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. ( Or Transistors, or Relays or even Vacuum Tubes! )

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”.

Where does the invention of Boolean Logic fall in the rise of computers?

Well, Charles Babbage conceived his “Difference Engine” in the 1820’s, then his Analytical Engine in the early 1840s. Ada Lovelace invented the first algorithm to run the Analytical Engine in 1843. Finally, in 1847, George Boole invented Boolean Logic and Boolean Algebra. His seminal work on it is available for free from Project Gutenberg at:

<https://www.gutenberg.org/files/36884/36884-pdf.pdf>

I think we can call it a “mature” foundation of computing.

Coming back to today, Boolean logic is implemented in the circuitry of our computers as Voltage and Ground. Typically, these are +5V DC being “1” or “True”, and ground representing “0” or “False”.

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

These have evolved over time from Babbage’s gear train to Valves ( Vacuum Tubes ) to Relays to Transistors to Integrated Circuits to our modern Single Board Computer chips and massive multi-core CPU chips. And the “System on a Chip” that powers your cellphone.

Quite a few YouTube channels are dedicated to these technologies:

Valves

<https://www.youtube.com/@UsagiElectric> (see the UE-1 Computer videos and Bendix G15)

Relay Computing

<https://www.youtube.com/@dipdoting>

Transistor Computing

<https://www.youtube.com/watch?v=_eo8l7HP-9U>

Integrated Circuit SAP-1 by Ben Eater:

<https://www.youtube.com/playlist?list=PLowKtXNTBypGqImE405J2565dvjafglHU>

This is the kit we are writing this course to support.

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 “hard code” the 1st Boolean constant, then use Boolean Logic to define the 2nd Boolean constant:

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. By the end of building the breadboard computer in this course you will be able to figure out why it does.

## Boolean Algebra

### Keyboard Symbols

These are typically used in user documentation whereas the Unicode or ALT symbols are used in documentation of Boolean Logic.

NOT ~ (tilde) or ! (exclamation point)

OR | (pipe)

AND & (ampersand)

### Unicode Symbols

Type the 4-digit hex value then immediately hit ALT+X.

AND (U+2227) ∧

OR (U+2228) ∨

NOT(U+00AC) ¬

XOR (U+2295) ⊕

### ALT Symbols

Hold the ALT key and use the num pad to enter the values.

AND 8743 ∧

OR 8744 (“V” or 124 (PIPE: |) ∨|

NOT 0172 ¬

XOR 8853 (small symbol) ⊕

# Boolean Logic Gates

## Basic Gates – 1 in, 1 out

### 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.

## Combinational Gates – 2 in, 1 out

### 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, and is called the “NOT OR”:

|  |  |  |
| --- | --- | --- |
| “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, as it is called the “NOT AND”:

|  |  |  |
| --- | --- | --- |
| “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 Exclusive Or ( “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 one of these:

“(A AND NOT B) OR (B AND NOT A)”

“(A ∧¬B) ∨ (B ∧ ¬A)”

“(A & !B) | (B & !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 NOT OR, or XNOR. Add the inverter dot:

Diagram

AI-generated content may be incorrect.

## Multi-Input Gates

To make circuits easier to build, there are many multi-input logic circuits that expand these. 4-input chips are frequently used, such as in a circuit determining if a byte value is zero. Here is the 4-input AND:

A picture containing diagram

AI-generated content may be incorrect.

# Performing Math with Logic

The most interesting thing about Boolean Logic and using Binary as our base 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 these 2-digit examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 00 |  |  | 00 |  |  | 01 |
| + | 00 |  | + | 01 |  | + | 00 |
|  | 00 |  |  | 01 |  |  | 01 |

But what does 01 + 01 equal in Binary? ...

|  |  |
| --- | --- |
|  | 01 |
| + | 01 |
|  | 10 |

yeah … it is “10” isn’t it? What does that represent in the “show your work” world?

Well, think way back to elementary school and your first math classes where you had to explain the answer:

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”.

Remember that these were called the “sum” and the “carry”. And this is how all our math circuits work. Let’s break it down, and figure out what circuits (Boolean logic gates) will do what we need.

We will 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 next column in a binary value. That one has to add its “A” and “B” value AND the “Carry In” from the prior column. All the digits past the 1 bit must handle 3 bits of input values, so they are “Full Adders”.

Actually, the 1st bit has to be a full adder, but we’ll figure that out when we try to subtract.

## 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.

### Parallel Full Adder

Of course, we don’t need a 1-bit computer. We need 8 (or 16, 32, 64…) so we need to put 8 of these “Full Adder” circuits in parallel to add 2 bytes together.

