

CS M152A Lab 4: Finite State Machine - Parking Meter

Arnold Pfahnl

TA: Mohit Garg, Winter 2021

Contents

1	Introduction	1
2	Parking Meter Design	2
2.1	Schematics	4
2.2	Design Summary Report	5
3	Simulation	7
4	Conclusion	9

1 Introduction

In this lab we utilize a finite state machine (FSM) to create a parking meter encapsulated by the Verilog module, *parking_meter*. The functionality of this parking machine module includes simulating the ability to load coins and displaying the appropriate time remaining to four seven-segment LED displays. The required inputs are shown in Table 1 below.

Inputs	Function
<i>add1</i>	Add 60 seconds.
<i>add2</i>	Add 120 seconds.
<i>add3</i>	Add 180 seconds.
<i>add4</i>	Add 300 seconds.
<i>rst1</i>	Reset time to 16 seconds.
<i>rst2</i>	Reset time to 150 seconds.
<i>clk</i>	100 Hz system clock.
<i>rst</i>	Resets to initial state.

Table 1: Inputs to *parking_meter* module. Each add and reset button is high for at most one clock cycle.

The output module has a seven-segment vector *led_seg* that displays the actual value fed to the four segments corresponding to the digits being displayed. Each segment in the vector represents CA, CB, CC, CD, CE, CF, and CG in Figure 1. Additionally, *a1*, *a2*, *a3*, and *a4* are one-bit anodes that act as enables for their corresponding digits. As a form of redundancy, *val1*, *val2*, *val3*, and *val4* are each four-bit output ports that display their corresponding digit in binary coded decimal (BCD).

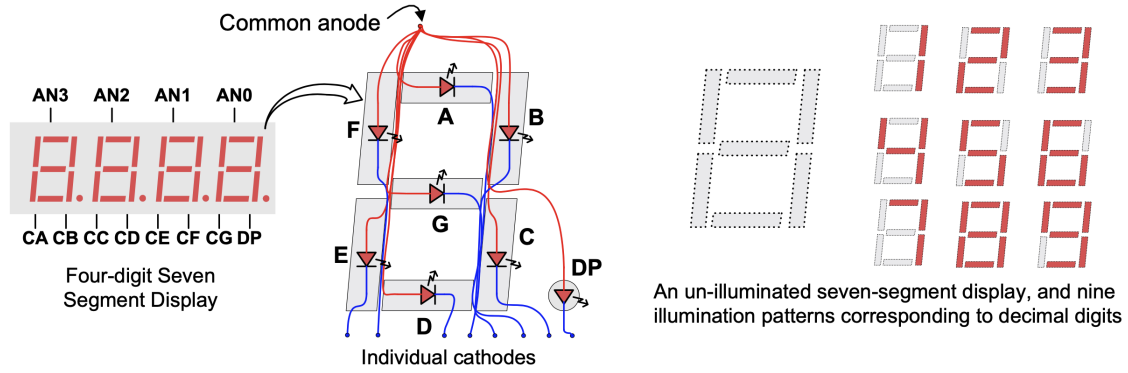


Figure 1: The four-digit seven-segment display specification for the Nexys3 board from Diligent Doc: 502-182.

2 Parking Meter Design

The states of the parking meter are based on the FSM diagram in Figure 2. An internal register called *counter* stores the countdown. The three main states are INITIAL, ONE, and TWO:

1. The **INITIAL** state flashes 0000 on the four seven-segment displays with a period of 1 second and duty cycle 50%. If the *add4* signal goes high, the machine enters the ONE state. If any other *add* state goes high, the machine enters the TWO state. If *rst* goes high, the machine will enter this state regardless of any other condition.
2. The **ONE** state flashes the four-digit integer value of the *counter* register with a period of 1 second and 50% duty cycle. Once the value of *counter* drops under 180, the machine will enter state TWO. If *rst1* or *rst2* are enabled, the machine will enter this state.
3. The **TWO** state flashes the four-digit integer value of the *counter* register with a period of 2 seconds and 50% duty cycle. If the value of *counter* is ever greater or equal to 180 due to an *add*, the machine will enter the ONE state. If the time expires (*counter* is 0), then the machine loops back to the INITIAL state.

Note that the machine does not account for multiple inputs being pressed at the same time.

