Project 2: Counter With Display

CS 200 • 20 Points Total
Due Friday, February 11, 2022

Objectives

- Design a modulo-10 (BCD) counter using JK flip flops.
- Design a BCD to 7-segment LED converter.
- Implement your designs using the Logisim software.
- Practice designing combinational and sequential circuits.

Overview

At every clock pulse a JK flip flop outputs one of the following four values based on its contol inputs: a one, a zero, the same thing it output during the last clock cycle, or the opposite of what it output during the last clock cycle.

This makes it an ideal building block for various types of counters because we have something that can be reset when necessary and subsequently toggled at will. Each bit of a counter can be represented by one JK flip flop, and we can have each bit be set up to toggle if all the bits to the right are set to 1 **AND** a clock pulse happens.

A standard 4-bit counter could be called a modulo-16 counter, since it counts 0-15 (0000-1111) and then resets to zero. For part of this project we want to design a modulo-10 counter that counts 0-9 (0000-1001) and then resets to zero on the next count. This could also be called a Binary-Coded Decimal (BCD) counter since its value can always be represented by a single base-10 digit.

To display the output from a modulo-10 counter we can use a seven-segment display. The trick with these is that every segment is controlled via a separate output from a combinational logic circuit. So in the same way that *sum* and *carry* are separate functions for an adder, so too are *top-right segment, top segment, center segment*, etc. for a 7-seg display. You'll build a 4-bit BCD value to 7-seg converter as part of this project.

Requirements

BCD To 7-Seg Converter

Write a truth table that lists all combinations of a 4-bit input and has 7 columns of output - one for each of the segments on a 7-segment display. In each row of input, identify whether each segment should be on or off. Or, you could make 7 truth tables, each with 4 inputs and a single output, if combining them into one big table is confusing.

Write functions that describe the behavior of each segment. The way to do this is to make a K-map for each output segment. Remember, there are four bits of input, so there are 16 combinations of input to set to a 4-variable K-map.

For example, the top right segment (hereafter segTR) would be turned on for every number 0-9 except 5 and 6. So you would put 1 in every position except 5 and 6 (and you could make 10 – 15 'don't cares' since your counter is perfect and there is no chance that they will accidentally appear as inputs. The inputs are numbered according to their place in the digit, so x3x2x1x0, and the K-map would look like:

x3x2 \ x1x0	00	01	11	10
00	1	1	1	1
01	1		1	
11	X	X	X	X
10	1	1	X	X

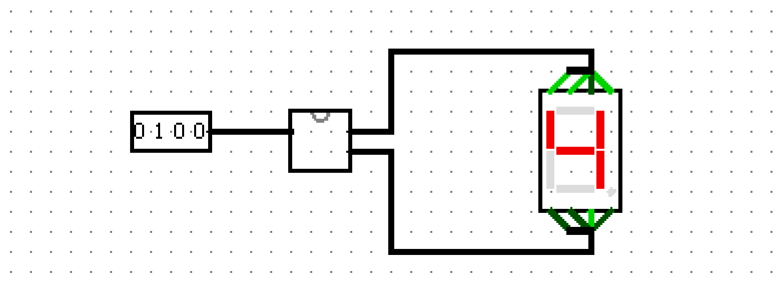
You can do the grouping yourself to see that the output would be x3 + x2' + x1'x0' + x1x0. But possibly you aren't as confident about your counter rejecting inputs that correspond to 10 – 15, so you replace the 'don't cares' with 0s to make sure that you don't display on bad inputs. Now your K-map would look like:

x3x2 \ x1x0	00	01	11	10
00	1	1	1	1
01	1		1	
11				
10	1	1		

And the output would be x3'x2' + x2'x1' + x3'x1'x0' + x3'x1x0, which is a bit more complicated but safer. Either way is fine by me. I've given you the equation for the top right, but the other six segments are up to you. By the way, there's actually an eighth segment, the decimal point, but you can just leave that off for all numbers.

You can create seven different circuits to implement your equations in Logisim. Each would accept four inputs and have one output. Then you could combine them into one circuit that has four inputs and seven outputs. The seven outputs can then be connected to a 7-seg display in Logisim. The four inputs can be driven by a 4-bit input to allow you to test all combinations to be sure that they display correctly.

The image on the next page shows this test circuit, except I bundled the outputs inside the combined circuit to make two 4-bit outputs (the bottom one also contains the decimal point output which is always 0). Then I used a fan-out to unbundle them before connecting them to the display. You don't have to do exactly like I did. If bundling and using the fan-outs confuses you, you can keep them un-bundled. Likewise, if combining circuits is tough, you can make one big circuit; I won't penalize you.



In the figure above, the input into the display circuit (shown as an abstract logic block) is a 4-bit wire bundle. There are two outputs, each also consisting of a 4-bit wire bundle. The outputs are unbundled at the display using a splitter with wires to each of the display inputs. Notice that the bundles are black but the split wires are black if 0 or green if 1, and correspond to the lit or dark segments of the display.

Modulo-10 Counter

In Logisim, design a modulo-10 counter using JK flip-flops. The book shows a schematic of a modulo-16 counter on page 142; this can serve as the basis for your design. I provide this base circuit in Logisim format for you to use.

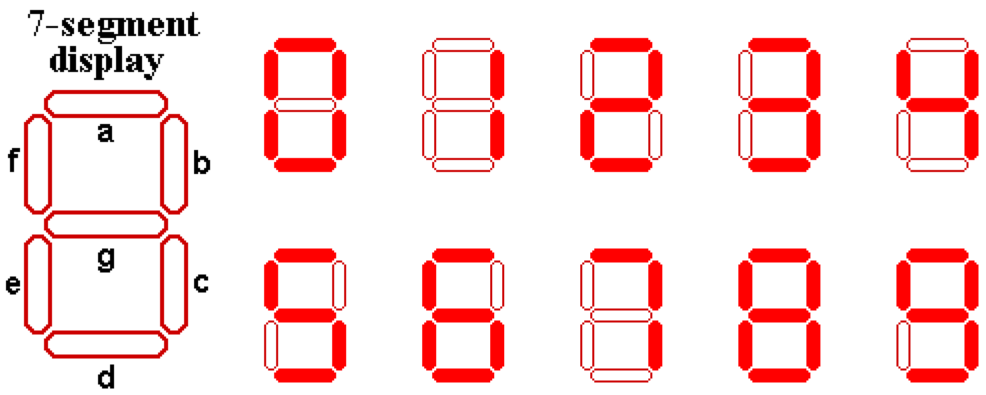
Notice that the JK flip-flops in Logisim have two blue input pins on the bottom. These are "set" on the left and "clear" on the right. These are both asynchronous - for instance, when "clear" becomes active, the flip-flop will change its output to zero no matter what the J, K, or clock inputs are.

There are several approaches you can use to making the counter modulo-10. The easiest would be to trigger the asynchronous clear on all the flip-flops if the count is at 9 AND the clock is high. However, that doesn't prevent the counter from being asynchronously set to a number higher than 9. Whatever you come up with, be sure to describe it in your report.

Bundle the 4 output wires of your counter together and hook them to the test circuit from above in place of the 40bit input. Hint: be sure the inputs are in the right order. It is not unusual for the problem to be that the inputs have been connected in reverse order. Hit CTRL+T to step through the different values of your counter!

Project Report

The final step of this assignment is to create a report consisting of a cover page, an overview of the project, sample output, and the source code. See Assignment Policies on either the class website or Bb Learn.



The segment names above don't match the numbering in Logisim. You should test to see which inputs turn on which segments for the simulated display in Logisim.