Project 5 Report

Thomas Kaneshige, Lab 7

Introduction and Requirements

In this lab, we were tasked with constructing another finite-state machine (FSM)—this time a parking meter. This parking meter consisted of the basic functionalities, excluding any operations related to accepting currency. This machine could add time and reset to a certain time, all while decrementing the total time in the system and displaying the time remaining at a specified interval.

Unlike the previous FSM lab where we constructed a vending machine, the spec for this lab did not provide us with a skeleton to construct the machine or any direction as to what the states of the machine should be. In the end, the final implementation of this machine ended up having no ‘real’ states, but it did contain some semblance of a state machine.

Design Description

As mentioned in the introduction, the final implementation of this parking meter did not end up having any defined states, partly due to how the machine should behave when given an input. In particular, there was no input that should be blocked because the machine is in a certain state (aside from the inputs given during a reset). In other words, all inputs are accepted at all times, regardless of the machine state. If we were to draw a state diagram, it would look something like this:

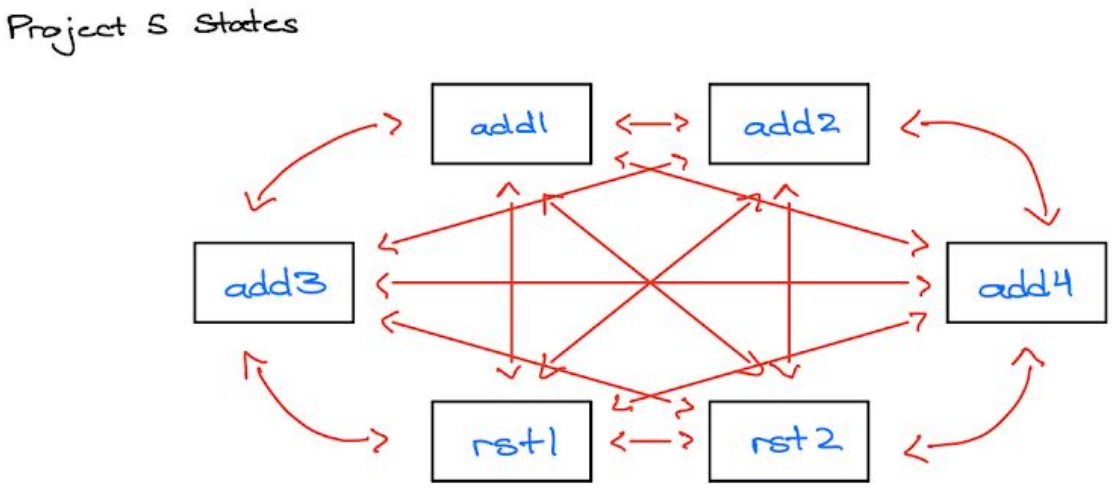

Figure 1. State Diagram for the Parking Meter.

As shown above, each ‘state’ can switch to any other state in the machine, which essentially eliminates any need to have states in the first place. Since there is no need for well-defined states to this machine, the design of this particular implementation is better described using the individual ‘modules’ that provide the main functionalities of the system.

1. Input Module
    a. This module deals with all the inputs given by the user. In addition, this module also completes the corresponding action associated with that input. For example, if the input ‘add1’ is asserted by the user, 60 seconds will be added to the internal counter. This module operates on the input clock given by the user, since using

the divide-by-100 clock would result in inputs failing to register. In addition, for the operations associated with adding time to the internal counter, this module also checks for potential overflow before increasing the counter.

2. Counter Module
    a. This module serves one simple function: decrement the internal counter of the parking meter. This module uses the divide-by-100 clock divider instead of the input clock, because the counter decrements at a slower rate (1Hz) compared to the input clock (which is 100Hz). This module also verifies that the internal counter is not 0 before decrementing it, in order to prevent overflow.
3. Output Module
    a. This module performs more of an auxiliary function; it produces a parse of the internal counter. In this module, the decimal digit values are split into individual registers and returned to the user, since it is easier to read binary-coded decimal (BCD) than anodes and LED segments.
4. Display Module
    a. This module takes the individual BCD digits from the output module and maps it to format using anodes and LED segments. The output ‘led_seg’ is also shared among the four BCD digits of the internal counter. Thus, in order to change the display digits, the four anodes (‘a1’, ‘a2’, ‘a3’, ‘a4’) corresponding to each digit on the LED segment must be asserted at separate times in order to create a 4 digit number. In other words, you can only change one digit of the LED segment at a time (in this implementation). The format of the LED segments is shown below, with the binary bits of ‘led_seg’ in the format of ABCDEFG.