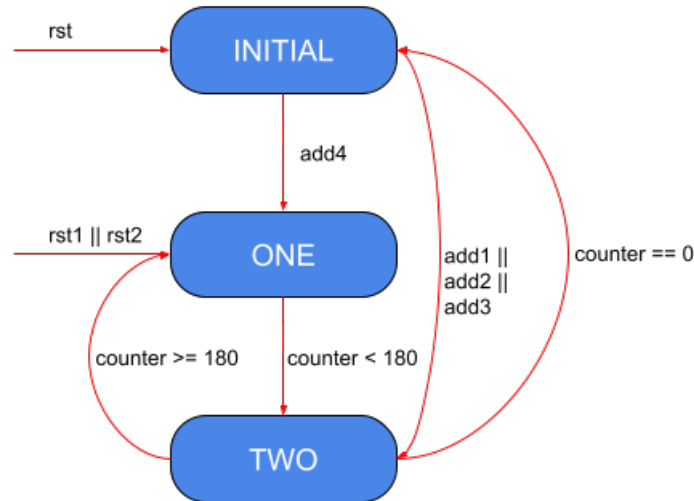


Figure 2: Finite state machine (FSM) diagram for the *parking_meter* module.

To show a four-digit number, the Nexys3 reference recommends a scanning display controller. The anode signals and cathode patterns of each digit are driven in a repeating, continuous succession. This pattern is visualized in Figure 3.

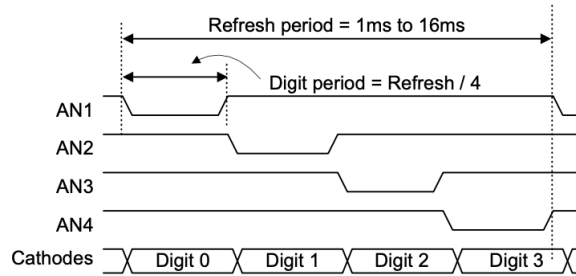


Figure 3: Four-digit display signal action for the Nexys3 board from Diligent Doc: 502-182.

In my implementation an internal register *digI* is set to the digit to be illuminated every one clock cycle. *digI* begins at 0, increments by one up to 3 and then resets to 0. A case statement then checks *digI* to route the appropriate logic for setting the digit cathode and anode variables. Since each digit gets one cycle in my implementation, it takes 4 input cycles to get through all digits making the overall cycle 25 Hz.

I also have two internal registers *fast_cycle* and *slow_cycle* that count up to 100 and 200 respectively and act as pseudo-clocks. To flash for a period of one second and a duty cycle of 50%, I display the values up until *fast_cycle* reaches 50 and then stop displaying until the next time *fast_cycle* loops back to 0. The same logic applies for flashing for a period of two seconds and 50% duty cycle, just with *slow_cycle*.

2.1 Schematics

The top level schematic in Figure 4 hides the considerable complexity of the RTL schematic in Figure 5. Due to the combination of both combinational and sequential logic used to determine states and output, the RTL features a mess of registers, muxes, and many other gates.

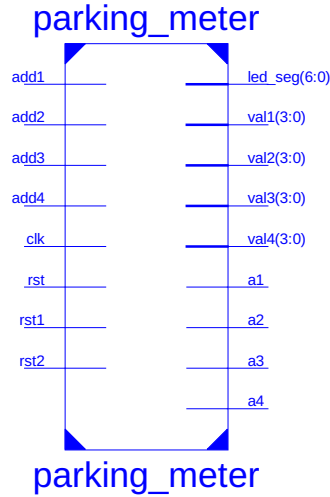


Figure 4: Top level schematic for the *parking_meter* module.

