circuit

(a) Inversions needed to convert to NORs

Figure 4.32 Circuit example for analysis
Figure 4.33 Circuit example for analysis

Figure 4.34 Circuit example for analysis

Figure 4.35 Circuit example for analysis

Figure 4.36 Circuit example for analysis

CAD Tools

- *espresso* – finds exact and heuristic solutions to the 2-level synthesis problem.
- *sis* – performs multilevel logic synthesis.
- Numerous commercial CAD packages are available from Cadence, Mentor, Synopsys, and others.

VHDL code for the function $f = \Sigma m(1, 4, 5, 6)$
Physical Design

- **Physical design** determines how logic is to be implemented in the target technology.
  - **Placement** determines where in target device a logic function is realized.
  - **Routing** determines how devices are to be interconnected using wires.

Timing Simulation

- Functional simulation does not consider signal propagation delays.
- After physical design, more accurate timing information is available.
- **Timing simulation** can be used to check if a design meets performance requirements.
STD_LOGIC type

- Defined in ieee.std_logic_1164 package.
- Enumerated type with 9 values.
  - '1' – strong one
  - '0' – strong zero
  - 'X' – strong unknown
  - 'Z' – high impedance
  - 'H' – weak one
  - 'L' – weak zero
  - 'W' – weak unknown
  - 'U' – uninitialized
  - '-' – don't care
- We will almost always use STD_LOGIC.

Figure 4.47 VHDL code using STD_LOGIC

Figure 4.48 VHDL code for the function $f = \Sigma m(0, 2, 4, 5, 6)$

Figure 4.49 Implementation of the VHDL code for the function $f = \Sigma m(0, 2, 4, 5, 6)$

Figure 4.50 Implementation using XOR synthesis ($f = I_1 \oplus I_2 \oplus I_3$)

Figure 4.51 VHDL code for $f = \Sigma m(0, 2, 4, 5, 6)$ implemented in a LUT
Logic Function Representation

- Truth tables
- Algebraic expressions
- Venn diagrams
- Karnaugh maps
- n-dimensional cubes
n-Dimensional Hypercube

- Function of n variables maps to n-cube.
- Size of a cube is number of vertices.
- A cube with k x’s consists of $2^k$ vertices.
- n-cube has $2^n$ vertices.
- 2 vertices are adjacent if they differ in one coordinate.
- Each vertex in n-cube adjacent to n others.

Using *-operation to Find Primes

- f is specified using a set of cubes, $C^k$ of f.
- Let $c^l$ and $c'^l$ be any two cubes in $C^k$.
- Apply *-operation to all pairs of cubes in $C^k$:
  - $G^{k+1} = c^l * c'^l$ for all $c^l, c'^l$ in $C^k$
- Form new cover for f as follows:
  - $C^{k+1} = C^k \cup G^{k+1}$: redundant cubes
  - A is redundant if exists a $B$ s.t. $A_i = B_i$ or $B_i = x$ for all i.
- Repeat until $C^{k+1} = C^k$.
Finding Essential Primes

- Let $P$ be set of all prime implicants.
- Let $p^i$ denote one prime implicant in $P$.
- Let $DC$ denote the don’t cares vertices for $f$.
- Then $p^i$ is an essential prime implicant iff:
  
  \[ p^i \# (P - p^i) \# DC \neq \emptyset \]

Procedure to Find Minimal Cover

- Let $C^0 = ON \cup DC$ be the initial cover of $f$.
- Find all primes, $P$, of $C^0$ using $*$-operation.
- Find the essential primes using $#$-operation.
- If essentials cover ON-set then done else
  - Delete any nonessential prime that is more expensive than some other prime.
  - Use branching technique to select lowest cost primes which cover ON-set.
Summary

• Described 2-level logic synthesis methods.
• Discussed multilevel logic synthesis.
• Introduced CAD tools for logic synthesis.