Episode 4.10 – More Boolean Simplifications

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. Due to the computational nature of this episode, you might want to visit the transcript page found at intermation.com to download the episode worksheet.

Because many students have trouble when trying to simplify Boolean expressions, we're going to dedicate another episode to examples of simplification. We're also going to show how sometimes, there's more than one way to crack an egg. The examples we present here simplify considerably. This isn't always the case, but we feel these examples will present two things. First, they will show the potential gains through simplification, and second, they will show that more often than not, there's more than one way to simplify an expression. So let's get started.

Our first Boolean simplification will be to show that $A \cdot B \cdot C + B \cdot C$ (read the inverse of the product of A and B and C OR-ed with the product of B and C) simplifies to one.

Remember to look for the red flags to indicate that something is going to drop out of our expression. These red flags included a term combined with itself, a term combined with its inverse, and a term combined with a constant. If we don't find one of these red flags, then we modify the expression using DeMorgan's theorem, the distributive law, the associative law, or the commutative law.

Our expression doesn't have any red flags, so let's modify it with one of the laws. Applying the associative law to the product under the bar, A AND B AND C, we can separate A from B AND C. This gives us $A \cdot (B \cdot C) + B \cdot C$ (read the inverse of the product of A AND-ed with the product B AND C, then OR-ed with the product of B and C).

Next, apply DeMorgan's Theorem to distribute the bar to A and to B AND C while changing the AND operation between A and B AND C to an OR. This results in the expression $A + B \cdot C + B \cdot C$ (read A-bar OR-ed with the inverse of the product B AND C, then OR-ed with B AND C).

What we have now is three terms or products, A-bar, B AND C bar, and B AND C, being OR-ed together. The associative law allows us to group B AND C bar and B AND C together. Since B AND C bar and B AND C are inverses of each other, applying the inverse law for the OR operation replaces them with one. This gives us $A + 1$ (read A-bar OR-ed with one).

Lastly, the annulment law for OR states that anything OR-ed with one is one. That means that A-bar OR-ed with one equals one, and our proof is complete.

One of the things that students of traditional algebra may find frustrating is that the road to a simplified expression seems narrow. Typically, this isn't the case with Boolean algebra. There are usually multiple ways to simplify a Boolean expression. To show this, we're going to do this proof again using a different sequence of operations.

Again, we start with A·B·C + B·C (read the inverse of the product of A and B and C OR-ed with the product of B and C). This time, instead of applying DeMorgan's theorem to only one of the AND operations under the bar, we're going to apply it across the whole product A AND B AND C to distribute the bar to each input and change all of the AND operations to OR. This gives us A + B + C + B·C (read A-bar OR B-bar OR C-bar OR B AND C).

There is an identity of Boolean algebra that states that when you OR a signal with a product containing the inverse of that signal, the inverse contained in the product drops out. In other words, A + A·B (read A OR A-bar B equals A OR B). Instead of using this identity, we will use the alternative form of the distributive law, where you can distribute a signal OR-ed into a product, to prove the same thing. By OR-ing C-bar across B AND C, our expression becomes A + B + (C + B)·(C + C) (read A-bar OR B-bar OR the sum of C-bar OR B AND-ed with the sum of C-bar OR C).

Notice that the sum C-bar OR C is a term being OR-ed with its inverse. The inverse law states that this is equal to one, so we replace C-bar OR C with one. This makes the last term of the expression the sum of C-bar OR B AND-ed with one. The identity law for the AND operation states that anything AND-ed with one is itself, so this term becomes C-bar OR B. The full expression is now A + B + C + B (read A-bar OR B-bar OR C-bar OR B).

Using the commutative law, we can swap the C-bar and B terms to get A + B + B + C (read A-bar OR B-bar OR B OR C-bar). This puts B-bar and B next to each other in the sum. Using the inverse law for OR again, we replace B-bar OR B with one and get A + 1 + C (read A-bar OR 1 OR C-bar).
Lastly, the annulment law for the OR operation states that anything OR-ed with one is one. That means A-bar OR 1 OR C-bar reduces to one…again.

