Episode 6.07 – 7-Segment Display Driver Design

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.

Let's talk about a simple digital device. It's an old device, but we still see them popping up here and there. It's called a seven-segment display, and the most common place to see them in the wild is displaying time on an old clock radio or maybe a microwave oven. They are one of the easiest ways to implement a numeric output for a digital circuit. The use of seven-segment displays is so extensive that special integrated circuits or ICs have been developed to convert a four-bit binary digit into a set of seven digital signals that illuminate the digit on the display.

A seven-segment display consists of seven long, thin LEDs arranged in the shape of an eight. Each segment is controlled individually so that any decimal digit can be displayed by turning on or off specific LEDs. Using a combination of both upper- and lower-case letters, A, B, C, D, E, and F can be displayed as well making seven-segment displays capable of displaying hexadecimal digits too.

By convention, these seven LEDs have been identified using lowercase letters a through g. The top horizontal segment is 'a' with the outer segments being labeled clockwise around the perimeter: 'b' is the top right vertical segment, 'c' is the bottom right vertical segment, 'd' is the bottom horizontal segment, 'e' is the lower left vertical segment, and 'f' is the upper left vertical segment. 'g' is used to identify the horizontal segment in the middle.

To make a digit appear, the user must know which segments to turn on and which to leave off. For example, to display a '0', all of the segments except for the horizontal middle segment, g, should be turned on. To display a '1', we need to turn on the two rightmost vertical segments, b and c, and leave the other segments off. And so on.

For the purpose of this design discussion, we are going to drive our LEDs using active-high output signals from our circuit. That means that if we wish to display a '1', then the binary circuits driving segments b and c would output logic ones while the binary circuits driving segments a, d, e, f, and g would output logic zeros. If the binary inputs to the display are set to a=1, b=1, c=0, d=1, e=1, f=0, and g=1, in other words, all segments are on except for c and f, a '2' would be displayed.

What we need is a circuit that will convert the four bits of a nibble to the pattern of LEDs that when illuminated, displays the correct digit. This circuit is referred to as a seven-segment display driver, and it is the subject of this episode's design.

We need seven truth tables, one to control the 'a' output, one to control 'b', one to control 'c', and so on. Each digital circuit circuit takes as its input the binary nibble that is to be displayed. For example, if the binary nibble 0-0-1-0, which equals a decimal two, is input to the digital circuitry driving the display,

then the digital circuit for segment ‘a’ would output one, the digital circuit for segment ‘b’ would output one, the digital circuit for segment ‘c’ would output zero, and so on, so we can display a two.

In the past, we have identified our binary circuit inputs as A, B, C, D, and so on. This may cause some confusion in this example since we are also using the letters ‘a’ through ‘g’ to identify the segments and we will be displaying the hexadecimal digits A through F on the display. Therefore, we are going to label the individual bits of our four-bit binary nibble as B3, B2, B1, and B0 where the numeric identifier represents the power of two for that binary digit. In other words, B0 is the two to the zero-th position, B1 is the two to the first position, etc. These individual bits of the number to be displayed will serve as our circuit inputs.

Now that we’ve identified the circuit inputs, we need to know which segments to turn on and which are to be left off for each digit. This ends up generating a complex truth table with sixteen rows, one row for each digit, and eleven columns, four for the inputs and seven for the outputs for each segement. Describing this table with words may not be the easiest to follow, but we’re going to take a shot at it now.

Let’s start with the top segment, ‘a’, and as we go through the rows representing each hexadecimal digit from zero to F, we will decide whether the ‘a’ segment should be on or off. To begin, when the input is zero, in other words B3, B2, B1, and B0 all equal zero, the top segment should be on, so we put a one in the ‘a’ column. To display a one, only segments ‘b’ and ‘c’ are on, so ‘a’ has a zero for this row. When we have a digital input of 0-0-1-0, in other words, two, the top segment is on, so ‘a’ has a one for this row. The same is true for three which has all segments on except ‘e’ and ‘f’. Segment ‘a’ is off to display a four, but is back on for five, six, seven, eight, and nine.

Now we get to the hexadecimal characters. Before we decide if the ‘a’ segment is on, let’s decide which letters to capitalize and which to make lowercase. We need to do this because some of the letters, such as B and D, look like numeric digits when capitalized. B looks like an eight while D looks like a zero. Therefore, both of these letters should be displayed as lower case. The rest of the letters will be displayed in uppercase.

A capital A has the top segment on, so segment ‘a’ should equal one for this row. B is going to be lowercase, so all the segments should be on except ‘a’ and ‘b’ meaning that segment ‘a’ should be zero for this row. C has the top and bottom and two leftmost segments on. Therefore, segment ‘a’ should equal logic one. A lowercase D should have segments ‘a’ and ‘f’ off, driven by a logic zeros. E and F both have segment ‘a’ on, so these truth table rows should have ones. This gives us the output values for the segment ‘a’ going down the truth table as 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, and 1.