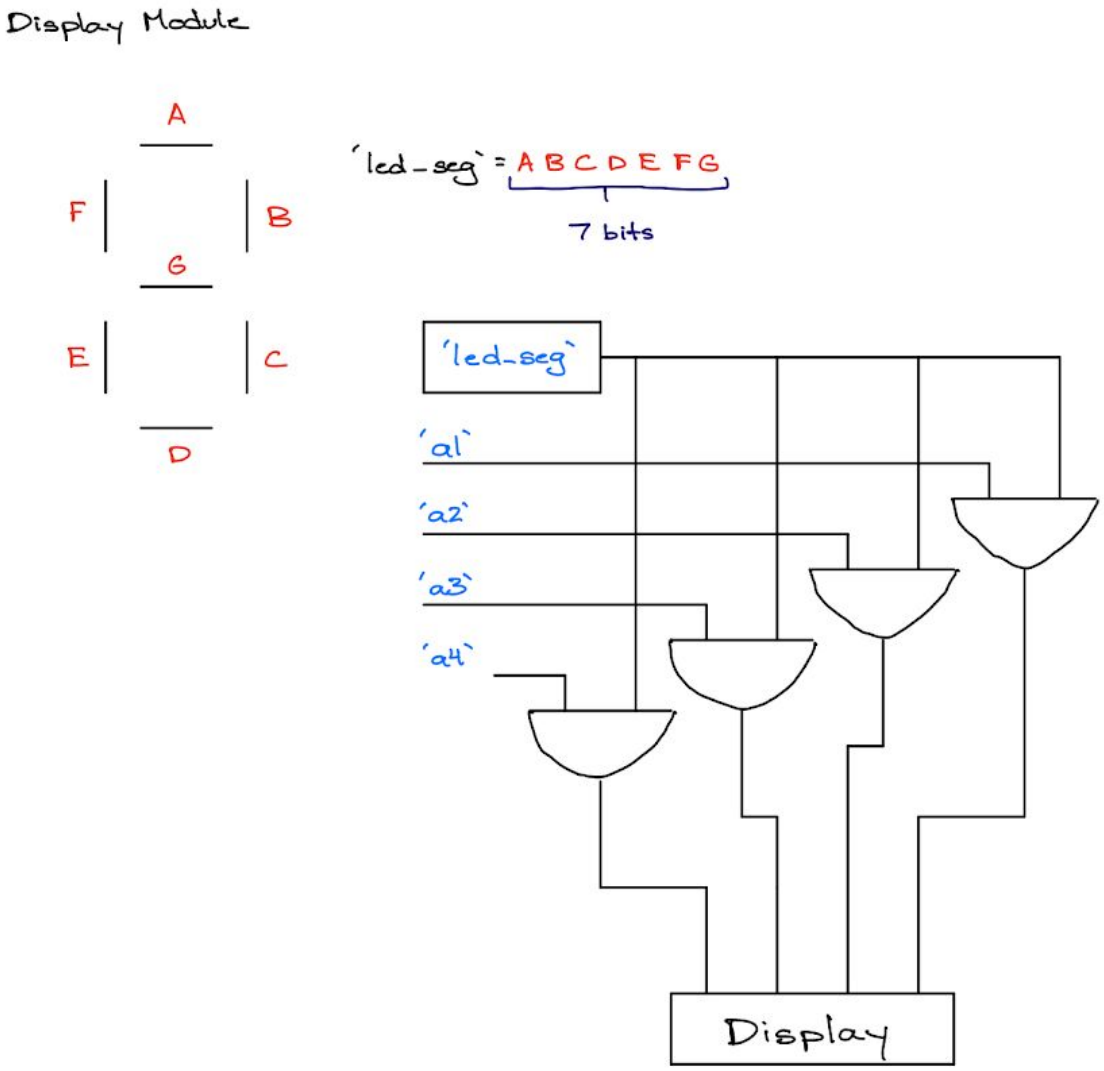

Figure 2. Format of the Display Module. Note: The LED segments are implemented using an active-low logic system. This means that '0' represents a segment that is on, while a '1' represents a segment that is off.

