Chapter 3
Gate-level Minimization

November 2005
Chapter 3 Gate-level Minimization

3-1 The Map Method
   Two-variable map and Three-variable map
3-2 Four-Variable Map
3-3 Five-variable Map
3-4 Product of Sums Simplification
3-5 Don’t-care Conditions
3-6 NAND and NOR Implementation
3-7 Other Two-Level Implementations
3-8 Exclusive-OR Function
3-9 Hardware Description Language (HDL)
3-1 The Map Method

• Simplification of Boolean Expression
  – Minimum # of terms, minimum # of literals
  – To reduce complexity of digital logic gates
  – The simplest expression is not unique

• Methods:
  – Algebraic minimization ⇒ lack of specific rules
    • Section 2.4
  – Karnaugh map or K-map
    • Combination of 2, 4, … adjacent squares

<table>
<thead>
<tr>
<th>Logic circuit</th>
<th>Boolean function</th>
<th>Truth table</th>
<th>K-map</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇔ Canonical form (sum of minterms, product of maxterms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>⇔ (Simplified) standard form (sum of products, product of sums)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two-variable Map

\[ m_1 + m_2 + m_3 = x'y + xy' + xy = x + y \]
Three-Variable Map

- 8 minterms for 3 binary variables
- Any two adjacent squares differ by only one variable (Gray code; wrap)

$m_0 + m_2 = x'y'z' + x'yz' = x'z'(y' + y) = x'z'$
$m_4 + m_6 = xy'z' + xyz' = xz' + (y' + y) = xz'$

(a)

(b)

Fig. 3-3 Three-variable Map
Examples 3-1 and 3-2

Fig. 3-4  Map for Example 3-1; $F(x, y, z) = \Sigma (2, 3, 4, 5) = x'y + xy'$

Fig. 3-5  Map for Example 3-2; $F(x, y, z) = \Sigma (3, 4, 6, 7) = yz + xz'$
Examples 3-3 and 3-4

One square represents one minterm, giving a term of three literals

- Two adjacent squares represent a term of two literals
- Four adjacent squares represent a term of one literal

Fig. 3-6  Map for Example 3-3; \( F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy' \)

Fig. 3-7  Map for Example 3-4; \( A'C + A'B + AB'C + BC = C + A'B \)
Two adjacent squares represent a term of three literals
Four adjacent squares represent a term of two literals
Eight adjacent squares represent a term of one literal
The larger the number of squares combined, the smaller the number of literals in the term
Examples 3-5 and 3-6

Fig. 3-9 Map for Example 3-5; $F(w, x, y, z)$

$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

Fig. 3-10 Map for Example 3-6; $A'B'C' + B'CD' + A'BCD'$

$A'B'C' = B'D' + B'C' + A'CD'$
Simplification Using Prime Implicants

- **prime implicant**: a product term obtained by combining the *maximum possible number* of adjacent squares in the map
  - A single 1 on a map represents a prime implicant if it is not adjacent to any other 1’s
  - *Two* adjacent 1’s form a prime implicant, provided that they are *not within* a group of *four* adjacent squares
  - *Four* adjacent 1’s form a prime implicant, provided that they are *not within* a group of *eight* adjacent squares
  - and so on
- If a minterm in a square is covered by *only one* prime implicant, that prime implicant is said to be *essential*
First determine all the essential prime implicants
- prime implicant: combining max # of adjacent squares
- essential prime implicant: containing a minterm that is covered by only one prime implicant

The simplified expression is obtained from
- the logical sum of all the essential prime implicants
- plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants
3-3 Five-Variable Map

Fig. 3-12 Five-variable Map
Any $2^k$ adjacent squares, for $k=(0, 1, 2, \ldots, n)$ in an $n$-variable map, will represent an area that gives a term of $n-k$ literals.
Example 3-7

Fig. 3-13  Map for Example 3-7; $F = A'B'E' + BD'E + ACE$
3-4 Product of Sums Simplification

