Episode 5.03 – The Product-of-Sums Expression

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.

Although the sum-of-products expression described in Episode 5.01 is by far the most common, we are going to examine another standard format of Boolean expressions: the product-of-sums. The fundamental difference is that the sum-of-products expression identifies and merges the conditions when the output should be active, in other words, a logic one, whereas the product-of-sums identifies the conditions where the output should be inactive, in other words, a logic zero. In addition, where an sum-of-products expression can be implemented entirely with NAND gates, a product-of-sums expression can be implemented entirely with NOR gates.

A products-of-sums, or POS expression, identifies each set of conditions that when true makes the expression false, and merges them using the AND operation which passes any false input directly to the output. An examination of the hardware may make this statement a bit clearer. In circuit form, a POS expression takes the output of one or more OR gates and AND's them together with a single AND gate to generate the output. The OR gates are set up to generate a logic zero output for one specific condition of inputs, that condition being defined by input signals that are either inverted or not inverted. Like the sum-of-products expression, this means that the maximum succession of gates that any input signal passes through before the effects of its value reach the output are a possible inverter, an OR gate, and an AND gate.

Let's look at a POS expression that uses four inputs: (A + B + C + D)·(A + B + D)·(A + D) (read A-bar OR B OR C OR D-bar AND-ed with A OR B-bar OR D AND-ed with A OR D-bar). If you're looking at the worksheet, you'll see that we are using parentheses to define the order of operations here. The OR operations are to be performed before the AND.

So, what does this expression look like in a truth table? Remember that if any input to an AND gate is a zero, the output is zero. Therefore, if we can identify the rows where each sum generates a zero, we have identified all the zeros in the final truth table. The rest of the rows will contain ones. Let's start with the first sum, A + B + C + D (read A-bar OR B OR C OR D-bar). The only time that a sum outputs a zero is if all its inputs are zero. For this sum, that happens when A-bar is zero, B is zero, C is zero, and D-bar is zero. There is exactly one row in the truth table where this situation occurs, and that is in the tenth row of the four-input truth table where A is one, B is zero, C is zero, and D is one.

Next, let's figure out where the second sum generates a zero in the truth table, in other words, when is A + B + D (read A OR B-bar OR D) equal to zero? This sum equals zero when A is zero and when B is one and when D is zero. But what about C? It isn't part of this expression. Since C is not part of this sum, it can be whatever it wants to be. Therefore, the sum places a zero in the truth table in two places: first,

where A is zero, B is one, C is zero, and D is zero, and second, where A is zero, B is one, C is one, and D is zero. This happens in the fifth and the seventh rows.

Now what about this last sum, A + D (read A OR D-bar)? This sum outputs a zero if A is a zero and D is not a zero. There are two input terms missing from this sum, B and C. That means that they can take on any value they want to as long as A equals a zero and D equals a one. There are four possible patterns of ones and zeros that B and C can take on: 0-0, 0-1, 1-0, and 1-1. Enumerating these four possible patterns along with A equals zero and D equals one gives us four rows where this sum places a zero in the output column: 0-0-0-1, 0-0-1-1, 0-1-0-1, and 0-1-1-1.

Like the products of a sum-of-products expression, when a sum in a POS expression utilizes every circuit input, it produces a single zero in the output column of the POS truth table. If one input is removed from the sum, two rows have zeros in the POS truth table. Two inputs missing generates four zeros in the truth table. Three inputs missing from a sum generates 2^3 or 8 zeros in the truth table, and so on.
As with sum-of-products expressions, there isn't a truth table around that cannot be represented with a POS expression. To show this, we're going to make up a truth table and derive its corresponding POS expression.

Before we do, let's talk about the OR operation. Remember that the truth table for an OR operation, regardless of the number of inputs, has exactly one row with a zero for its output. This is the row where all the inputs equal zero. Every other row has a one output. By using inverters on one or more of the inputs, we can move this single zero to any of the other rows.

For example, the truth table for A OR B OR C has a single zero in the top row where A is zero, B is zero, and C is zero. If we invert A in our expression to get A-bar OR B OR C, then that single zero moves to the row where A is not zero, in other words, it's one, B is zero, and C is zero. Now we have a truth table with a zero in the fifth row where the inputs are 1-0-0. If, on the other hand, we want to move that single zero to the seventh row where A is one, B is one, and C is zero, we would need to invert A and invert B to get the expression A-bar OR B-bar OR C. The truth table for each of these sums has a single zero in the output column.

