Boolean Logic and Truth Tables

Computer programs use logic to control execution by comparing operands. In the real world control and embedded systems frequently deal with individual bits in order to control specific operations or to determine the condition of part of a system. For example, a bit might be turned on to light a lamp or activate a relay, or a bit might be off to indicate a switch is on. Both systems are based on Boolean logic.

Boolean logic, developed by George Boole (1815-1864), is simply **a way of comparing operands or bits and determining their truth value**. It uses what are called **operators** to determine how the bits are compared. They simulate the **gates** that you will see in the hardware section you will read shortly.

Think of operators as boxes with multiple inputs and one output. Feed in various combinations of bit values, and the output will be high or low depending on the type of operation. The examples show 2 inputs, although gates can have more. Also, gates are often combined to form more complex logic. A modern microprocessor contains huge numbers of them with many inputs and many varying combinations. Please note that the terms **on**, **high** and **1** will be considered the same logical state, and **off**, **low** and **0** will be considered the same logical state in the discussions that follow. Typically **1 is used to indicate true and 0 is used to indicate false** and in the C programming language true is actually any value not equal to 0 and this can be used in powerful ways.

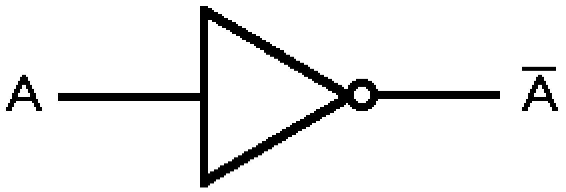
The Boolean operators used most often are **AND** and **OR and NOT**. The **AND** operation says if and only if **all** inputs are on, the output will be on. The output will be off if any of the inputs are off. The **OR** operation says if **any** input is on, the output will be on. The **NOT** operator negates the input (e.g. True ➔ False and False ➔ True). It's easy to see all of the combinations by using what are called **Truth Tables**, illustrated below. At the bottom of each table is shown the schematic symbol found in circuit diagrams.

Basic Logic Gates

There are three basic logic gates each of which perform a basic logic function, they are called NOT, AND and OR. All other logic functions can ultimately be derived from combinations of these three. For each of the three basic logic gates a summary is given including the *logic symbol* , the corresponding *truth table* and the *Boolean expression*.

The NOT gate

The NOT gate is unique in that it only has one input. It looks like



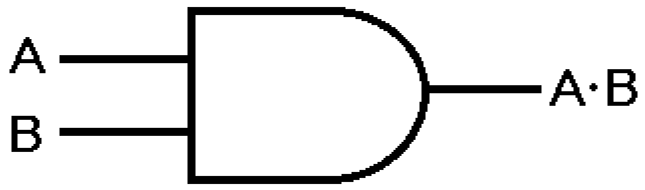
The input to the NOT A gate is *inverted* i.e the binary input state of 0 gives an output of 1 and the binary input state of 1 gives an output of 0.

$\overline{A}$ is known as "NOT A" or alternatively as the *complement* of A. The truth table for the NOT gate appears as below

A	$\overline{A}$
0	1
1	0

The AND gate

The AND gate has two or more inputs. The output from the AND gate is 1 if and only if all of the inputs are 1, otherwise the output from the gate is 0. The AND gate is drawn as follows



The output from the $A \cdot B$ AND gate is written as (the dot can be written half way up the line as here or on the line. Note that some textbooks omit the dot completely).

The truth table for a two-input AND gate looks like

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

It is also possible to represent an AND gate with a simple analogue circuit.

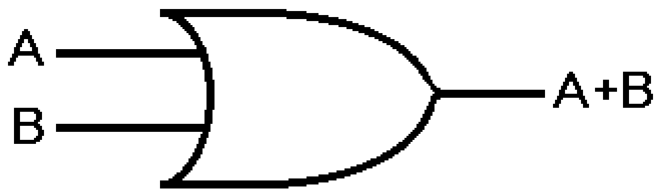
Short Circuit Evaluation

We and the computer always strive for efficiency. Have a look at the above truth table. Is there any condition where we can evaluate just a single operand (input) to determine the result thereby saving the computation time to evaluate the 2nd operand in conjunction with the 1st operand?