- Boolean functions can be expressed in sum of products or product of sums
  - Recall from Table 2-4
    \[
    f_1 = m_1 + m_4 + m_7 = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6
    = x'y'z + xy'z' + xyz
    = (x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z)
    = \Sigma(1, 4, 7) = \Pi(0, 2, 3, 5, 6)
    \]
    \[
    f_1' = m_0 + m_2 + m_3 + m_5 + m_6 = M_1 \cdot M_4 \cdot M_7
    = x'y'z' + x'yz' + x'yz + xy'z + xyz'
    = (x+y+z')(x'+y+z)(x'+y'+z')
    = \Sigma(0, 2, 3, 5, 6) = \Pi(1, 4, 7)
    \]
  - Complement of f' (by DeMorgan’s Theorem)

- Simplifying Boolean function in product of sums
  1. Derive f’ in sum of products from the map
  2. Complement of f’
Example 3-8

Simply Boolean function \( F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) \) in (a) sum of products and (b) product of sums

(a) \( F = B'D' + B'C' + A'C'D \)

(b) 1. Obtain simplified complemented function:
\( F' = AB + CD + BD' \)

2. Applying DeMorgan’s theorem to obtain \( F \)
\( F = (F')' = (A' + B')(C' + D')(B' + D) \)
Gate Implementation for Example 3-8

\[ F = B'D' + B'C' + A'C'D \]

\[ F = (A' + B')(C' + D')(B' + D) \]

**FIGURE 3-15**
Gate Implementation of the Function of Example 3-8
$F(x, y, z) = \Sigma(1, 3, 4, 6) = \Pi(0, 2, 5, 7)$

$F = x'z + xz'$

$F' = xz + x'z' \implies F = (x' + z')(x + z)$
3-5 Don’t-Care Condition

• In practice, there are some applications where the function is not specified for certain combinations of the variables

• Functions that have unspecified outputs for some input combinations are called incompletely specified functions

• It’s customary to call the unspecified minterms of a function don’t-care conditions
  – marked as X, indicating that we don’t care where 0 or 1 is assigned to F for the particular minterm
  – can be used on a map to provide further simplification
    • may be assumed to be either 0 or 1
Example 3-9

Simplify the Boolean function

\[ F(w, x, y, z) = \sum(1, 3, 7, 11, 15) \]

which has the don’t-care conditions

\[ d(w, x, y, z) = \sum(0, 2, 5) \]

FIGURE 3-17
Example with don’t-care Conditions

\[ F(w, x, y, z) = yz + w'x' = \sum(0, 1, 2, 3, 7, 11, 15) \]
\[ F(w, x, y, z) = yz + w'z = \sum(1, 3, 5, 7, 11, 15) \]
3-6 NAND and NOR Implementation

Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates – easier to fabricate, basic gates used in all IC

**NAND**

\[ F = (xy)' \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**NOR**

\[ F = (x + y)' \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
NAND Circuits

NAND gate: a universal gate
– Any digital system can be implemented with it
• including AND, OR and complement

Fig. 3-18 Logic Operations with NAND Gates

Fig. 3-19 Two Graphic Symbols for NAND Gate
Two-Level Implementation with NAND

sum of product expression and its equivalent NAND implementation

\[ F = AB + CD \]
\[ = [(AB + CD)']' \]
\[ = [(AB)'*(CD)']' \]

Fig. 3-20 Three Ways to Implement \( F = AB + CD \)
Example 3-10
Implement the following Boolean function with NAND gates

\[ F = xy' + x'y + z \]

Fig. 3-21 Solution to Example 3-10
Procedures of Implementation with two levels of NAND gates

1. Express simplified function in sum of products
2. Draw a NAND gate for each product term that has at least two literals to constitute a group of first-level gates
3. Draw a single gate using AND-invert or invert-OR in the second level
4. A term with a single literal requires an inverter in the first level
Multilevel NAND Circuits

1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
3. Check all the bubbles in the diagrams. For a single bubble, insert an inverter (one-input NAND gate) or complement the input literal.