The only way to get better at the simplification of Boolean expressions is to practice, so let's do one more. This time, let's show that (A·B + A·B·C·D) · (A + B) (read the sum of A AND B OR A AND B AND C AND D AND-ed with the sum of A-bar OR B-bar) equals zero.

For our first proof, let's just apply FOIL. After all, this looks just like a factored quadratic expression. AND-ing the first terms gives us A AND B AND A-bar; AND-ing the outside terms gives us A AND B AND B-bar; AND-ing the inside terms gives us A AND B AND C AND D AND A-bar; and AND-ing the last terms gives us A AND B AND C AND D AND B-bar.

Notice that all four of these products contain an input signal being AND-ed with its inverse. We simply need to rearrange the elements of each product using the commutative law for the AND operation so that they are next to each other. That gives us A·A·B + A·B·B + A·A·B·C·D + A·B·B·C·D (read A AND A-bar AND B OR-ed with A AND B AND B-bar OR-ed with A AND A-bar AND B AND C AND D OR-ed with A AND B AND B-bar AND C AND D).

The inverse law for the AND operation states that anything AND-ed with its inverse equals zero. Therefore, inside of each product where a signal is being AND-ed with its inverse, we replace the pair with a zero. In the first and third products, we have A AND-ed with A-bar. In the second and fourth products, we have B AND-ed with B-bar. Replacing these pairs with zero gives us 0·B + A·0 + 0·B·C·D + A·0·C·D (read zero AND B OR A AND zero OR zero AND B AND C AND D OR A AND zero AND C AND D).

Now each product in our expression contains a zero. The annulment law for the AND operation states that anything AND-ed with zero is zero. That turns each of our products into zero giving us 0 OR 0 OR 0 OR 0, which equals zero.

I know it's been a long trudge through these simplifications, but let's prove the same expression again using a different sequence of operations to show that there's more than one way to do this. We start again with (A·B + A·B·C·D) · (A + B) (read the sum of A AND B OR A AND B AND C AND D AND-ed with the sum of A-bar OR B-bar).

The sum of A-bar and B-bar looks suspiciously like the result of applying DeMorgan's theorem to the inverse of the product A AND B. Let's use this backwards application of DeMorgan's theorem, where we pull a bar off of A and a bar off of B to get the inverse of the product A AND B, to give us the final expression of (A·B + A·B·C·D) · (A·B) (read the sum of A AND B OR A AND B AND C AND D AND-ed with the inverse of A AND B).

Now let's distribute the inverse of the product of A and B across the sum of A AND B OR A AND B AND C AND D. This gives us (A·B) · (A·B) + (A·B·C·D) · (A·B) (read A AND B AND-ed with the inverse of A AND B OR A AND B AND C AND D AND-ed with the inverse of A AND B).

By performing this distribution, we see the first product, A AND B AND-ed with the inverse of A AND B, is just a case of the inverse law across an AND, so we replace it with a zero. This zero drops out due to the identity law for the OR operation. Now we are left with (A·B·C·D) · (A·B) (read A AND B AND C AND D AND-ed with the inverse of A AND B).

Using the associative law, we can pair up A AND B. The commutative law allows us to move the inverse of the product of A AND B so that it's next to A AND B. Once again, this is an example of the inverse law, and A AND B AND-ed with the inverse of A AND B should be replaced with zero. We now have zero AND-ed with C AND-ed with D. The annulment law for the AND operation states that anything AND-ed with zero equals zero. That means that the term C AND D drops out leaving zero.

Well, that just about does it for the simplification of Boolean expressions. If the audio of this episode was a little overwhelming, visit the episode page at intermation.com and download the worksheet where all four of these proofs are presented. In our next episode, we will introduce two standard formats for Boolean expressions that are designed for optimum performance. They also provide us with our first chance to derive a Boolean expression from a truth table.

For episode transcripts, worksheets, 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.