Lecture 5: Logic Simplification & Karnaugh Maps

Dr Pete Sedcole
Department of E&E Engineering
Imperial College London
http://cas.ee.ic.ac.uk/~nps/

(Floyd 4.5 – 4.10)
(Tocci 4.1 – 4.5)

In this lecture:

- Standard form of Boolean Expression:
  - Sum-of-Products & Product-of-Sums (SOP/POS)
  - Canonical form
- Boolean simplification with Boolean algebra
- Boolean simplification with Karnaugh Maps
- The “Don’t care” logic state

Forms of Boolean Expressions

- Sum-of-products (SOP)
  - e.g.: \( X = ABC + DEF + AEF \)
  - the product (AND) terms are formed first, then summed (OR)
- Product-of-sums (POS)
  - e.g.: \( Y = (A+B+E)(C+D+E)(B+F) \)
  - the sum (OR) terms are formed first, then the product (AND)
- It is possible to convert between the two forms using DeMorgan’s theorems.

Canonical form

- In the canonical form of a Boolean expression, every variable appears in every term
  - e.g.: \( f(A, B, C, D) = ABCD + \overline{A}BCD + \overline{A}BCD \)
- Canonical form is not an efficient way of writing the Boolean expression, but is useful sometimes in design and analysis
• How do we find the canonical form of an expression?
  – Start with the SOP (sum-of-products) form
  – AND each incomplete expression with \( X + \overline{X} \) where
  \( X \) is the missing variable
• e.g.:
  \[
  f(A, B, C) = AB + BC \\
  = AB(C + \overline{C}) + (A + \overline{A})BC \\
  = ABC + AB\overline{C} + \overline{A}BC
  \]
• The product term in a canonical SOP expression is called a ‘minterm’

Canonical form and \( \Sigma \) notation

• Construct the truth table for the function
  \[
  f(A, B, C) = \overline{A}BC + AB\overline{C} + \overline{A}BC
  \]

<table>
<thead>
<tr>
<th>Row number</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

An alternative notation: write \( f \) as a sum of the row numbers that have TRUE minterms:

\[
 f = \Sigma(3,6,7)
\]

Simplifying logic circuits

Obtain the expression of the circuit’s function, then try to simplify

We will look at two methods:

Algebraic and Karnaugh maps

Method 1: Minimisation by Boolean Algebra

• Make use of rules and theorems of Boolean algebra to simplify the Boolean expression
  – Try to reduce the complexity of the equation, so that the circuit is also less complex
  – This method relies on your algebraic skill
  – Three things you can try…
Tool 1: Grouping

- Given: \( A + AB + BC \)
- Write it as: \( A(1 + B) + BC \)
- Then apply: \((1 + B) = 1\)
- Minimised form: \( A + BC \)

Tool 2: ANDing with redundant logic

- ANDing an expression with \((X + \overline{X})\) or \((1 + 1)\) does not alter the logic
- ANDing then expanding may enable an expression to be simplified
- Example:
  \[
  AB + A\overline{C} + BC = AB(C + \overline{C}) + A\overline{C} + BC \\
  = ABC + AB\overline{C} + A\overline{C} + BC \\
  = BC(1 + A) + A\overline{C}(1 + B) \\
  = BC + A\overline{C}
  \]

Tool 3: DeMorgan’s Theorems

- Inversions of compound expressions can be simplified using DeMorgan’s Theorems
- Example:
  \[
  \overline{ABC} + \overline{ACD} + BC = (A + B + \overline{C}) + (A + \overline{C} + D) + \overline{B}C \\
  = A\overline{B}CD
  \]

Example of logic design

- Design a logic circuit having three inputs (A, B, C) that will have its output HIGH only when a majority of the inputs are HIGH
- Step 1: Draw the truth table
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
- Step 2: Write the product terms for each case where the output is 1
  \( \overline{A}\overline{B}C \) for \((0, 0, 0)\)
  \( \overline{A}BC \) for \((0, 0, 1)\)
  \( \overline{A}\overline{B}C \) for \((0, 1, 0)\)
  \( \overline{A}BC \) for \((0, 1, 1)\)
  \( \overline{A}BC \) for \((1, 0, 0)\)
  \( \overline{A}\overline{B}C \) for \((1, 0, 1)\)
  \( \overline{A}BC \) for \((1, 1, 0)\)
  \( \overline{A}BC \) for \((1, 1, 1)\)
Step 3:
Write the SOP form for the output

\[ Z = \overline{ABC} + \overline{A}BC + ABC \]

Step 4:
Using the rules of Boolean algebra, try to simplify the expression

\[ Z = \overline{ABC} + \overline{A}BC + ABC \]
\[ = \overline{ABC} + \overline{ABC} + ABC + ABC \]
\[ = BC(A + \overline{A}) + AC(B + \overline{B}) + AB(C + \overline{C}) \]
\[ = BC + AC + AB \]