Although the episode worksheet has the full truth table, we’re going to do one more column here, specifically, segment ‘e’, the lower left vertical segment. This is the one for which we are going to make a sum-of-products expression. So how does segment ‘e’ behave with the different digital inputs? When all four bits of our input equal zero, ‘e’ should be on, so we put a one in its column. To display a one, only segments ‘b’ and ‘c’ are on, so segment ‘e’ has a zero for this row. For the digit two, segments ‘a’, ‘b’, ‘g’, ‘e’, and ‘d’ are on, so the ‘e’ column has a one for this row. To display three, segments ‘e’ and ‘f’ are off, so the ‘e’ column has a zero in this row. Segment ‘e’ is also off when displaying four or five, so there are logic zeros in these two rows for the ‘e’. Segment ‘e’ is on for six, so there is a logic one in this row for ‘e’, but off for seven which only illuminates segments ‘a’, ‘b’, and ‘c’. Eight has all segments on, so there is a one in the ‘e’ column. For nine, ‘e’ is the only segment that is off, so there is a zero here. As

for the hexadecimal digits, all six values, A, B, C, D, E, and F, have the ‘e’ segment on. This gives us the output values for the segment ‘e’ column of the truth table as 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, and 1. Please refer to the worksheet to see how the other segments behave based on the digital input to the circuit.

The next step is to create a Karnaugh map for each of the seven segments in order to determine the minimum SOP expression and digital circuit to be used to drive each segment. Here we will only do one of the circuits, the one for segment ‘e’. Using the process described in Episode 6.02 – Two- and Four-Variable Karnaugh Maps, we transfer the output values of the sixteen rows of our four-input truth table to the Karnaugh map. If we use B3 and B2 to identify the rows of the Karnaugh map and B1 and B0 to identify the columns, we get the following mapping.

There is a one in the top left cell where all four inputs equal zero. There are ones down all four cells of the rightmost column identified by B1 equals one and B0 equals zero. There are ones in all four cells of the third row where B3 and B2 are both equal to one. There is a one in the bottom left cell where B3 equals one and B2, B1, and B0 equal zero. There are also ones in the two rightmost cells of the bottom row where B3 equals one and B2 equals zero while B1 equals one.

Next, we need to identify the optimum set of rectangles for the Karnaugh map. We start with the ugliest of them, the four corners. Note that since a rectangle can wrap around from top to bottom and from left to right, all four corners of a four-variable Karnaugh map constitute a rectangle. Another way of looking at this is that the one in the top left corner has zeros adjacent to it on the right and below it. We can, however, double the size of the rectangle by wrapping around to the right side, then double it again by wrapping around to the bottom row. This is the largest rectangle we can have covering the one in the upper left corner.

To come up with the product that results from this rectangle, we look at the variables that stay constant across all four cells. When going from the left column to the right column, B1 changes, so it drops out. When going from the top row to the bottom row, B3 changes, so it too drops out. This rectangle covers the range for B2 equals zero and B0 equals zero, so the product is B2-bar AND B0-bar.

The ones down the rightmost column constitute a nice rectangle that crosses all four possible combinations of ones and zeros for B3 and B2, which means they drop out. This leaves B1 equals one and B0 equals zero, so the product for the right column is B1 AND B0-bar. The ones across the third row also constitute a nice rectangle that crosses all four possible combinations of ones and zeros for B1 and B0. This means they drop out. This leaves B3 equals one and B2 equals one, so the product is B3 AND B2 for the third row.

The one in the third column of the bottom row is the only one we haven’t yet covered with a rectangle. We can double the size of the rectangle covering this cell by pairing it with the cell to the right, and then doubling it again by pairing it with the ones above. This gives us a rectangle that crosses both values of B2 and both values of B0. Since both B3 and B1 equal one for this rectangle, the final product it represents is B3 AND B1. This gives us the final sum-of-products expression $B2 \cdot B0 + B1 \cdot B0 + B3 \cdot B2 + B3 \cdot B1$ (read as B2-bar AND B0-bar OR-ed with B1 AND B0-bar OR-ed with B3 AND B2 OR-ed with B3 AND B1).

Unfortunately, this design of the display driver is not complete. There are six more logic circuits to design, one for each of the remaining segments. We could, however, make our job easier by using those

little don’t care wild cards that do so much to simplify the products we derive from our Karnaugh maps. Can we apply don’t cares to this design? Well, yes. Like the design presented in Episode 6.05 – Don’t Cares, the Logical Kind, we could modify our design requirement to allow for only decimal digits. After all, who needs a hexadecimal alarm clock? (No comments from the geeks, please.) This would make it so the patterns representing A through F would be don’t cares, thereby placing X’s in all four cells in the third row of the Karnaugh map representing the truth table rows 1-1-0-0, 1-1-0-1, 1-1-1-1, and 1-1-1-0 and in the two right cells of the Karnaugh map’s bottom row for the truth table rows 1-0-1-1 and 1-0-1-0.

Typically, a don’t care will allow us to make larger rectangles, but in this case, it shows us that the rectangle covering the row where B3 equals one and B2 equals one is no longer needed. It also eliminates the rectangle covering the four cells in the lower right quadrant of the Karnaugh map where B3 equals one and B1 equals one, so the product for this rectangle is no longer needed either. This simplifies our sum-of-products expression that drives segment ‘e’ to just B2·B0 + B1·B0 (read as B2-bar AND B0-bar OR-ed with B1 AND B0-bar).

In our next episode, we are going to discuss some traditional logic circuits that do things like enable a device or route a digital signal. 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.