5. Divide-by-100 Module
    a. This module is exactly what it sounds like. It is a clock divider that takes an input clock and divides it by 100. This module is for the internal counter, because it is supposed to decrement at a rate of 1Hz, but the input clock operates at 100Hz.

Simulation Documentation

1. Test 1 Series: The test 1 series simply verifies that the inputs of the parking meter work correctly. For example, if an input were to add a specified time to the internal counter of the parking meter, the test 1 series would verify and confirm the expected behavior. Below are some short descriptions of each test.
    a. Test 1a: Verifies that the ‘add1’ input adds 60 seconds to the internal counter.
    b. Test 1b: Verifies that the ‘add2’ input adds 120 seconds to the internal counter.
    c. Test 1c: Verifies that the ‘add3’ input adds 180 seconds to the internal counter.
    d. Test 1d: Verifies that the ‘add4’ input adds 300 seconds to the internal counter.
    e. Test 1e: Verifies that the ‘rst1’ input resets the internal counter to 16 seconds.

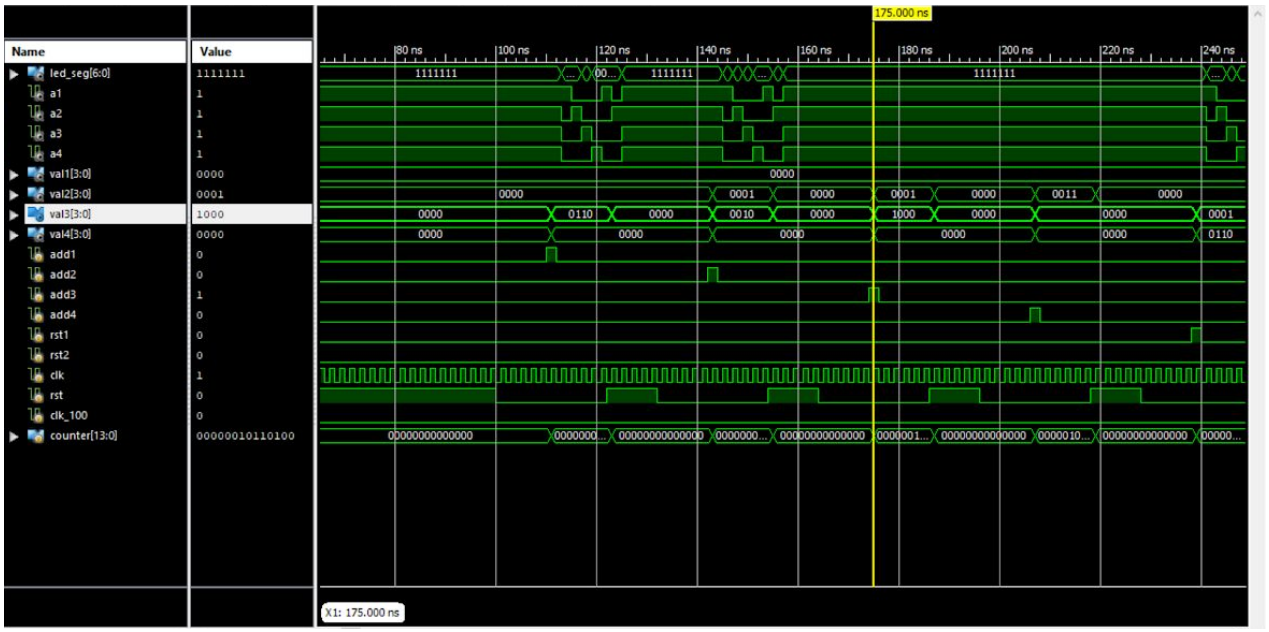

Figure 3. Simulation Documentation for Test 1c. Notice how when ‘add3’ is asserted, the value ‘180’ appears in the internal counter at the bottom and in BCD in ‘val1’, ‘val2’, ‘val3’, and ‘val4’. The other test cases in this series follow the same pattern as the one shown above (some of the other test 1 series test cases are also visible in the image as well).

