The Building Blocks: Binary Numbers, Arithmetic, Boolean Logic and Gates

Topics:

Boolean Logic
Gates and Circuits

Prof. Dr. Slim Abdennadher

17.1.2008

Quick Review

You need to know:

- **Floating point** numbers (positive or negative)
- **Unsigned** integer numbers (single representation, range depends on number of bits used, minimum numbers is 0)
- **Signed** integer numbers (**positive** numbers have the same representation as unsigned numbers, **negative** numbers are in 2's complement representation, a 1 at the left most bit indicates that the number is negative)
- All **subtractions** are converted into addition ($A - B = A + (-B)$)

Boolean Logic

- **Boolean logic** is a branch of mathematics that deals with rules for manipulating two logical values **true** and **false**.
- Named after George Boole (1815-1864)
- Why is Boolean logic so relevant to computers?
  - Straightforward mapping to binary digits!
  - **0** is **false**
  - **1** is **true**

Boolean Expression

- **Boolean expression** is any expression that evaluates to either true or false (ex: $X = 5$, $2 < 4$).
- Boolean expressions are widely used in programming. They are formed from **variables** and **operations**.
- **Variables** are designated by letters, ex. A,B,C,X,Y,.... Each variable takes only one of 2 values: **0** or **1**.
- The three basic **operations** are:

Operation:	AND (product) of two inputs	Or (sum of) two inputs	NOT (complement) of one input
Expression:	xy or $x * y$	$x + y$	x' or $\bar{x}$ or $-x$

Basic Boolean Operations

- Boolean expressions are created from three basic **operations**

Operation:	AND (product) of two inputs	Or (sum of) two inputs	NOT (complement) of one input
Expression:	xy or $x * y$	$x + y$	x' or $\bar{x}$ or $-x$

Truth Table:

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

Logical AND

- The AND takes two expressions for input, e.g. A, B
- It evaluates to **TRUE** only if both expressions are TRUE
- Written as $A * B$ or AB
- **Example:**

  $A =$ It is sunny

  $B =$ I am in vacation

  $(A * B) =$ It is sunny **AND** I am in vacation

A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR

- The OR takes two expressions for input, e.g. A, B
- It evaluates to**TRUE** if either A is TRUE or B is TRUE or both expressions are TRUE
- Written as $A + B$
- **Example:**

$A =$ It is sunny

$B =$ I am in vacation

$(A + B) =$ It is sunny **OR** I am in vacation **OR** both

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Logical NOT

- The NOT takes only one expression for input, e.g. A
- It is used to invert a meaning
- Written as A'
- **Example:**

  $A =$ It is sunny

  $A' =$ It is not sunny

A	A’
0	1
1	0

Boolean Operations are special

- The **AND** and **OR** are **similar** to multiplication and addition.
  - **AND** yields the same results as multiplication for the values 0 and 1.
  - **OR** is almost the same as addition, expect for the case $1 + 1$.

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

- This explains why we borrow the arithmetic symbols $*, +, 0$ and 1 for Boolean operations.
- But there are **important differences** too.
  - There are a finite number of Boolean values: 0 and 1.
  - **OR** is not quite the same as addition.
  - **NOT** is a new operation.

Boolean Expressions

- Using the basic operations, we can form more complex expressions

$$f(x, y, z) = (x + y')z + x'$$

- **Terminology and notation**
  - f is the **name** of the function
  - x, y and z are **input variables**, which range over 0 and 1.
  - A **literal** is any occurrence of an input or its complement.
- **Precedences** are important.
  - **NOT** has the highest precedence, followed by **AND**, and then **OR**.
  - Fully parenthesized, the expression above would be written:

$$f(x, y, z) = (((x + (y'))z) + x')$$

Truth Tables

- A truth table represents all possible values of an expression given the possible values of its inputs.
- How do we build a truth table?
  - **Step 1.** Create columns for all variables
  - **Step 2.** Determine the number of rows needed (how many rows should appear?)
    - For n variables, 2^n rows.
  - **Step 3.** Define all possible values for the inputs starting from all **0's** to all **1's**, e.g. for 3 input variables from 000 to 111
  - **Step 4.** Find the value of the expression for each input value and fill in the table.

Truth Tables – Example

$$f(x, y, z) = (x + y')z + x'$$

x	y	z	$f(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

How to translate a Truth Table to a Boolean Expression?

- **Idea:**

A	B	Output
0	0	0
0	1	0
1	0	1
1	1	0

- **Sum-of-Products-Algorithm:**
  - Form AND terms for each row that has 1 as the expected
    - use x if it corresponds to $x = 1$
    - use x' if it corresponds to $x = 0$
  - OR the terms together
- The resulting expression then represents the complete functionality of the truth table.

