Boolean Algebra & Logic I.C.’s

Boolean Expressions and Integrated Circuit Logic Gates. Illustrated.

by Frederick Hoehn, copyright 2014, all rights reserved.

Chapter 1

Students of Algebra can take four semesters to learn Algebra. But it doesn’t even take one semester to learn Boolean Algebra. There are a few rules for Boolean Algebra, and there are DeMorgan’s laws. I’ll explain it to you.

Boolean Algebra gives us a mathematical way to deal with logic functions such as those that are employed in the everyday operation of personal computers, IPAD’s, smart phones, Programmable Logic Controllers, and microprocessors.

Those devices are simply loaded with logic gates: and gates, or gates, inverters, exclusive or gates, nand gates, and nor gates.

It takes hundreds of such gates to make a microprocessor. Microprocessors are the heart of a personal computer. But microprocessors also need memory chips, and other circuitry to make a whole computer.

You have your keyboard, your hard drive, your optical drive for CD’s and DVD’s, your display or monitor, perhaps a microphone input to the computer, and headphone output. Your USB ports for use with flash drives and other devices.

All of that is tied together with chips that have and gates and or gates. Those “tying together” chips used to be a significant portion of the real estate on the mother board until someone figured out the necessary logic functions and combined them in “chipsets.”

Chipsets were a significant step forward by freeing up space on the motherboard. More important for real estate in laptops than in desktop computers, but still important.

Each logic function required can be expressed as a Boolean expression.

Boolean expressions can then be combined in producing the chipsets.

In Algebra, we use letters of the alphabet to represent “variables.” A variable is a number that might take on different values. When solving for “X” in Algebra, we don’t know what X is until we’ve finished the algebraic manipulations.

We use variables, too, in computer programs. A variable may take on different values. We can start a variable at some initial number, but as the program executes, that variable could count up to ten thousand, or be multiplied or divided, according to what’s been written in the program.

Quantities are also represented with letters in Boolean Algebra. But in Boolean Algebra, we’re dealing with digital logic.

The gates in figure 1 show inputs to the left of the gate, and the output at the right of the gate. Each input and each output could be represented by a bit in memory. A bit is a binary digit. It can be either a 1 or a 0.

Illustration G in figure 1 shows some gates connected together to produce an “exclusive or” output.

The two inputs are labeled H & J, and the output is labeled Z.

H and J are both inputs to the “or” gate and to the “and” gate below it.

That and gate below the or gate will yield a 1 at its output if both H and J are 1 at its inputs. The output of that and gate goes into an inverter. An inverter will change a 1 to a 0, or a 0 to a 1.

With both H and J high (a 1) the output of the inverter will be 0. This 0 will cause that the output of the second and gate of illustration G to be 0.

If either input to an and gate is 0, the output will be 0.

If either H or J is 1, but not both 1 at the same time, the 1 output from the or gate will pass through the second and gate as a 1 output.

If both H and J are 0, the output at Z is 0.

Thus the Z output from those three gates and one inverter will have the same “truth table” as an exclusive or gate.

The way that the “exclusive or” gate is different from an “or” gate is that if both inputs are 1 at the same time, then the output is 0.

Here is the Boolean expression for those gates at G in Figure 1:

(H + J) \* (H \* J)’

+ means “or”. \* means “and”. ‘ means inversion.

But since we know that this is the exclusive or function, an easier way to write it is just:

HJ’ + H’J

Of course, you can buy chips with the exclusive or gates. We’re just showing how it could be accomplished with and gates, an or gate, and an inverter.

Intel is a manufacturer of Integrated Circuit chips. They started out as a manufacturer of memory chips, but now, they’re more widely known as a manufacturer of microprocessors such as the Pentium.

AMD is also a manufacturer of microprocessors.

Microprocessors contain thousands of and & or gates, inverters, nand gates, nor gates.

Your logic gates, properly connected together will add numbers, or subtract numbers. Such circuits are sometimes synchronous (using a timing pulse from timing generators), or they can be asynchronous (no timing pulse).

You can connect them together as RS (reset, set) latches for remembering one bit of data.

On my wrist, I have a digital wrist watch. It has a liquid crystal display (LCD). Inside, it has an Integrated Circuit chip and a battery.

That digital circuit chip has dozens of and gates and or gates to do the functions of a digital watch.

Beside digital integrated circuits, there are also analog integrated circuits.

Analog deals with a range of voltages, as you would be doing with music amplifiers. Of course CD’s store music digitally, but when we want to listen, it’s converted back to analog so that our ears can hear the music.

Digital circuits deal with only two voltages: high and low. Low can be zero volts, or close to it. The high voltage depends on which family of logic devices you’re using. Sometimes, the high will be about +5 volts d.c.