2. Test 2 Series: The test 2 series verifies that the display output behaves correctly. For instance, when the internal counter has some value between 0 and 180, the spec requires that the clock flashes with a period of 2 seconds with a 50% duty cycle. Otherwise, the clock should flash with a period of 1 second with a 50% duty cycle. Below are more in-depth descriptions of each test.
    a. Test 2a: Verifies that the display output flashes '0000' with a period of 1 second with a 50% duty cycle.
    b. Test 2b: Verifies that the display flashes the value of the internal counter with a period of 2 seconds with a 50% duty cycle when the value of the internal counter is between 0 and 180 seconds.
    c. Test 2c: Verifies that the display flashes the value of the internal counter with a period of 1 second with a 50% duty cycle when the value of the internal counter is 180 seconds or more.
    d. Test 2d: Verifies that the display correctly transitions between the two differing periods for the two ranges of values (0 < counter < 180 and counter >= 180).

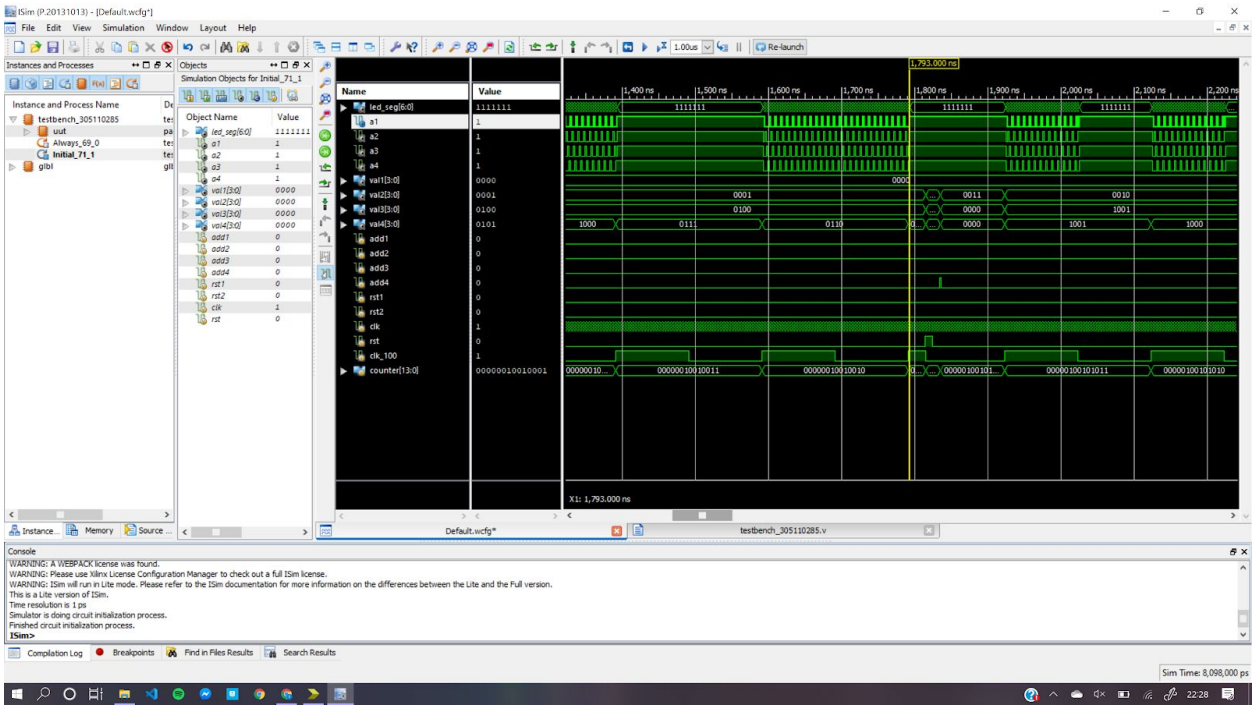

Figure 4. Simulation Documentation for Test 2b and Test 2c. In the image above, on one side of the yellow line, the display flashes with a period of 2 seconds (proportionally) with a 50% duty cycle. Notice how that period is twice as long as the cycle on the right, where the display flashes with a period of 1 second with a 50% duty cycle. This image well-represents the test cases in the test 2 series.

