CSD1100

Boolean Algebra

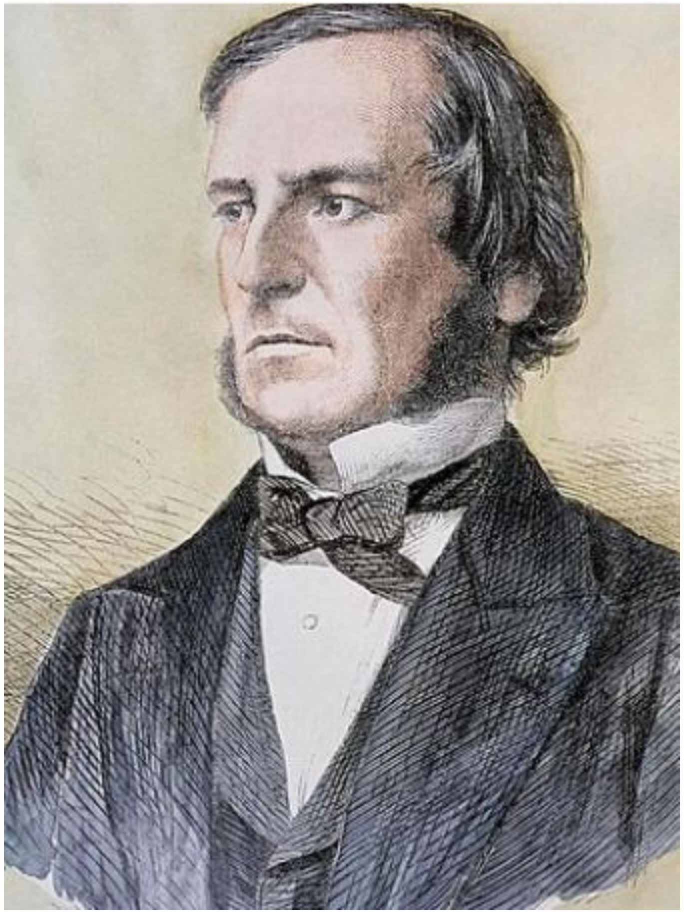
Vadim Surov

Introduction

- Objectives for next few weeks
  - Understand the relationship between Boolean logic and digital computer circuits.
  - Learn how to design simple logic circuits.
  - Understand how digital circuits work together to form complex computer systems.

Introduction

- In the nineteenth century George Boole suggested that **logical thought could be represented through mathematical equations**.
- Computers, as we know them today, are implementations of **Boole's Laws of Thought**.



Introduction

- In the middle of the twentieth century, computers were commonly known as “thinking machines” and “electronic brains.”
- Nowadays, we rarely ponder the relationship between electronic digital computers and human logic. Computers are accepted as part of our lives.
- Next few weeks, you will learn the simplicity that constitutes the essence of the machine.

Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values:
  - **true** and **false**
- true AKA 1, on, high (>3v), any sequence of bits when at least 1 bit is 1.
- false AKA 0, off, low (<1v), any sequence of 0s

Boolean Algebra

- Boolean expressions are created by performing operations on Boolean variables.
- Common Boolean operators include
  - AND (&, ^(caret), *(multiplication))
  - OR (|, v, +)
  - NOT (¬ (elbow), ‾(overbar), ‘(prime mark))

Boolean Operators

- A Boolean operator can be completely described using a **truth table**.
- In the table, X and Y are variables.
- Truth tables for other operators on next slide

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Operators

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

Boolean Operators

- The AND operator is also known as a Boolean product:

  x AND y $x \cdot y$ or just xy
- The OR operator is the Boolean sum or addition:

  x OR y $x+y$
- The NOT operation is most often designated by an overbar when writing. When typing a prime mark ‘ is used instead.

  NOT x $\overline{x}$ x'

Boolean Expressions. Example 1

- Boolean expression is a logical statement that is either 1 or 0.