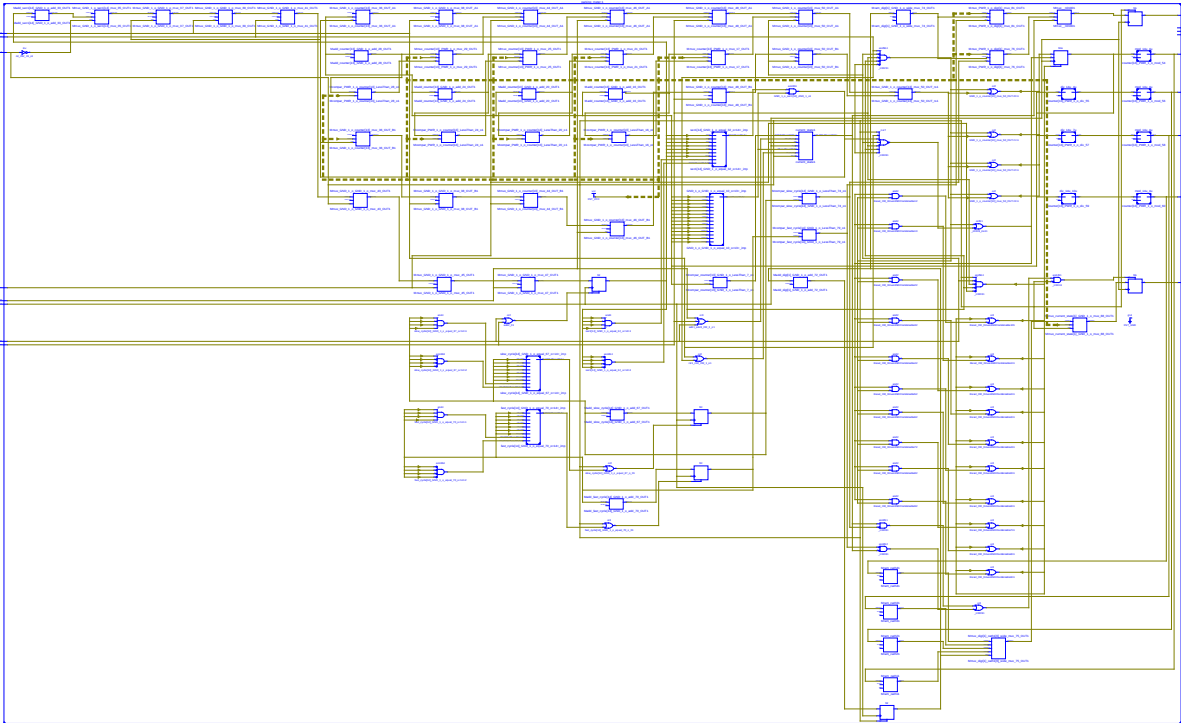



Figure 5: Detailed RTL schematic for the *parking_meter* module.


2.2 Design Summary Report

parking_meter Project Status (03/14/2021 - 15:46:45)			
Project File:	parking_meter.xise	Parser Errors:	No Errors
Module Name:	parking_meter	Implementation State:	Programming File Generated
Target Device:	xc6slx16-3csg324	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	No Warnings
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary [+]				
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	102	18,224	1%	
Number used as Flip Flops	102			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	1,317	9,112	14%	
Number used as logic	1,310	9,112	14%	
Number using O6 output only	1,175			
Number using O5 output only	62			
Number using O5 and O6	73			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number used exclusively as route-thrus	7			
Number with same-slice register load	0			
Number with same-slice carry load	7			
Number with other load	0			
Number of occupied Slices	441	2,278	19%	
Number of MUXCYs used	324	4,556	7%	
Number of LUT Flip Flop pairs used	1,325			
Number with an unused Flip Flop	1,232	1,325	92%	
Number with an unused LUT	8	1,325	1%	
Number of fully used LUT-FF pairs	85	1,325	6%	
Number of unique control sets	9			
Number of slice register sites lost to control set restrictions	34	18,224	1%	
Number of bonded IOBs	35	232	15%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	0	64	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	

Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of BUFG/BUFGMUXs	1	16	6%
Number used as BUFGs	1		
Number used as BUFGMUX	0		
Number of DCM/DCM_CLKGENs	0	4	0%
Number of ILOGIC2/ISERDES2s	0	248	0%
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%
Number of OLOGIC2/OSERDES2s	0	248	0%
Number of BSCANs	0	4	0%
Number of BUFHs	0	128	0%
Number of BUFPLLs	0	8	0%
Number of BUFPLL_MCBs	0	4	0%
Number of DSP48A1s	0	32	0%
Number of ICAPs	0	1	0%
Number of MCBs	0	2	0%
Number of PCILOGICSEs	0	2	0%
Number of PLL_ADVs	0	2	0%
Number of PMVs	0	1	0%
Number of STARTUPs	0	1	0%
Number of SUSPEND_SYNCs	0	1	0%
Average Fanout of Non-Clock Nets	4.96		

Performance Summary 			
Final Timing Score:	0 (Setup: 0, Hold: 0)	Pinout Data:	Pinout Report
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report
Timing Constraints:	All Constraints Met		

