Foundation of Boolean Logic

To begin our understanding of the basic circuits in a computer, we must learn “Boolean” (True/False) logic, how it is reflected in binary data, and how it is implemented in Integrated Circuits.

At its core, Boolean refers to values of “True” and “False”. These are represented by the binary values of “1” and “0” – “1” being “True” and “0” being “False”.

These are then implemented in the circuitry of Integrated Circuits as Voltage and Ground. Typically, these were +5V DC being “1” or “True”, and ground representing “0” or “False”.

These circuits are combined into functions called “Gates”, which have input and output, and sometime a trigger to “set” or “reset” the gate.

These have evolved over time from Valves (tubes) to Relays to Transistors to Integrated Circuits.

There are many hobbyists out there building (or restoring) relay-based computers and some are quite incredible.

The concept of “Boolean” is also used in computer languages, as is the direct implementation of Boolean logic and functions. “And”, “Or”, and “Not” are quite common – as are the built-in values of “TRUE” and “FALSE” in many languages.

This was not always the case – before the languages implemented these natively, we had to define them ourselves. One trick was to use Boolean Logic to define Boolean Values:

FALSE = 0

TRUE = NOT FALSE

For fun, I love to use this snippet in my code:

Define an integer “x” as equal to 0. Then, use this java code (or other language)

x = Math.abs(~x)

The tilde (~) is the Boolean symbol for the “NOT” bit-flipper, and pass that into the Absolute Value procedure from the Math library.

This sets X to the absolute value of NOT X. It increments X. See if you can figure out why it does.

# Logic Gates

## Basic Gates

### Buffer or “YES” gate

This one is the absolute simplest gate. From a logic perspective, it does nothing. Whatever the input value is, that is the output value. It is used to “synchronize” signals. I’ll explain that later when we get to the XOR gate.

|  |  |
| --- | --- |
| Input Value | Output Value |
| TRUE | TRUE |
| FALSE | FALSE |

The symbol for the buffer is a triangle – input from the left, output on the right:

Shape

AI-generated content may be incorrect.

### NOT

The first gate that actually modifies the input is the “NOT”. Its output is the opposite of its single input:

|  |  |
| --- | --- |
| Input Value | Output Value |
| TRUE | FALSE |
| FALSE | TRUE |

The symbol for the NOT gate is the buffer with a dot on the right that signifies inverting the value:

Shape

AI-generated content may be incorrect.

### OR

The next gate uses 2 input values to control the one output value, and is the “OR” gate. If one OR the other OR both inputs is true, then the output is true:

|  |  |  |
| --- | --- | --- |
| “A” Input | “B” Input | Output Value |
| FALSE | FALSE | FALSE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| TRUE | TRUE | TRUE |

The symbol for the OR gate is:

Diagram

AI-generated content may be incorrect.

### NOR

This gives the inverse output value of the OR gate:

|  |  |  |
| --- | --- | --- |
| “A” Input | “B” Input | Output Value |
| FALSE | FALSE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| TRUE | TRUE | FALSE |

The symbol for the NOR gate adds the dot for inversion to the OR:

Diagram

AI-generated content may be incorrect.

### AND

The next gate again uses 2 input values to control the one output value, and this is the “AND” gate. Only if BOTH inputs are true is the output value true:

|  |  |  |
| --- | --- | --- |
| “A” Input | “B” Input | Output Value |
| FALSE | FALSE | FALSE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| TRUE | TRUE | TRUE |

The symbol for the AND gate is:

A picture containing shape

AI-generated content may be incorrect.

### NAND

And this gives the inverse value of the AND gate:

|  |  |  |
| --- | --- | --- |
| “A” Input | “B” Input | Output Value |
| FALSE | FALSE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| TRUE | TRUE | FALSE |

The symbol for the NAND adds the dot for inversion to the AND:

A picture containing shape

AI-generated content may be incorrect.

## Advanced Gates

### XOR

The first advanced gate we encounter is the “XOR” which is an extension of the OR, with one key difference. Its output value is TRUE only when one OR the other BUT NOT BOTH are true:

|  |  |  |
| --- | --- | --- |
| “A” Input | “B” Input | Output Value |
| FALSE | FALSE | FALSE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| TRUE | TRUE | FALSE |

The symbol for the XOR is similar to the OR, but with a line in front of the input:

A picture containing text, athletic game, sport

AI-generated content may be incorrect.

Want to know how this one is built?