3. Test 3 Series: The test 3 series verifies that the counter never overflows and caps at the value '9999'. For example, if the user attempts to add more time to the parking meter when the internal counter of the parking is nearly maxed out, the internal counter should increase to '9999', but never overflow. Below are some short descriptions of each test case used to test this.
    a. Test 3a: Verifies that 'add1' does not cause any overflow.
    b. Test 3b: Verifies that 'add2' does not cause any overflow.
    c. Test 3c: Verifies that 'add3' does not cause any overflow.
    d. Test 3d: Verifies that 'add4' does not cause any overflow.

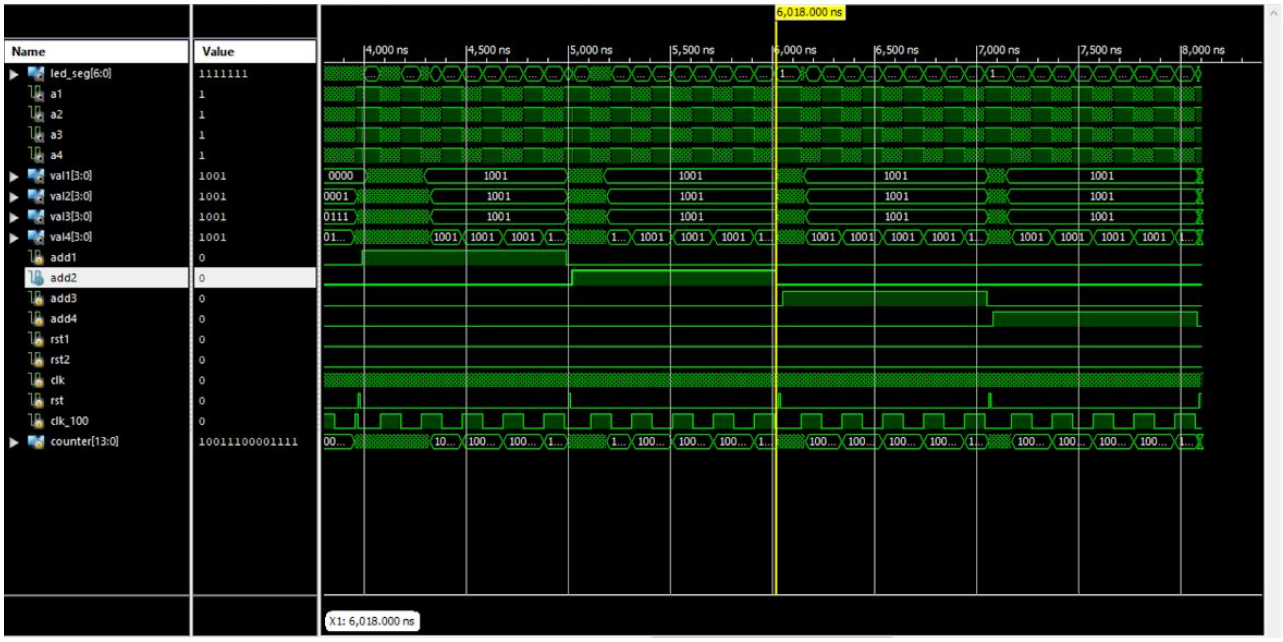

Figure 6. Simulation Documentation for the Test 3 Series. All 4 test cases are shown in the picture above. Notice how even though time is continually added in each separate test case (the counter is reset between cases), the counter never overflows and instead caps at 9999, as shown by 'val1', 'val2', 'val3', and 'val4' which encode the digits of the counter into BCD.

Conclusion

In the end, this particular implementation for the parking meter ended up not being a true finite-state machine. The states were optimized out in the development process, which ended up making the project much simpler and easier to read. The final implementation was also much more modularized, which made debugging and explaining the process much easier.

Overall, this lab was filled with many difficulties and many adjustments had to be made. Originally, I implemented this machine with the same concepts used for the vending machine in the previous lab, but I quickly ran into a wall when the states did not yield the same advantages as those used in the vending machine. The display also presented numerous difficulties. It was difficult to understand how to produce the required clocks, simply because I was unsure of what the waveforms for the necessary clocks were supposed to look like.