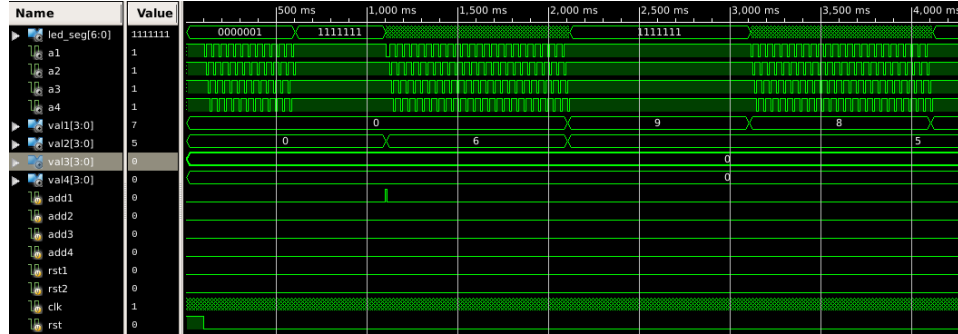
Detailed Reports 					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sun Mar 14 15:45:33 2021	0	0	8 Infos (3 new)
Translation Report	Current	Sun Mar 14 15:45:42 2021	0	0	0
Map Report	Current	Sun Mar 14 15:46:03 2021	0	0	6 Infos (0 new)
Place and Route Report	Current	Sun Mar 14 15:46:13 2021	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Sun Mar 14 15:46:18 2021	0	0	4 Infos (0 new)
Bitgen Report	Current	Sun Mar 14 15:46:41 2021	0	0	0
Report Name	Status	Generated			
ISIM Simulator Log	Out of Date	Sun Mar 14 15:44:59 2021			
WebTalk Report	Current	Sun Mar 14 15:46:41 2021			
WebTalk Log File	Current	Sun Mar 14 15:46:45 2021			

Date Generated: 03/14/2021 - 15:46:45

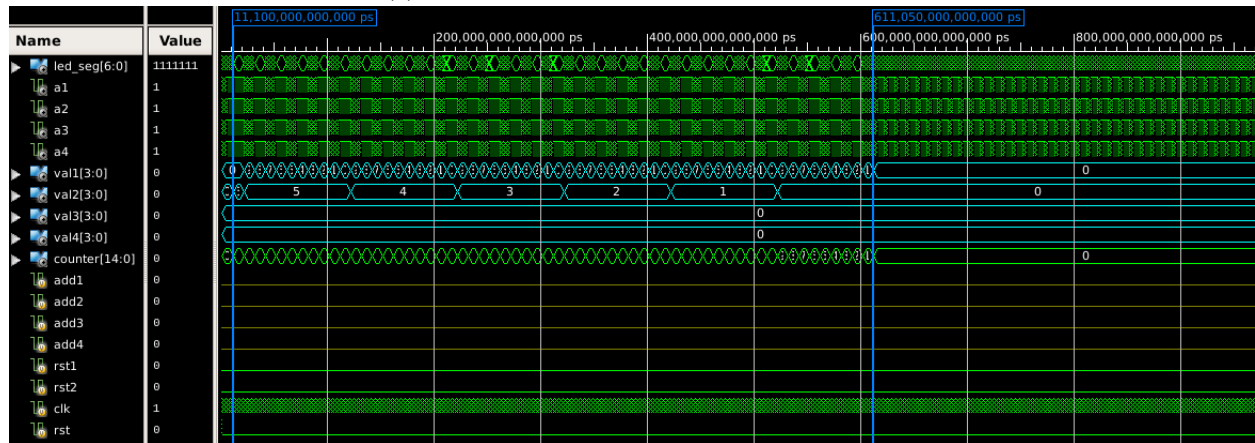
3 Simulation

The following test cases encompass all possible functions of the parking meter in addition to some corner cases.

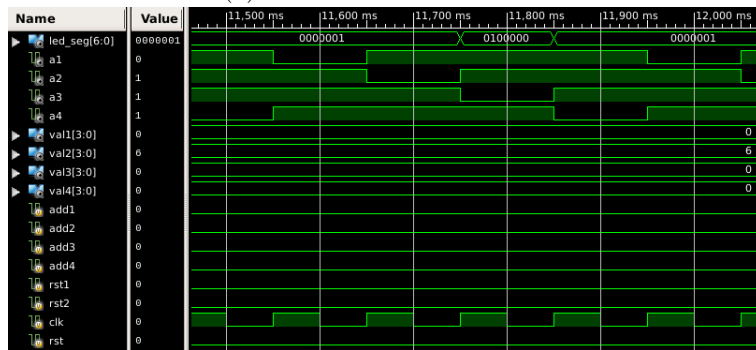
1. **Add 60 seconds.** This test encompasses many base cases including making sure that the INITIAL state flashing is working, the TWO state flashing is working properly, global reset works, adding works, and that the countdown is working. There were no issues with this test case.