Note the AND operator is also known as logical conjunction and is also represented by ^

The OR gate

The OR gate has two or more inputs. The output from the OR gate is 1 if any of the inputs is 1. The gate output is 0 if and only if all inputs are 0. The OR gate is drawn as follows



The output from the OR A+B. gate is written as

The truth table for a two-input OR gate looks like

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

Short Circuit Evaluation
Have a look at the above truth table. Is there any condition where we can evaluate just a single operand (input) to determine the result thereby saving the computation time to evaluate the 2nd operand in conjunction with the 1st operand?

Note the OR operator is also known as logical disjunction and is also represented by v

Desk Check Example

Set A equal to 7
Set B equal to 6
Set C equal to 13
Set D equal to 4

What is the truth value? Hint, link up the parenthesis, create memory locations as you would do in a desk check and work left to right adhering to mathematical precedence. The answer has been whited out so you may change the font of the whitespace to black to reveal the answer

If (NOT ((B < A) AND ((C > D) OR (D == A))))

Answer:

Now this one which has some bad programming practices but is good for demonstration. There are several interpretations or uses of the following code:

> **In Boolean Algebra, 1 is true and 0 is false**
>
> **In Linux Scripting an exit code of 0 is a successful exit whereas anything else is an exit with errors.**

If ((B == (A – (C > D))) AND (NOT (((A + B) == C) OR (NOT (True == False)))))

Answer:

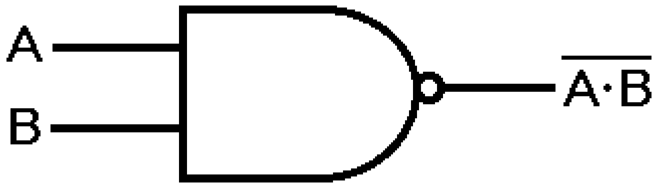
Optional

Other Logic Gates

The three basic logic gates can be combined to provide more complex logical functions. Four important logical functions are described here, namely NAND, NOR, XOR and XNOR. In each case a summary is given including the *logic symbol* for that function, the corresponding *truth table* and the *Boolean expression*.

The NAND gate

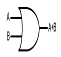
The NAND gate has two or more inputs. The output from the NAND gate is 0 if and only if all of the inputs are 1 otherwise the output is 1. Therefore the output from the NAND gate is the NOT of A AND B (also known as the *complement* or *inversion* of $A \cdot B$). The NAND gate is drawn as follows



where the small circle immediately to the right of the gate on the output line is known as an *invert bubble*.

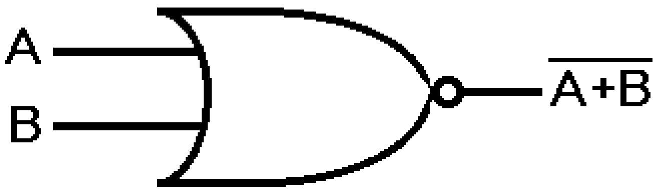
The output from the NAND gate is written as $\overline{A \cdot B}$ (the same rules apply regarding the placement and appearance of the dot as for the AND gate - see the section on basic logic gates). The Boolean expression $\overline{A \cdot B}$ reads as "A NAND B".

The truth table for a two-input NAND gate looks like

A		$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

The NOR gate

The NOR gate has two or more inputs. The output from the NOR gate is 1 if and only if all of the inputs are 0, otherwise the output is 0. This output behaviour is the NOT of A OR B. The NOR gate is drawn as follows



The output from the NOR gate is written as $\overline{A+B}$ which reads "A NOR B".

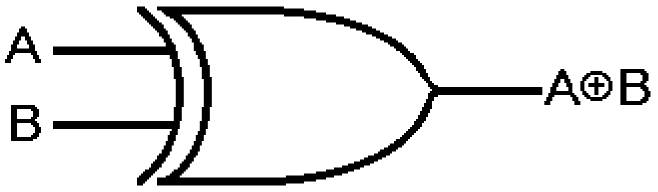
The truth table for a two-input NOR gate looks like

A		$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

The eXclusive-OR (XOR) gate

The exclusive-OR or XOR gate has two or more inputs. For a two-input XOR the output is similar to that from the OR gate except it is 0 when both inputs are 1. This cannot be extended to XOR gates comprising 3 or more inputs however.

In general, an XOR gate gives an output value of 1 when there are an odd number of 1's on the inputs to the gate. The truth table for a 3-input XOR gate below illustrates this point. The XOR gate is drawn as



The output from the XOR gate is written as $A \oplus B$ which reads "A XOR B".

The truth table for a two-input XOR gate looks like

A		$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0