It is a combination of the NOT, AND, and OR gates, and the Boolean logic expression is:

“(A and NOT B) OR (B and NOT A)”

To show the signals moving through this in synchronization, we can take the first AND gate – it uses the “A” as-is, but processes the “B” through a NOT gate. So if we add the YES gate on the “A” line, then both signals run thorough a gate, staying in synch. We don’t actually do this, but sometimes it helps to explain a diagram.

An interesting effect is achieved by combining gates – these gates can emulate or implement every other gate – they are made through the combination of the AND and OR gates with the NOT gate, creating the “NOT OR” and “NOT AND”

### XNOR

And the inverter for the XOR is the Exclusive NOR, or XNOR. Add the inverter dot:

Diagram

AI-generated content may be incorrect.

# Performing Math with Logic

The most interesting thing about Boolean Logic and Binary is that you can perform math with these simple gates, and we’re about to learn how easy it is.

Let’s look at a few examples in Binary, with this 2-digit example:

00 + 00 = 00

00 + 01 = 01

01 + 00 = 01

But what does 01 + 01 equal in Binary? ... yeah … it is “10” isn’t it?

What does that represent?

Well, think way back to elementary school and your first math classes.

6 + 6 is NOT “12”. It is “Two and carry the one”.

5 + 5 is “Zero and carry the one”.

Same in binary.

1 + 1 = “0 and carry the one”.

Let’s break it down, and figure out what circuits (Boolean logic gates) will do what we need.

Lets start with the 1’s column and the desired output for our “Sum” and “Carry” values:

|  |  |  |  |
| --- | --- | --- | --- |
| “A” | “B” | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

What gates do this for us? Well, the “Sum” needs to be one if ONLY one of the inputs is one, so that is the “XOR” gate. The carry needs to be one only if BOTH inputs are 1, so that is the “AND” gate.

A simple addition is therefore 2 gates:

Sum = A XOR B, Carry = A AND B.

And this simple circuit is called a “Half Adder”. Why is it only a “half” of an adder? Well, think about the “2” column in a binary value. That one has to add the “A” and “B” value AND the “Carry In”. All the digits past the 1 bit must handle 3 bits of input values, so they are “Full Adders”.

## Half Adder

Here is the logic gate diagram of a Half Adder, using a single XOR and a single AND gate:

Diagram

AI-generated content may be incorrect.

And its simple Boolean Logic Table:

Calendar

AI-generated content may be incorrect.

## Full Adder

Here is the logic gate diagram of a full adder, using 2 XOR, 2 AND, and 1 OR gate. Its “Boolean Logic” is:

Sum = ((A XOR B) XOR Cin), Carry Out = (((A XOR B) AND Cin) OR (A AND B))

Diagram

AI-generated content may be incorrect.

And its Boolean Logic table:

Table

AI-generated content may be incorrect.

## We can Add, but can we Subtract?

Most computers can only add, as they are based on the half and full adders. Do they have “Subtract” instructions? Of course they do.

But the circuitry does NOT.

Again, let’s go back to Elementary School Math and look at something dumb. How do you turn subtract into add?

10 minus 5 is the same thing as 10 plus (negative 5).

So that is how our computer does it. It “inverts” the value (using NOT gates) and adds it. It uses “two’s complement” to perform the inversion, but that’s just to make it easy for the circuit to implement. And means our first bit can’t be a “half” adder as the two’s complement does an invert and set carry in.

Let’s do it in Binary to see how the circuit has to handle it:

8 = “1000”; 2 = “0010”

Invert the 2 = “1101”, and add:

1000

1101

(1)0101

That’s wrong. Let’s add that “carry in” after the invert:

0001

1000

1101

-======

(1) 0110

There it is!

And now you can Add and Subtract – which means you can multiply and divide using loops.

# The Basic Boolean Circuits in Transistors

You can build the circuits from individual transistors or relays as well, or just see how the transistors are used from the data sheets of an integrated circuit, such as these for the “NOT”, “OR” and “AND” gates:

|  |  |  |
| --- | --- | --- |
| Diagram  AI-generated content may be incorrect. | Diagram, schematic  AI-generated content may be incorrect. | AND gate using transistors |

Here we see the “NOT”, where the output is +5V (1/True) until the input is +5V, which turns on the transistor and “grounds out” the output.

The “OR” shows that if either transistor is turned on, then the output will be on.

The “AND” shows that both transistors must be on for the output to be on.

We could even build the “YES” buffer gate by using just one transistor from the OR or AND gate. If the input is on, the output is on.

One thing to note about Electronic Engineers is that they can come up with strange ways of implementing the desired result. As an example, an XOR gate can be made using an OR and an AND gate – “stack” the OR gate with the AND – if either input is a 1, the OR turns on, but if both are 1, the AND turns on and “Grounds out” the result of the OR.