Episode 4.07 – Identities of 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.

Through extensive use, many of the laws of traditional algebra have become second nature to us. For example, the negative of a negative is positive. Zero times any value equals zero. One times any value returns that value. Zero added to or subtracted from any value results in that value. Anything added to its negative equals zero.

Boolean algebra has analogous laws, and in this episode, we're going to look at some of them. In Episode 4.05 – Introduction to Boolean Algebra, we discussed how the logical OR and AND have similar behavior to addition and multiplication respectively. In this episode, we will reinforce that assertion by showing how the identity and inverse laws of algebra have parallel laws Boolean algebra. We will also see how limiting our values to one and zero create some additional laws such as those of annulment and idempotent. What we will see is that any Boolean signal combined with a constant, itself, or its inverse will result in a term dropping out. It may be the constant or it may be the signal itself, but something's going to simplify.

Let's start our discussion with the logical OR operation. In traditional algebra, the identity law of addition states that when you add zero to a value, the zero drops out. Is there a constant we can OR with a signal that results in the original signal? Like addition, it turns out that this constant is zero. In other words, A + 0 = A. Let's visualize this law using the switch analogy. The OR operation acts like switches connected in parallel. A switch controlled by a logic zero will never be closed. If we wire this switch in parallel with a switch controlled by a signal A, then only the switch controlled by A will pass a logic one along to the output, in other words, the output will follow the signal A.

Now for the inverse law. What happens when you combine a signal with its inverse across the OR operation, in other words, what does A + A equal? Let's think about the switches in parallel again. Picture two switches in parallel, one that's controlled by A while the other is controlled by the inverse of A. When the first switch is closed, a logic one can pass through it to the other side of the circuit and the output is one. If the first switch is open, then being controlled by the inverse of A, the second switch must be closed, and once again, a logic one passes through to the output. That means regardless of the value of A, A + A = 1, because one of the switches will always be closed.

In addition to laws of identity and inverse, there is an annulment law for the OR operation. In Boolean algebra, an annulment law means that there is a constant that when combined with a signal cancels out the signal. In the case of the OR operation, that constant is 1, in other words, A + 1 = 1. Picture two switches in parallel, one that's controlled by A while the other is controlled by a logic one. The switch that is controlled by the logic one remains closed regardless of the value of A, and therefore, always passes a logic one across the circuit. That means regardless of the value of A, A + 1 = 1.

The fourth and last class of laws examined here is the idempotent law. This type of law states that when a Boolean signal is combined with itself, the output of the operation equals the original signal. In other words, A + A = A. Think about the switches again. Two switches, each controlled by the same signal, A, and wired in parallel will either both be open for A = 0 or both be closed for A = 1. When both are open, the circuit's output is zero. When both are closed, the circuit passes a one to the output. Therefore, A + A = A.

In summary, the laws for the OR operation are

A + 0 = A (the identity law)
A + 1 = 1 (the annulment law)
A + A = A (the idempotent law)
A + A = 1 (the inverse law)

As with any Boolean proposition, each of these laws can be proven using truth tables. The truth tables for each of these laws are quite simple having only two rows: one for A = 0 and one for A = 1. For example, picture the truth table for the identity law. When A = 0, 0 + 0 = 0. When A = 1, 1 + 0 = 1. This shows that A + 0 follows A.

The AND operation also has an identity law, an annulment law, an idempotent law, and an inverse law. Starting with the identity law, we ask ourselves, "What constant when AND-ed with a signal A results in an output equivalent to A?" Like multiplication, it turns out that this constant is one. In other words, the product of A and 1 equals A.

The switch analogy for the AND operation has switches connected in series. For a logic one to get from one end of the circuit to the other, all the switches must be closed. A switch controlled by a logic one is always closed. When this switch is connected in series with a switch controlled by a signal A, then it always passes along the value passed through by the signal A.