Now think about the AND gate. The output of an AND gate is a zero if any of the inputs are zero. Therefore, when we AND two or more sums together, a zero appears in the output column for each of the zeros. For example, if we AND-ed together the three sums described earlier to get $(A + B + C) \cdot (A + B + C) \cdot (A + B + C$ (read A OR B OR C AND-ed with A-bar OR B OR C AND-ed with A-bar OR B-bar OR C), the resulting truth table would have a zero in the first row, a zero in the fifth row, and a zero in the seventh row with ones everywhere else.

It turns out that we can reverse this process to turn a truth table into an POS expression. All we need to do is identify the rows where there are zeros in the truth table, create the unique sum for each of those rows using inverses over the inputs to move the zero from the top row to the desired row, and then AND together the sums together. Note that each sum in this resulting POS expression will use all the variables for inputs, in other words, it won't be simplified. Unlike the episode where we simplified an SOP expression, we won't be simplifying the POS expressions here. In general, applying the distributive law will result in SOP expressions, even when starting with a POS expression.

Now for an example. Let's say we want to derive the POS expression for a three-input truth table with zeros in the first, third, sixth, and eighth rows. Since there are four zeros in our truth table, there will be

four sums, all of which will include the inputs A, B, and C, either inverted or non-inverted. Also note that this is the same truth table used in the example from Episode 5.01 – The Sum-of-Products Expression. Refer to that episode to see how the two procedures result in very different Boolean expressions. First, let's create a sum to place a zero in the first row where A is zero, B is zero, and C is zero. A truth table with a zero only in the first row should look familiar. It's just the straight OR of A OR B OR C. No inverters are needed to move the zero to a different row.

To place a zero in the third row, however, we will need one or more inverters. To place a zero in the third row, we need to have a sum that is zero when A is zero, B is not zero, and C is zero. That sum is A OR B-bar OR C.

To place a zero in the sixth row, where A is one, B is zero, and C is one, we need to invert A and C before running the signals into the OR gate. This gives us A-bar OR B OR C-bar. To place a zero in the bottom row where all three signals are one, we need to invert all three signals before sending them into the OR gate. This gives us the sum A-bar OR B-bar OR C-bar.

Now we have sums that place zeros in the first row, the third row, the sixth row, and the eighth row. If we AND them together, the zeros from each of the OR gates will be passed through to the output while logic ones will be placed in all the rows for which no zero is created by a sum. This gives us the Boolean expression for our truth table, $(A + B + C)\cdot(A + \bar{B} + C)\cdot(\bar{A} + B + \bar{C})\cdot(\bar{A} + \bar{B} + C)$ (read A OR B OR C AND-ed with A OR B-bar OR C AND-ed with A-bar OR B OR C-bar AND-ed with A-bar OR B-bar OR C-bar). Once again, please recognize the need for parentheses around each of the sums so that the OR's are performed before the final AND operation.

In Episode 5.02 – NAND Logic, we showed that by replacing the OR gate at the output of any SOP expression with a NAND gate with inverted inputs, we could implement an SOP expression entirely with NAND gates. A similar procedure can be performed with the POS expression. Using DeMorgan's Theorem, we can show that the AND gate truth table is the same as the truth table for the NOR gate with inverted inputs. If we replace the AND gate at the output of the POS expression with a NOR gate with inverted inputs, each one of the OR gates of the expression now has an inverter at its tip. Each of these OR gates with inverted outputs can be replaced with a NOR gate. This means that any POS expression can be implemented using nothing but NOR gates.

And just like the SOP expressions, since a POS expression can be created for any truth table, and since any POS expression can be implemented entirely with NOR gates, then we really only need one type of gate to implement a POS expression: a NOR gate. That means that like NAND gates, NOR gates are functionally complete.

In our next episode, we present a graphical method to create a most simplified sum-of-products expression from a truth table without the need for Boolean simplification. Since it's graphical in nature, you might want to consider downloading the episode worksheet before listening. 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.