(a) Initial countdown after adding 60 seconds.



(b) Full 60 second countdown.



(c) Zoomed in view with the scanning display values.

Figure 6: Different views of the 60 second addition waveform (*add1* signal).

2. **Add 120 seconds.** This test case is a continuation of the previous test case to make sure that *add2* is working properly. This test case also had no issues.

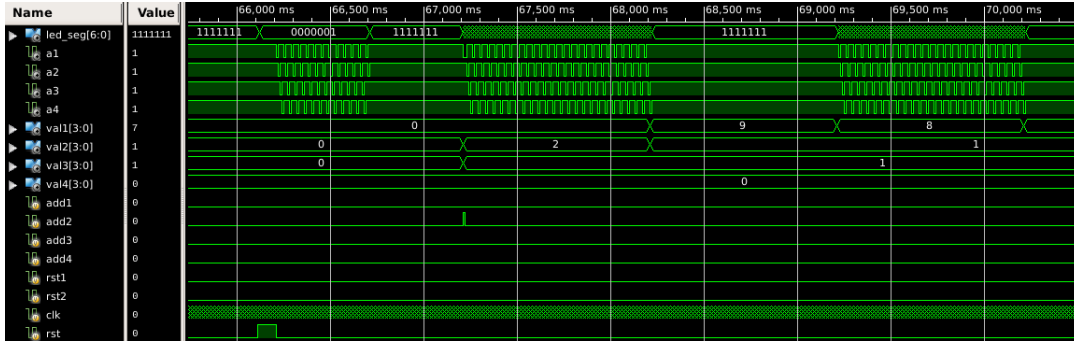


Figure 7: Testing the *add2* signal.

3. **Add 180 seconds.** Here I check that *add3* is working properly. This test case passed with no issues.

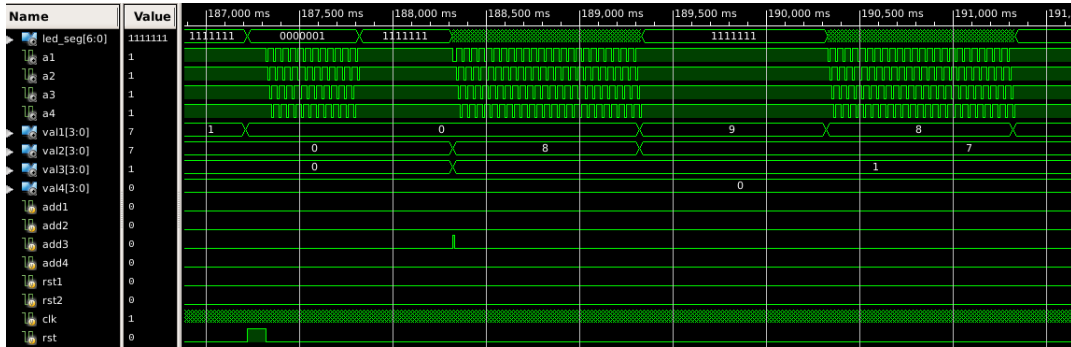
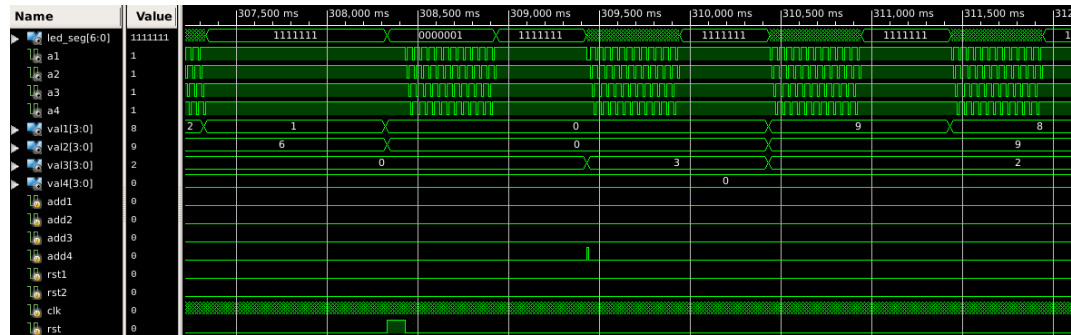
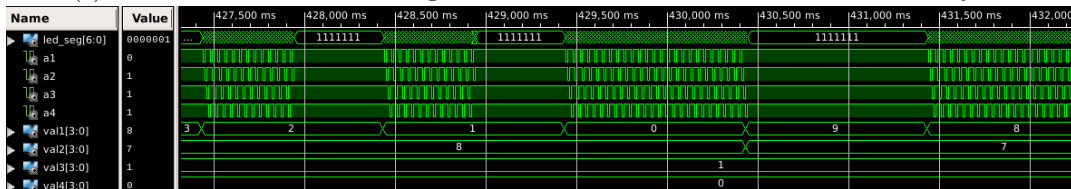


Figure 8: Testing the *add3* signal.