Now for the inverse law. What happens when you AND a signal A with its inverse, A? Let's use the switch analogy again. Picture two switches in series, one that's controlled by A while the other is controlled by the inverse of A. When the first switch is closed, the second must be open. When the switch controlled by A is closed, the first switch is open. Therefore, regardless of the value of the signal A, one of the switches is open and the logic one cannot get from one side of the circuit to the other. That means regardless of the value of A, A · A = 0.

Now, let's look at the annulment law. What Boolean value when AND-ed with a signal A cancels out that signal? It turns out that for the AND operation, that constant is zero. Picture two switches in series, one that's controlled by A while the other is controlled by a logic zero. The switch that is controlled by the logic zero remains open regardless of the value of A, and therefore, prevents a logic one from crossing the circuit. That means regardless of the value of A, A AND-ed with zero always equals zero.
Lastly, let's examine the idempotent law. What happens when we AND A with itself? Think about the switches one last time. Two switches, each controlled by the same signal, A, and wired in series will either both be open for A = 0 or both be closed for A = 1. When both are open, the circuit's output is zero. When both are closed, the circuit passes a one to the output. Therefore, A · A = A.

In summary, the laws for the AND operation are

$A \cdot 1 = A$ (the identity law)
$A \cdot 0 = 0$ (the annulment law)
$A \cdot A = A$ (the idempotent law)
$A \cdot A = 0$ (the inverse law)

Although the exclusive-OR operation doesn't have a parallel in traditional algebra, its numerous applications in Boolean and digital operations suggest that we could benefit from examining what happens when we exclusive-OR a Boolean signal with a zero, with a one, with itself, and with its inverse. Before we do this, however, remember that the two-input XOR gate outputs a one if its inputs are different and a zero if its inputs are the same. We can use this attribute to see what happens to our signal A for each of these four situations.

What happens if you exclusive-OR the signal A with a zero? When A equals zero, the signals are the same and the output equals zero. When A equals one, the signals are different and the output equals one. Therefore, $A \oplus 0 = A$, an identity law so to speak.

Next, what happens if you exclusive-OR the signal A with a one? When A equals zero, the signals are different and the output equals one. When A equals zero, the signals are the same and the output equals zero. Therefore, $A \oplus 1 = A$. That's different.

Does the idempotent law work the same for the exclusive-OR operation as it does for OR and AND? It turns out that it doesn't. When you exclusive-OR anything with itself, both inputs are the same, and hence, the output equals zero. Therefore, $A \oplus A = 0$.

Lastly, let's look at the inverse law for the exclusive-OR. When you exclusive-OR anything with its inverse, the inputs to the exclusive-OR are always different, and hence, the output equals one. Therefore, $A \oplus A = 1$.
This gives us the four laws for the exclusive-OR operation:

$A \oplus 0 = A$
$A \oplus 1 = A$
$A \oplus A = 0$
$A \oplus A = 1$

Before we leave, let's look at one more law. What happens when you take a signal A and run it through an inverter? If A equals one, the inverter's output is zero and if A equals zero, the inverter's output is one, right? What if we take the output of that inverter and run that into a second inverter? Well, the Boolean value of the signal will revert to the original value, A. Although the NOT operation or inverse does not have an equivalent in mathematical algebra, its behavior is like that of algebraic negation when it comes to taking the inverse of the inverse. In algebra, the negative of a negative is a positive. In Boolean algebra, taking the inverse of an inverse restores the original value.

This property is referred to as the Double Negation Law and is represented with a double overbar over the signal. Note that both bars of the double overbar must cover identical sections of the Boolean expression. If the lower bar covers only a portion of the expression covered by the upper bar, the

Double Negation Law cannot be applied. That doesn’t mean that nothing can be done to simplify the expression, however. We’ll just have to wait until we examine DeMorgan’s Theorem to see how.

In our next episode, we will look at DeMorgan’s Theorem. In traditional algebra, we are used to distributing a negative sign across a sum. We will find no similarity here when we distribute an inverse across an OR, or for that matter, an AND either. DeMorgan’s theorem will lay that out for us. 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.