Fig. 3-22 Implementing $F = A(CD + B) + BC$
Figure 3-23 Implementing $F = (AB' + A'B)(C + D')$

(a) AND-OR gates

(b) NAND gates

Fig. 3-23 Implementing $F = (AB' + A'B)(C + D')$
NOR Circuits

• The NOR operation is the dual of the NAND operation
• The NOR gate is another universal gate to implement any Boolean function
• Easy for OR-AND (product of sums)

![Logic Operations with NOR Gates](attachment:image.png)

![Two Graphic Symbols for NOR Gate](attachment:image.png)
NOR Implementation

Transformation from OR-AND diagram to NOR diagram
• OR gates => OR-invert
• AND gate => invert-AND

Fig. 3-26 Implementing $F = (A + B)(C + D)E$

Fig. 3-27 Implementing $F = (AB' + A'B)(C + D')$ with NOR Gates
3-7 Other Two-level Implementations

wired logic: some NAND or NOR gates allow a direct wire connection between the outputs of two gates to provide a specific logic function

- The wired-AND gate or wired-OR gate is not a physical second-level gate, but only a symbol

\[ F = (AB)' \cdot (CD)' = (AB + CD)' \]

\[ F = (A + B)' + (C + D)' = [(A + B)(C + D)]' \]

(a) Wired-AND in open-collector TTL NAND gates.  
(AND-OR-INVERT)

(b) Wired-OR in ECL gates  
(OR-AND-INVERT)

Fig. 3-28 Wired Logic
Nondegenerate Forms

• 16 possible combinations of two-level forms with 4 types of gates: AND, OR, NAND, and NOR
  – 8 degenerate forms: degenerate to a single operation
  – 8 generate forms
    • NAND-AND = AND-NOR = AND-OR-INVERT
    • OR-NAND = NOR-OR = OR-AND-INVERT

<table>
<thead>
<tr>
<th>1st \ 2nd</th>
<th>AND</th>
<th>OR</th>
<th>NAND</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND</td>
<td>3-4</td>
<td>NAND</td>
<td>=</td>
</tr>
<tr>
<td>OR</td>
<td>3-4</td>
<td>OR</td>
<td>x</td>
<td>NOR</td>
</tr>
<tr>
<td>NAND</td>
<td>=</td>
<td>NAND</td>
<td>3-6</td>
<td>AND</td>
</tr>
<tr>
<td>NOR</td>
<td>NOR</td>
<td>x</td>
<td>OR</td>
<td>3-6</td>
</tr>
</tbody>
</table>
AND-OR-INVERT Implementation

AND-NOR = NAND-AND = AND-OR-INVERT

\[ F = (AB + CD + E)' \]

Similar to AND-OR, AND-OR-INVERT requires an expression in sum of products.

Given \( F \), we can implement \( F' \) with AND-OR-INVERT.

---

**FIGURE 3-29**

AND-OR-INVERT Circuits; \( F = (AB + CD + E)' \)
OR-AND-INVERT Implementation

OR-NAND = NOR-OR = OR-AND-INVERT

F = (A+B) (C+D) E ' 

Similar to OR-AND, OR-AND-INVERT requires an expression in products of sum.
Given F, we can implement F' with OR-AND-INVERT

---

FIGURE 3-30
OR-AND-INVERT Circuits; F = [(A + B)(C + D)E]'
Example 3-11
Other Two-level Implementations

\[ F = x'y'z' + xyz' \]
\[ F' = x'y + xy' + z \]

\[ F = (F')' = (x'y + xy' + z)' \]
\[ = (x + y')(x' + y)z \]
\[ F' = (F)' = (x'y'z' + xyz')' \]
\[ = (x + y + z)(x' + y' + z) \]
3-8 Exclusive-OR (XOR) Function