A high represents a 1. A low represents a 0.

There have been analog computers made, but they proved to be not as reliable as digital computers.

You can buy I.C.’s that have four “and” gates to a chip, or four “or” gates to a chip.

I had a job in Silicon Valley with a contractor on the Hubble Space Telescope project. They were using the bit-slice technology for the Hubble’s computer.

One source for the bit-slice chips was National Semiconductor. You can put together as many four bit slices as you need to make a computer wide enough for your data, such as 128 bits.

Even at that time, bit-slice technology was probably ten years old. But the decision was to go with something older, and proven.

I was working at that place with a circuit board where they had decided to replace some logic chips with a PAL.

A PAL is programmable array logic.

I asked one of the lead Engineers what he wanted in the PAL. He showed me another drawing with some logic chips and circled the logic chips and circuitry that would be replaced by the PAL.

On a printed circuit board, the connections between I.C.’s and other components on the p.c. board are made by strips of copper on the board that are arranged to do the job by a printing process called “silk screening.”

I wrote the Boolean expression for that logic, and took it to the computer that had the software for programming the PAL. The way that you “program” a PAL is to blow the correct fuses in the PAL.

A PAL comes with too much circuitry inside. You get rid of what you don’t need by blowing fuses.

I entered the Boolean expression into the computer. Then, before blowing any fuses, you can look at another screen to see which fuses will blow.

I saw that the wrong fuses were going to blow that would not produce the logic that we needed. That was because there was a bug in the software. I found that by rearranging the Boolean terms, I could get the right fuses to blow.

This should not have happened, because a Boolean expression should always evaluate the same no matter what the order of terms is, just as the order of inputs to a four input “or” gate shouldn’t matter.

Chapter 2

Terms in a Boolean expression are those parts of the expression separated by + signs. If you were to read this last sentence out loud, you might say “plus signs.”

And that sign is the same sign as a plus sign, but in a Boolean expression, that sign means “or”.

We’re representing inputs here with a single capital letter. If you see two such letters next to each other in a Boolean expression, such as HJ, it means that the two are anded together.

In Boolean Algebra, a multiplication sign or a dot between two variables means they are anded together.

There is also, in Boolean Algebra, the vinculum bar. It’s just a horizontal bar placed over a variable and is read “not”, such as “not A”. Not A means the inversion of A. It means a 0 when A is 1, and a 1 when A is zero.

But in this text, we’ll use an apostrophe “ ‘ “ to mean not, since the keyboard doesn’t have the vinculum bar key. An asterisk ( \* ) means “and”.

A “Truth Table” for a logic device shows the various combinations of inputs that it can have, and what the corresponding output will be.

Here is a truth table for an “or” gate:

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Here is a truth table for an “and” gate:

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Here is a truth table for an “exclusive or” gate:

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

High is 1, low is 0.

Referring to the drawing, Figure 1, gate A is a two input “or” gate. Either input high (or both) yields a high output. But if both inputs are low, the output is low.

Gate B is an “and” gate. Both inputs must be high for the output to be high, otherwise, the output is low.

Gate C is an “exclusive or” gate. If either input is high and the other input low, the output is high. If both inputs are low, the output is low. If both inputs are high, the output is low.

At letter D is an inverter. It changes a high to a low, or a low to a high.

Gate E is a nor gate. If either input is high (or both) the output will be low. Otherwise, the output is high. A nor gate is like an “or” gate followed by an inverter.

Here is a truth table for the nor gate:

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Gate F is a nand gate. Both inputs must be high for the output to go low. Otherwise, the output is high. A nand gate is like an “and” gate followed by an inverter.

Here is a truth table for the nand gate:

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Notice that both the nor and the nand gate have a small circle at the output. That’s called a “state indicator.” It means an active low at those outputs.

You can also have state indicators at gate inputs, but if you look at logic chip catalogs from logic chip manufacturers, they don’t use state indicators on logic chip inputs.

If you’re going to do a lot of logic design, it would be good to have one of those catalogs, so you know what’s available.

Those logic chip catalogs also give the pin numbers on the chips for the inputs and outputs of the gates.

Some of this might now be on the Internet, but when I was doing logic design, I worked out of the chip manufacturer’s catalogs.

Texas Instruments had quite a selection of useful chips.

Zilog manufactured microprocessors, and also Mostek.

Chip manufacturers, beside gates also have D flip flops, JK flip flops (a flip flop is a bistable multivibrator), registers, 8 to 3 encoders, 3 to 8 decoders, transparent latches, static memory chips, dynamic memory chips.

Also tri-state devices like buss drivers that have high and low outputs or high impedance output so that you can switch between buss drivers with you tri-state enable input.