Sum-of-Products-Algorithm – Example

x	y	z	$f(x,y,z)$	
0	0	0	**1**	$\leftarrow$
0	0	1	**1**	$\leftarrow$
0	1	0	**1**	$\leftarrow$
0	1	1	**1**	$\leftarrow$
1	0	0	0	
1	0	1	**1**	$\leftarrow$
1	1	0	0	
1	1	1	**1**	$\leftarrow$

$$f(x,y,z) \;=\; x'y'z' \;+\; x'y'z \;+\; x'yz' \;+\; x'yz \;+\; xy'z \;+\; xyz$$

Expressions and Circuits

- A **circuit** is a network of gates that implements one or more boolean functions.
- We can build a circuit for any Boolean expression by **connecting primitive logic gates** in the correct order.
- Notice that the order of operations is explicit in the circuit.

Primitive Logic Gates

- A **gate** is an electronic device that operates on a collection of binary inputs to produce a binary output.
- Each basic operation can be implemented in hardware with a logic gate.

Operation:	AND (product) of two inputs	Or (sum of) two inputs	NOT (complement) of one input
Expression:	XY or $X * Y$	$X + Y$	X' or $\bar{X}$ or $-X$

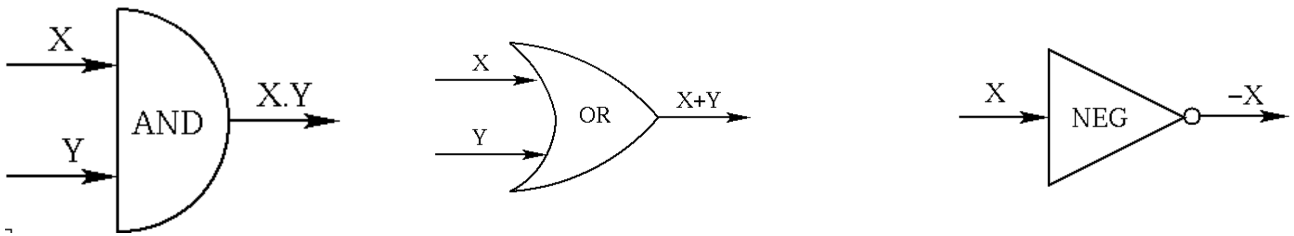
Truth Table:

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	$X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

X	X'
0	1
1	0

Logic Gate:

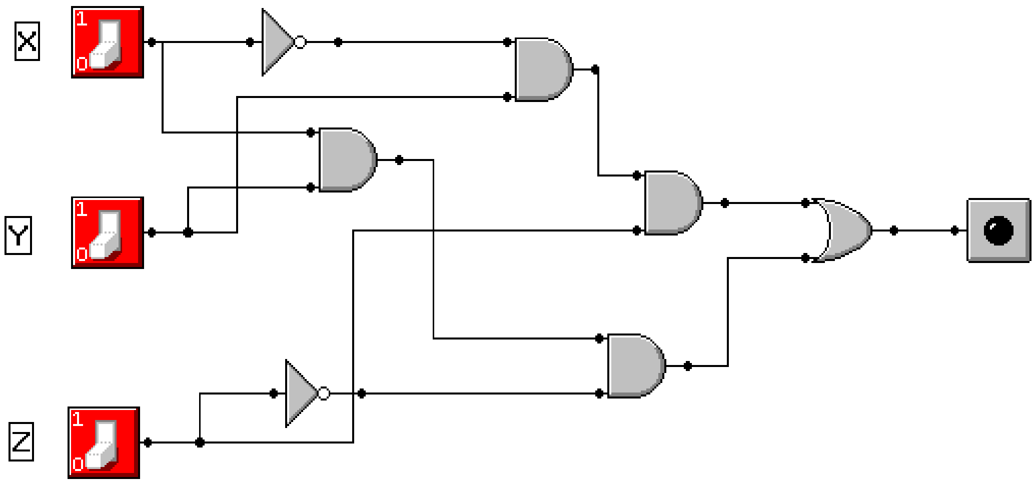


Expressions and Circuits – Example

- **Truth table:**

X	Y	Z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Circuit:



- **Boolean Expressions:**

$$S = X' * Y * Z + X * Y * Z'$$

Equivalence Proof with Truth Tables

- Two expressions are **equivalent** by showing that they always produce the same results for the same inputs
- **Example:** $(x+y)' = x'y'$

x	y	$x+y$	$(x+y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x	y	x'	y'	$x'y'$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0