Episode 4.05 – Introduction to Boolean Algebra

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java. And one more thing. Our topics involve a bit of figuring, so it might help to keep a pencil and paper handy.

Algebra – it's a word that strikes fear into the hearts of secondary school students around the globe. This discipline of mathematics along with its variables, equations, and proofs deals with numbers, an infinite number of them. What we're going to talk about in this episode is Boolean algebra, and in Boolean algebra, there are only two numbers, one and zero. This finite number of numbers simplifies many of the properties and identities of traditional algebra. And to prove those identities, well, all we need is a truth table. By limiting the values a variable can take on to zero or one, a proof becomes a matter of showing how the axiom is true for every possible combination of ones and zeros for which the input variables can equal.

To get to the point where we are evaluating expressions in Boolean algebra, we need to have a way to record those expressions. Different disciplines have different ways of representing Boolean expressions. The meaning is the same, but the symbols can be different. Let's begin with the operations themselves. As we discussed in Episode 4.01 – Intro to Logic Gates, the AND operation behaves like the minimum function while the OR operation behaves like the maximum function. We see this in how mathematicians write Boolean expressions. The AND function can be represented with the caret or wedge (the symbol found above the 6 key on standard US QWERTY keyboards). In mathematics, this is the min function.[1] The OR function can be represented with the small 'v' or cup symbol, which in mathematics is the max function.[1] This is supported by the fact that the typesetting system, TEX (pronounced "teck"), which is known for its proficiency at typesetting complex mathematical expressions, represents the caret or wedge with the string \land for logical AND and the cup or small 'v' shape with the command \lor for logical OR. Similarly, in the list of HTML character entities, the wedge or caret is ∧ or ∧ while the cup or small 'v' is ∨ or ∨.[2]

In engineering, however, the AND operation is often referred to as the product and represented with the dot while the OR operation is often referred to as the sum and represented with the plus sign. This makes sense for two reasons. First, when we begin discussing the properties and identities of Boolean algebra, we will see that many of the traditional algebraic expressions involving addition and multiplication behave much like the Boolean algebraic operations of OR and AND respectively. Second, after reducing each of our variables to binary zero or one, we will see that the addition and multiplication operations behave much like the OR and AND operations. For example:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Just like the AND operation, the only time the multiplication of ones and zeros equals one is when all the input values are one.

Similarly, the two input OR behaves a bit like addition when only zeros and ones are allowed.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1

1 + 1 is…well…it doesn't equal zero, so it must equal the only non-zero value which is left, one, right? This leaves the third operation: the inverse. Boolean algebra uses an overbar drawn over the signal that needs to be inverted. For example, if we wanted to take the inverse of A, we would place a bar over the single variable A. We typically read this as "A bar". If a bar crossed multiple variables and operations, it would be treated as a set of parentheses in addition to performing the inverse operation. In other words, the result of the entire expression under the bar must be calculated before taking the inverse. When it comes to representing combinational logic expressions, more than likely there will be a mixture of operations. That means we will have to apply another concept from traditional algebra to Boolean algebra: order of operations. Many students learn the acronym "PEMDAS", which gives the order in which algebraic operations should be applied: parentheses, then exponents, then multiplication, then division, then addition, and lastly subtraction. Our British audience may be more familiar with the acronym "BODMAS", which stands for brackets or braces, then orders, then division, then multiplication, then addition, and lastly subtraction. It's all the same thing.

Boolean algebra also has an order of operations, but since we don't use exponents, division, or subtraction, it's reduced to simply parenthesis, and, and or. Remember that the inverse is treated like parenthesis in that everything below the inverse must be calculated before the inverse is applied. So maybe the order of operations should look like, "Inverse, Parenthesis, AND, and OR." IPAO? Alternatively, since parenthesis and inverses are equivalent, and since AND is treated like a product and OR is treated like a sum, we could have, "Parenthesis, Inverse, Product, and Sum." That would give us PIPS. Okay, maybe not.

Now a brief word about the variables themselves. As we discussed in Episode 4.01 – Intro to Logic Gates, our input variables are single letters starting with A. Unlike traditional algebra, we capitalize these letters.

It's time for an example. Assume that we want to perform the following operation on the two input signals: A and B. First, OR A and B together. Then, take the inverse of the result of that OR. Lastly, take the logical AND of B with the inverse of the OR of A and B. Written in Boolean algebra, this would look like:

$B \cdot (A + B)$ (Read, "B and-ed with the inverse of A or B.")

Now let's see what the truth table for this expression looks like for each of the four possible binary values for A and B. Because they are inside the parenthesis, under the bar, we begin by OR-ing A and B. When both A and B are zero, we get 0 + 0 = 0. Next, since the bar is over the OR operation, we invert the 0 to get 1. Lastly, we AND B, which equals zero, with the one from our inverse to get $0 \cdot 1 = 0$. That means that when A and B are both zero, the result of $B \cdot (A + B)$ equals zero.

Let's repeat this for A equals zero and B equals one. We start by OR-ing A and B together, once again, because they are inside the parenthesis, and under the inverse. This gives us 1 + 0 = 1. Next, since the bar is over the OR operation, we invert the 1 result to get 0. Lastly, AND this zero with B, which equals one, to get 1 · 0 = 0. That means that when A is zero and B is one, B · (A + B) equals zero.

Now for A equals one and B equals zero. We start by OR-ing A and B together to get 1 + 0 = 1. Next, we invert the result of the OR operation to get 0. Lastly, we AND this zero with B, which equals zero, to get 0 · 0 = 0. This means that when A is one and B is zero, B · (A + B) equals zero.

Lastly, let's determine what our expression equals for A equals one and B equals one. We start by OR-ing A and B together to get 1 + 1 = 1. Next, we invert the result of the OR operation to get 0. Lastly, we AND this zero with B, which equals one, to get 1 · 0 = 0. That means that when both A and B are one, B · (A + B) equals zero.

Well this is interesting. It appears that for every possible value that A and B can take on, the result of B · (A + B) is zero. Believe it or not, we've written a proof. We've proven that the expression B · (A + B) is equivalent to zero.

In our next episode, we will present some basic properties of Boolean algebra. If you are familiar with traditional algebra, these will look surprisingly familiar – except for one case. And that case, which is a direct result of limiting ourselves to just ones and zeros, is going to look a little odd. For transcripts, links, or other podcast notes, please visit us at intermation.com where you will also find links to our Instagram, Twitter, Facebook, and Pinterest pages. Until the next episode, remember that while the scope of what makes a computer is immense, it's all just ones and zeros.

References:

1 – https://en.wikipedia.org/wiki/List_of_mathematical_symbols
2 – https://dev.w3.org/html5/html-author/charref