XOR: $x \oplus y = xy' + x'y$

$$
\begin{align*}
 x \oplus 0 &= x \\
 x \oplus 1 &= x' \\
 x \oplus x &= 0 \\
 x \oplus x' &= 1 \\
 x \oplus y' &= x' \oplus y = (x \oplus y)' \\
\end{align*}
$$

Exclusive-NOR = equivalence

$$(x \oplus y)' = (xy' + x'y)'$$
$$= (x' + y)(x + y') = x'y' + xy$$

Communtative: $A \oplus B = B \oplus A$

Associative: $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

Only a limited number of Boolean functions can be expressed in terms of XOR operations, but it is particularly useful in arithmetic operations and error-detection and correction circuits.

Fig. 3-32 Exclusive-OR Implementations
Odd Function

- The 3-variable XOR function is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.
- Multiple-variable exclusive OR operation = odd function: odd number of variables be equal to 1.

\[
A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C
= AB'C' + A'BC' + ABC + A'B'C
= \sum(1, 2, 4, 7)
\]

Fig. 3-33 Map for a Three-variable Exclusive-OR Function

(a) Odd function
\[
F = A \oplus B \oplus C
\]

(b) Even function
\[
F = (A \oplus B \oplus C)'
\]

Fig. 3-34 Logic Diagram of Odd and Even Functions
Figure 3-35 Map for a 4-variable XOR Function

\[ A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D) \]
\[ = (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D) \]
\[ = \sum (1, 2, 4, 7, 8, 11, 13, 14) \]

(a) Odd function
\[ F = A \oplus B \oplus C \oplus D \]

(b) Even function
\[ F = (A \oplus B \oplus C \oplus D)' \]

Fig. 3-35 Map for a Four-variable Exclusive-OR Function
Parity Generation and Checking

parity bit: an extra bit included with a binary message to make the number of 1’s either odd or even

3-bit even-parity-generator

\[ P = x \oplus y \oplus z \]

Table 3-4
Even-Parity-Generator Truth Table

<table>
<thead>
<tr>
<th>Three-Bit Message</th>
<th>Parity Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

FIGURE 3-36
Logic Diagram of a Parity Generator and Checker
4-bit Even-Parity-Checker

\[ C = x \oplus y \oplus z \oplus P \]

**Table 3-5**

<table>
<thead>
<tr>
<th>Four Bits Received</th>
<th>Parity Error Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 3-36**

Logic Diagram of a Parity Generator and Checker

P = 0 \implies 4-bit even parity checker = 3-bit even parity generator
Digital Gates of TTL or CMOS series 7400

Fig. 11-1 Digital Gates in IC Packages with Identification Numbers and Pin Assignments

Fig. 11-1(Cond) Digital Gates in IC Packages with Identification Numbers and Pin Assignments
Put it All Together

Logic circuit
• AND-OR, OR-AND
• NAND-NAND, NOR-NOR
⇔ Truth table
⇔ K-map
⇔ Boolean function
⇔ Canonical form
• sum of minterms, $\Sigma$
• product of maxterms, $\Pi$
⇔ (Simplified) standard form
• sum of products
• product of sums
Problem 3-6

(a) \(A'B'C'D'+AC'D'+B'CD'+A'BCD+BC'D\)

(b) \(x'z+w'xy'+w(x'y + xy')\)

\[\Sigma(0,2,5,7,8,10,12,13)\]  \(= \Pi(1,3,4,6,9,11,14,15)\)

\[\Sigma(1,3,4,5,9,10,11,12,13)\]  \(= \Pi(0,2,6,7,8,14,15)\)
Summary

Chapter 3  Gate-level Minimization
3-1 The Map Method
   Two-variable map and Three-variable map
3-2 Four-Variable Map
3-3 Five-variable Map
3-4 Product of Sums Simplification
3-5 Don’t-care Conditions
3-6 NAND and NOR Implementation
3-7 Other Two-Level Implementations
3-8 Exclusive-OR Function
3-9 Hardware Description Language (HDL)