However, this was a real issue for early computers. Do you see the problem? Look back at the logic gate diagram of the Full Adder.

Well, the “Sum” goes through 2 XOR gates, but the “Carry” goes through 3 gates. This means that the carry bits are set AFTER the first sum bits have appeared on the output. We have to wait for the carry bits to “ripple” through the total number of bits before we can trust the answer.

Circuits needing to wait on ripples have long been the bane of computer hardware engineers, with one engineer actually implementing a “NotYet” signal in their product to solve this problem.

[ Soul of a New Machine by Tracy Kidder ]

## We can Add, but can we Subtract?

The only “math” circuit we have is the parallel full adder. So obviously we can implement “ADD” instructions, but what about subtract? Yes, our CPU and ALU chips implement “SUB” instructions. (Some even have multiply and divide instructions, but most rely on assembly language code to do it)

However, 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? Way back in 3rd or 4th grade you learned about the “Number Line” and “Negative Numbers”.

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 or valves 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.

So that is our introduction to Boolean Logic and the circuits that implement these basics. In our course, we will be using integrated circuits (ICs) that implement these logic gates as well as a few specialized ICs for the Full Adders and moving data via the data bus.

# The Basic Boolean Circuits in Integrated Circuits

The basic Integrated Circuits (ICs) we will be using are just packages of the transistors, capacitors, and resistors above. Several of these ICs “datasheets” (the documentation of the chip) actually include the internal circuit diagram, as this one for the LM555 Timer shows:

Diagram

AI-generated content may be incorrect.

You can build this from the transistors and resistors – there is even an “Evil Mad Scientist Labs” kit for it:

<https://shop.evilmadscientist.com/productsmenu/tinykitlist/652>

Many datasheets are comprised of the basic Boolean Logic gate symbols from above, here we see the 74LS00 Quad 2-Input NAND Gate that shows the wiring to the each of the 4 NAND gates within its package, and the NAND gate diagram for each:

Diagram, schematic

AI-generated content may be incorrect.

Of note is the usage of terminology – Dual, Quad, Hex, Octal (2-4-6-8) to signify how many of the logic gates are contained within the package of the integrated circuit. Let’s look at those before we go through each of our basic logic gate ICs, and see how the “word” is used to indicate the number of logic gates, and the “digit” is used to indicate the number of input signals:

Here is a Quad 2-input NAND:

Diagram, schematic

AI-generated content may be incorrect.

Here is a Dual 4-Input NAND:

Diagram

AI-generated content may be incorrect.

Here is the 74LS04 Hex Inverter:

Diagram, engineering drawing

AI-generated content may be incorrect.

The 74LS245 Octal bus transceiver:

Diagram

AI-generated content may be incorrect.

## Basic Gates

### Buffer (YES) [74LS07 Hex Buffer]

Diagram, engineering drawing, schematic

AI-generated content may be incorrect.

### NOT [74LS04 Hex Inverter]

Diagram, engineering drawing

AI-generated content may be incorrect.

### OR [74LS32 Quad 2-input]

Diagram

AI-generated content may be incorrect.

### NOR [74LS02 Quad 2-input]

Diagram, schematic

AI-generated content may be incorrect.

### AND [74LS08 Quad 2-input]

Diagram

AI-generated content may be incorrect.

### NAND [74LS00 Quad 2-input]

Diagram, schematic

AI-generated content may be incorrect.

## Advanced Gates

### XOR [74LS86 Quad 2-input]

Diagram

AI-generated content may be incorrect.

### XNOR [74LS266 Quad 2-input]

Diagram, engineering drawing

AI-generated content may be incorrect.

Yeah, the manufacturers still won’t use the same diagramming standard. Or numbering standard.

Some number from 0 to 3 or 0 to 7; others from 1 to 4 or 1 to 8. Most use “A” and “B” for the input, and “Y” for the output. But not all. Chips with 2 output “flip-flops” frequently use Q and !Q (Q and NOT Q).

One thing that seems to be universal is that Vcc (Voltage at Common Collector) and GND (Ground) are the same label for all the ICs. Most (but not all) have them in the same location on the IC. GND being the highest pin on the “1” side, and +5V being the highest numbered pin of the IC.

So that is a quick introduction to the fundamentals of Boolean Logic and how Integrated Circuits implement it for the computers we build and run every day.

As we go through the 8-bit, 16-byte microcode based computer kit, we will be learning many more ICs and what they do inside our CPU cores.

I hope this has proven to be useful, and I hope you enjoyed learning some of the forgotten basics.