# Combinatorial Logic

## CIS008-2 Logic and Foundations of Mathematics

David Goodwin

david.goodwin@perisic.com


10:00, Monday 26th March 2012

# Outline

# Outline

# Introduction

Boolean Algebras and combinatorial logic are indispensible tools for the analysis of electrical circuits. Although we will not be concerned with the entirity of electrical engineering and it's connection to logic gates, we will be concerned with the basis of the mathematics that uses symbols instead of words to describe logic (from the work of George Boole).
The basis of the mathematics is that a computer can have one of two values, symbolised in one of many ways

- 0 or 1
- On or Off
- True of False
- Open or Closed
- $+$ or $-$

# Definitions

We can consider a logic gate from elecrical engineering to take inputs and producing outputs having the same truth tables as symbolic logic earlier in the course:

- AND

$$x_1 \wedge x_2 = \begin{cases} 1 & \text{if } x_1 = 1 \text{ and } x_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

- OR

$$x_1 \vee x_2 = \begin{cases} 1 & \text{if } x_1 = 1 \text{ or } x_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

- NOT

$$\bar{x}_1 = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$$

Combinatorial logic simply takes combinations of these basic building blocks, to represent a circuit of logic gates.

# Outline

# Laws for combinatorial logic

## Associative laws

$$(a \vee b) \vee c = a \vee (b \vee c)$$
$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$

## Commutative laws

$$a \vee b = b \vee a$$
$$a \wedge b = b \wedge a$$

## Distributive laws

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

## Identity laws

$$a \vee 0 = a$$
$$a \wedge 1 = a$$

## Complement laws

$$a \vee \bar{a} = 1$$
$$a \wedge \bar{a} = 0$$

# De Morgan's Laws

Just as in earlier in the course, De Morgan's laws for logic also apply here:

1. $$\overline{(x \vee y)} = \bar{x} \wedge \bar{y}$$
2. $$\overline{(x \wedge y)} = \bar{x} \vee \bar{y}$$

# Outline

# Boolean Algrbra

We can define a boolean algebra following the same laws as that of combinatorial logic, except we symbolise operators differently.

$$\vee \rightarrow +$$

$$\wedge \rightarrow \cdot$$

$$\bar{x} \rightarrow x'$$

# Boolean Algrbra

Besides the laws for combinatorial logic applying to boolean algebra, there are further laws that define boolean algebra:

## Idempotent laws

$$x + x = x$$
$$x \cdot x = x$$

## Involution laws

$$(x')' = x$$

## Bound laws

$$x + 1 = 1$$
$$x \cdot 0 = 0$$

## 0 and 1 laws

$$0' = 1$$
$$1' = 0$$

## Absorption laws

$$x + x \cdot y = x$$
$$x \cdot (x + y) = x$$

## De Morgan's laws

$$(x + y)' = x' \cdot y'$$
$$(x \cdot y)' = x' + y'$$

University of Bedfordshire

# Outline

# Boolean functions

Functions that can be represented by Boolean expressions are called **Boolean functions**. Let $X(x_1, \ldots, x_n)$ be a Boolean expression. A function $f$ of the form

$$f(x_1, \ldots, x_n) = X(x_1, \ldots, x_n)$$

is called a Boolean tunction.

## Example

The function $f : Z_2^3 \to Z_2$ defined by $f(x_1, x_2, x_3) = x_1 \wedge (\bar{x}_2 \vee x_3)$ is a Boolean function. The inputs and outputs are given in the following table:

| $x_1$ | $x_2$ | $x_3$ | $f(x_1, x_2, x_3)$ |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 |

# Minterm & Maxterm

A **minterm** in the symbols $x_1, \ldots, x_n$ is a Boolean expression of the form $y_1 \wedge y_2 \wedge \cdots \wedge y_n$ where each $y_i$ is either $x_i$ or $\bar{x}_i$.

## Theorem

*If $f : Z_2^n \to Z_2$, then $f$ is a Boolean function. If $f$ is not identically zero, let $A_1, \ldots, A_k$ denote the elements $A_i$ of $Z_2^n$ for which $f(A_i) = 1$. For each $A_i = (a_1, \ldots, a_n)$, set $m_i = y_1 \wedge \ldots y_n$ where*

$$y_j = \begin{cases} x_j & \text{if } a_j = 1 \\ \bar{x}_j & \text{if } a_j = 0 \end{cases}$$

*Then $f(x_1, \ldots, x_n) = m_1 \vee m_2 \vee \cdots \vee m_k$. This representation of a Boolean function is called the* **disjunctive normal form** *of a function.*

The previous theorem for the minterm has a dual, where $f(x_1, \ldots, x_n) = M_1 \wedge M_2 \wedge \cdots \wedge M_k$ for each $M_i$ having the form $y_1 \vee \cdots \vee y_n$ called a **maxterm**. This functional representation is called the **conjunctive normal form**.