Minimisation using Karnaugh Maps

- What is a Karnaugh map?
  - A grid of squares
  - An example with three variables:
  - Similar to a truth table: shows the output value for every combination of inputs
  - Each square represents a minterm
  - Only one variable changes between adjacent squares (similar to Gray Codes)
  - Squares at the edges are adjacent to squares on the opposite edges

An example with 4 variables:

The square marked ? represents:

The square marked ?? represents:

Note that they differ only in the C variable
Filling out a Karnaugh Map

- Write the Boolean expression in SOP form
- For each product term, write a 1 in all the squares which are included in the term
- Write a 0 in the remaining blank squares
- Example:

\[ X = \overline{A}BC + AB\overline{C} + AB \]

<table>
<thead>
<tr>
<th>A (\overline{BC})</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Minimisation technique

- Minimisation is done by spotting patterns of 1s and 0s
- The rules of algebra can be used to describe the patterns in simple product terms
- For example, take a pair of adjacent 1s:
  - Adjacent squares differ by one variable
  - The expression for a pair of adjacent 1s has the form:
    \[ (P)C + (P)\overline{C} = (P)(C + \overline{C}) \]
    This can be simplified to just \(P\)

- The example from an earlier slide:
  - The adjacent squares \(\overline{A}BC\) and \(ABC\) differ only in \(A\)
    - Therefore they can be combined into just \(BC\)
    - Normally we draw boxes around the squares to be combined
  - Can do this for larger groups of 1s (of 4, 8, …)

\[ X = \overline{A}BC + AB\overline{C} + AB \]

<table>
<thead>
<tr>
<th>A (\overline{BC})</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- “Cover” all the 1s with the largest groups
- The simplified Boolean expression is found by summing all the terms corresponding to each group

\[ X = AC + BC + AB \]
More examples of grouping

\[ \begin{align*}
ABCD & 00 & 01 & 11 & 10 \\
00 & 0 & 1 & 0 & 0 \\
01 & 0 & 1 & 0 & 0 \\
11 & 1 & 1 & 1 & 1 \\
10 & 0 & 1 & 1 & 0 \\
\end{align*} \]

\[ AB + \overline{CD} \]

\[ \begin{align*}
ABCD & 00 & 01 & 11 & 10 \\
00 & 0 & 0 & 0 & 0 \\
01 & 1 & 0 & 0 & 1 \\
11 & 1 & 0 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 \\
\end{align*} \]

\[ B\overline{D} + ABC \]
K-map simplification

- Goal: find the form of a Boolean expression with the minimum number of product terms
- Method: group together 1s on a K-map
- Constraints:
  - all 1s must be part of a group (even if it is a single group)
  - all groups must be of size 1, 2, 4, 8, 16, ...

Complete K-map simplification process

1. Construct the K-map and place 1s and 0s in the squares according to the SOP expression or truth table
2. Find the largest grouping of 1s that are not already all in a group; if there is more than one possibility, choose a grouping that minimises the total number of groups
3. Repeat step 2 until only isolated 1s remain
4. Form single groups of the remaining isolated 1s
5. Find the product term that corresponds to each group
6. OR together all the product terms

'Don’t Care' conditions

- In certain cases some input combinations may never occur, or the output doesn’t matter when they do
- In this case we fill in the truth table / K-map with an ‘X’
- ‘X’ means ‘don’t care’ (it is not a Boolean variable)
- ‘X’ acts like a joker or wildcard – it can be either a 1 or a 0 depending on which value will help the minimisation

Example:

<table>
<thead>
<tr>
<th>X</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>X = 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ Z = B \]
More ‘don’t care’ examples

- (a) $A B C D = 0000$
- (b) $A B C D = 0001$
- (c) $A B C D = 0011$
- (d) $A B C D = 0111$

Open the lift doors when the lift is stopped at a floor

The Karnaugh map of 5 variables

K-map method summary

- Compared to algebraic manipulation, the K-map method is a more structured process requiring a fixed number of steps
- K-maps always produce a minimum expression
- The minimum expression is in general NOT unique
- For circuits with a large number of inputs (more than 4) other, more complex techniques must be used
Summary

- SOP and POS are useful forms of Boolean equations
- Designing a combinational logic circuit:
  1. Construct a truth table
  2. Convert it to SOP
  3. Simplify using Boolean algebra or a K-map
  4. Implement
- A Karnaugh map is a graphical method for representing and simplifying Boolean expressions
- "Don’t care" entries in a K-map can take values of 1 or 0 depending on which value is more helpful in the simplification