  Ex: x AND NOT z OR y

  $xz'+y$

- To make evaluation of the Boolean expression easier, the truth table contains extra columns to hold evaluations of subparts of the expression.

Boolean Expressions. Example 1. xz'+y

x	y	z	z'	xz'	xz'+y
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Boolean Expressions. Example 2. XOR

- $x \cdot y' + x' \cdot y$
- Also known as boolean operator XOR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Expressions And Precedence

- As with common arithmetic, Boolean operations have **rules of precedence**. This is how we chose the expression subparts during evaluation.
- The NOT operator has highest priority, followed by AND and then OR.

Boolean operator	Priority
NOT	1 (highest)
AND	2
OR	3 (lowerst)

Boolean Expression Simplification

- Digital computers contain circuits that implement Boolean logic.
- The simpler that we can make a Boolean expression, the smaller the circuit that will result.
- With this in mind, we always want to reduce our Boolean expressions to their simplest form.
- There are a number of **Boolean identities** that help us to do this.

Boolean Identities

Logical Inverse	0’ = 1 1’ = 0
Involution / Double Complement	A’’ = A

Boolean Identities

Dominance	$A+1=1$	$A \cdot 0=0$
Identity	$A+0=A$	$A \cdot 1=A$
Idempotence	$A+A=A$	$A \cdot A=A$
Complementarity	$A+A'=1$	$A \cdot A'=0$
Commutativity	$A+B=B+A$	$A \cdot B=B \cdot A$

Boolean Identities

Associativity	$(A+B)+C = A+(B+C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
Distributivity	$A+(B \cdot C)=(A+B) \cdot (A+C)$	$A \cdot (B+C)=(A \cdot B)+(A \cdot C)$
Absorption	$A \cdot (A+B) = A$	$A+(A \cdot B) = A$
DeMorgan's	$(A+B)' = A' \cdot B'$	$(A \cdot B)' = A'+B'$

Canonical Forms

- Through our exercises in simplifying Boolean expressions, we see that there are numerous ways of stating the same Boolean expression.
  - These “synonymous” forms are **logically equivalent**.
  - Logically equivalent expressions have identical truth tables.
- In order to eliminate as much confusion as possible, designers express Boolean expressions in **standardized** or **canonical form**.

Canonical Forms

- There are two canonical forms for Boolean expressions: **Sum-Of-Products (SOP)** and **Product-Of-Sums (POS)**.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In **SOP** form, AND'ed variables are OR'ed together.
  - For example: $x \cdot y + x \cdot z + y \cdot z$
- In **POS** form, OR'ed variables are AND'ed together:
  - For example: $(x+y) \cdot (x+z) \cdot (y+z)$

Sum-Of-Products

- Inspect the truth table and start from the first row.
- For all the input variables in a given row whose output is 1:
  - If the value of variable P is 1, then write P
  - If the value of variable P is 0, then write P'
- Connect all the input variables in the row with the ' · ' operator.
- Repeat for all the rows in the truth table where the output is 1.
- When all rows (with output =1) have been translated to Boolean expressions, connect these expressions with the '+' operator

SOP Example

$$x' \cdot y \cdot z' +$$
$$x' \cdot y \cdot z +$$
$$x \cdot y' \cdot z' +$$
$$x \cdot y \cdot z' +$$
$$x \cdot y \cdot z$$

We note that this expression is not in simplest terms. Our aim is only to rewrite our function in canonical SOP form.

x	y	z	xz'+y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Product-Of-Sum

- Inspect the truth table and start from the first row.
- For all the input variables in a given row whose output is 0:
  - If the value of variable P is 1, then write P’
  - If the value of variable P is 0, then write P
- Connect all the input variables in the row with the '+' operator.
- Repeat for all the rows in the truth table where the output is 0.
- When all rows (with output =0) have been translated to Boolean expressions, connect these expressions with the '·' operator

POS Example

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

We note that this expression is not in simplest terms. Our aim is only to rewrite our function in canonical POS form.

x	y	z	xz'+y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1