# Outline

# Minimisation and Karnaugh Maps

In combinatorial logic minimization, a device known as a Karnaugh map[1] is frequently used. It is similar to a truth table, but the various variables are represented along two axes and are arranged in such a way that only one input bit changes in going from one square to an adjacent square. It is also known as a Veitch diagram, K-map, or KV-map.

The Karnaugh map may be used to quickly eliminate redundant operations in a Boolean function. The easiest to read Karnaugh maps are those drawn for a function in the form of a complete product or "sum of products," where the latter name also implies the use of $\cdot$ and $+$ for the AND and OR operators. In a typical truth table for such a function, the inputs are enumerated using 0 for false and 1 for true, and ordered as a counting sequence when read as positive binary integers.

[1] Weisstein, Eric W. and Wilson, Stuart. "Karnaugh Map." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/KarnaughMap.html

# Minimisation and Karnaugh Maps

A truth table for a function of four variables is illustrated below.

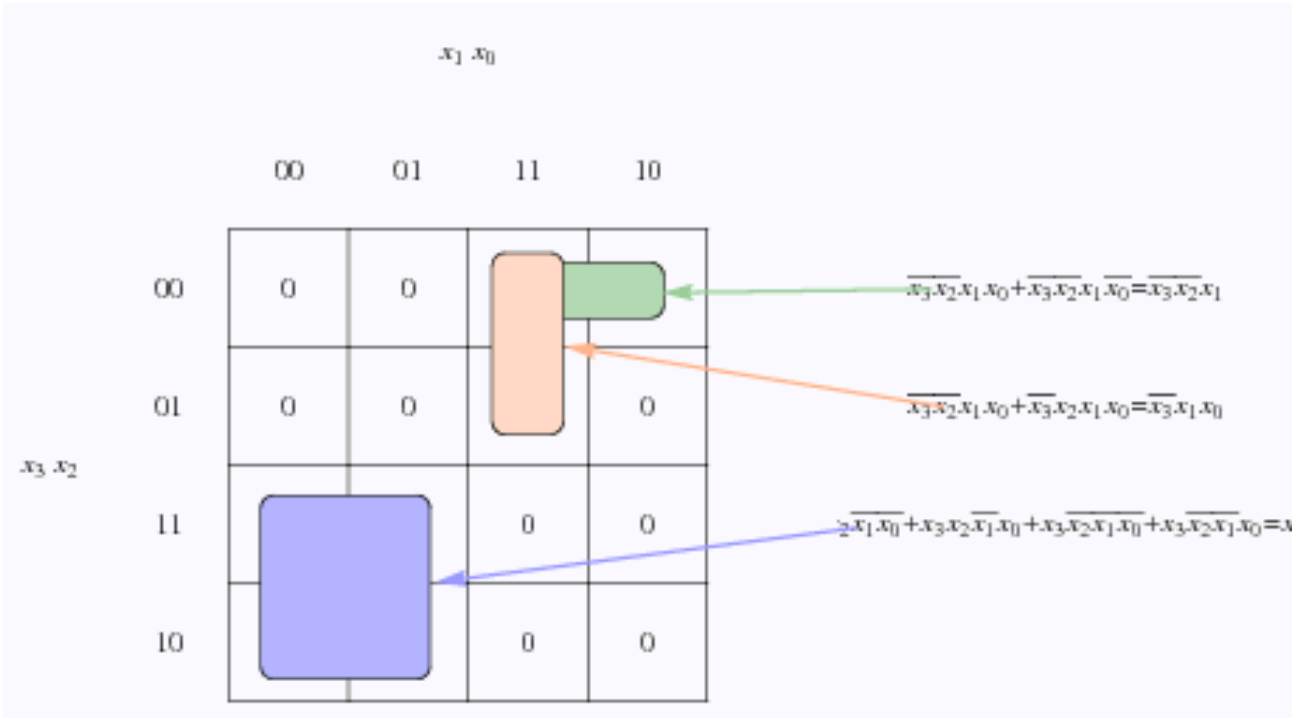
| $x_1$ | $x_2$ | $x_3$ | $x_3$ | $f$ | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 0 | 1 | $\bar{x}_1\bar{x}_2x_3\bar{x}_4$ |
| 0 | 0 | 1 | 1 | 1 | $\bar{x}_1\bar{x}_2x_3x_4$ |
| 0 | 1 | 0 | 0 | 0 | |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 1 | $\bar{x}_1x_2x_3\bar{x}_4$ |
| 0 | 1 | 1 | 1 | 1 | $\bar{x}_1x_2x_3x_4$ |
| 1 | 0 | 0 | 0 | 1 | $x_1\bar{x}_2\bar{x}_3\bar{x}_4$ |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 0 | 0 | |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 0 | 1 | $x_1x_2\bar{x}_3\bar{x}_4$ |
| 1 | 1 | 0 | 1 | 1 | $x_1x_2\bar{x}_3x_4$ |
| 1 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | 0 | |