There’s also a book called “The TTL Cookbook.”

You can also purchase logic gates with an “open collector.” They are used with an external collector resistor from the output pulled up to your more positive voltage source. Then, several gates can use the same collector resistor for a “wired and” function, since any of the gates with the same collector resistor can pull the output at the common output point low.

Bipolar, junction transistors have an emitter, base, and collector. If you buy an individual (“discrete”) transistor, it has three wires coming out of it, one for emitter, one for base, and one for collector.

The current in the emitter-base circuit controls the current in the emitter-collector circuit.

Discrete transistors would usually have their collectors connected externally to collector resistors.

But if your transistors are inside logic integrated circuits, the collector resistors are usually inside the inside, unless you buy the kind that is “open collector,” in which case, you use an external collector resistor.

But field effect transistors (f.e.t.’s) instead of emitter, base, and collector, have source, gate, and drain.

They are voltage operated devices and are good for input circuits of voltmeters because of their high input impedance.

If you want to buy a voltmeter, it’s desirable to get one with a high input impedance. In electronics, the letter used for impedance is capital Z.

A voltmeter with a high input impedance will become less a part of the circuit being measured than one with a low impedance.

When studying bipolar, junction transistors, I believe it’s more helpful to study first the NPN transistors, and then PNP transistors later.

NPN transistors are taught with electron flow. But with PNP’s they sometimes start talking about “hole flow.”

If you take pure silicon, and “dope” it with an element that has excess electrons, it becomes “N” type material. If you dope with an element that is a little short of electrons, it becomes “P” material.

An NPN transistor is two layers of N material sandwiching a layer of P material in between.

In schematic diagrams, the circuit symbol for a diode, a two element device is an arrow head bumping broadside into a short line.

A diode is a device that passes current in one direction, and blocks it in the other direction.

Looking at a schematic diagram, one might think that the direction of current flow through the diode would be in the direction of the arrow. But no, current is in the direction against the arrow.

When the diode symbol was invented, it was thought that current flowed from the positive terminal of the battery, through the external circuit, and back into the negative terminal of the battery.

Then someone figured out that current flow is the flow of electrons which are attracted to the positive terminal of the battery, so current flow is in the opposite direction than was first thought.

So, we’re stuck with a circuit symbol for a diode where current flow is in the opposite direction of the arrow head in the schematic diagram.

With PNP transistors, they’re sometimes taught as having holes flowing through them in one direction. Actually, it is not holes that flow. It is electrons that flow in the opposite direction.

Mathematical formulas have been devised for hole flow, and they work, but a hole is the absence of an electron. An electron moves out of one position, leaving a hole, and goes to a new position where it fills a hole.

It was not the hole that moved, it was the electron that moved.

Do you remember from Algebra class that if two variables such as X and Y are placed next to each other (like XY), that it means that they are understood to be multiplied together?

Similarly, XY next to each other in Boolean Algebra would mean that they are “anded” together in an “and” function. Or, you could have ABC anded together, or four or five variables.

A variable is a variable because it can take on different values. But in Boolean, it can only evaluate to 1 or 0. To true, or false.

Those groups of anded variables are separated by “plus” signs in a Boolean expression. But the “plus” sign means the “or” function.

Here is an example of a Boolean expression:

AB’ + A’BC + (ABC’)’

Remember that the apostrophe here means inversion. The apostrophe after the term that’s in parentheses means that the result of the anding inside the parentheses is inverted.

At the beginning of the parentheses one would read “the quantity”, as in “the quantity A, B, and not C”.

Here are some rules for Boolean Algebra.

A + 0 = A

A + 1 = 1

(If you or anything with 1, you get 1.)

A \* 0 = 0 (anded)

(If you and anything with 0, you get 0.)

A \* 1 = A

A + A’ = 1

A \* A’ = 0

(If you and anything with its inversion, one of them will be 0, and the result is 0.)

DeMorgan was a fellow who figured out some things about logic and Boolean Algebra. Here are his laws:

DeMorgan’s laws:

(A + B)’ = A’ \* B’

(A \* B)’ = A’ + B’

You may be able to use these DeMorgan transformations to rearrange your Boolean Expression to something more convenient.

Consider illustration K in Figure 1. It shows a four input nand gate. If we DeMorganize that we get the four input or gate immediately below with state indicators on all four inputs. The truth tables for both gates will be the same.

Now see illustration L in Figure 1. It shows a four input nor gate. If we DeMorganize it we get the and gate immediately below it with state indicators on all four inputs. The truth tables for both gates will be identical.

But I haven’t seen logic chip manufacturers use state inputs on gate inputs. But you may have occasion to prefer to consider a nor gate to be an and gate for lows.