# Minimisation and Karnaugh Maps

For those rows in the table where the function value is 1 (True), a logical expression called a minterm is shown. The minterms use the overbar notation to mean NOT. When all seven minterms are accumulated,

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2x_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3x_4 + \bar{x}_1x_2x_3\bar{x}_4 + \bar{x}_1x_2x_3x_4 + x_1\bar{x}_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3x_4$$

the function is realized. This realization is not optimal, and a Karnaugh map can be used to reduce it.




University of Bedfordshire

# Minimisation and Karnaugh Maps

A Karnaugh map for the function is shown above. It is a two dimensional layout of the truth table. Each dimension spans an adjacent, though not necessarily so, pair of variables. Instead of a counting sequence, the variables' sequence is in Gray code, so that between each pair of adjacent cells (in rows or columns) only a single variable changes state.

On the map, neighboring cells where the function value is 1 are grouped together with loops. Each loop represents minterms which can be reduced. Those variables which change within a loop can be eliminated. The validity of this, using the uppermost expression as an example, can be shown algebraically.

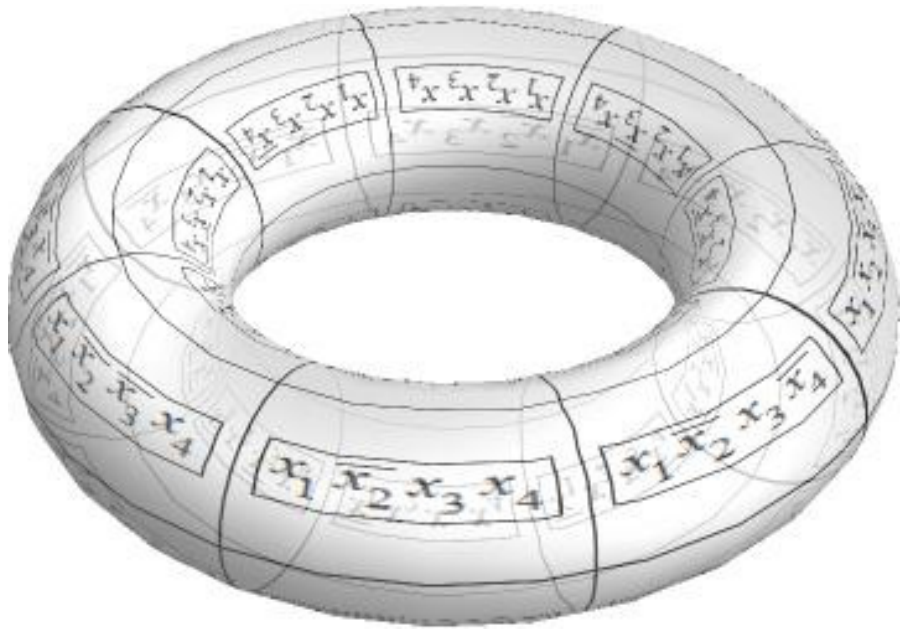
$$
\begin{aligned}
& x_1 x_2 \bar{x}_3 \bar{x}_4 + x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \\
=& (x_1 \bar{x}_3 \bar{x}_4)(x_2 + \bar{x}_2) \\
=& x_1 \bar{x}_3 \bar{x}_4
\end{aligned}
$$

# Minimisation and Karnaugh Maps

Loops can also be drawn wrapping around rows and/or columns, because the Karnaugh map (up to four variables) is really the surface of a torus.

Karnaugh maps such as in the above example can be similarly drawn for two (degenerate) or three variables. They can be employed for more than four variables (e.g., five variables by overlaying two four-variable maps), but visualization becomes a much greater chore than algebra.

There are other logic design and minimization methods based on the Karnaugh map, such as for using NAND operators instead of AND and OR.




University of Bedfordshire