4. **Add 300 seconds.** Here I check to make sure that *add4* enters the ONE state and transitions from the ONE to TWO state correctly. This test case passes as expected.



(a) Initial countdown after adding 300 seconds. This section has a 1 second flash cycle.



(b) ONE to TWO state transition. 180 seconds and lower have a 2 second flash cycle.

Figure 9: *add3* signal test case waveforms.

5. Add 300, reset time to 16 seconds, then reset time to 150 seconds. Here I check that *rst1* and *rst2* work and that the transition from ONE to TWO state is working. This test passes as expected.

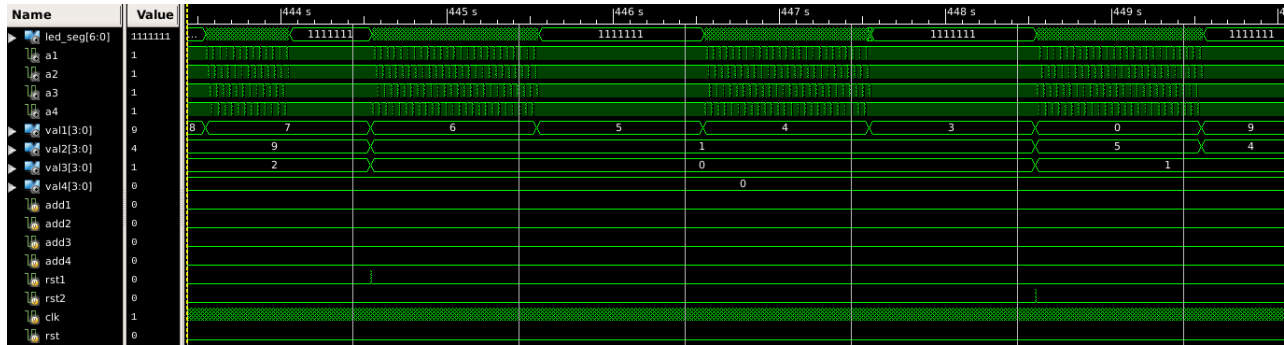


Figure 10: Testing *rst1* and *rst2*.

6. Add beyond 9999 seconds. Here I check that the counter doesn't exceed the maximum allowed value. This works as expected.

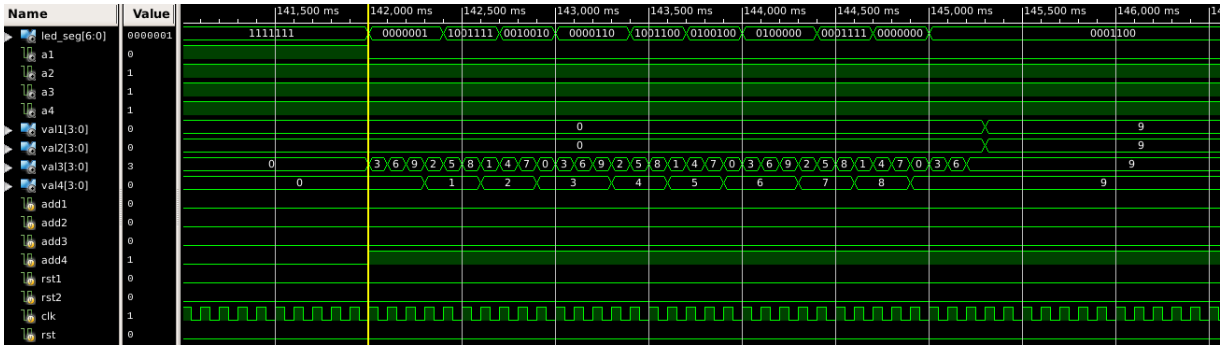


Figure 11: Testing the maximum display value and over-adding.

4 Conclusion

In this project, I successfully implemented a finite state machine for a parking meter. This project was more difficult than it first seemed, but I was able to learn a lot, as well as apply many of the concepts I learned from previous labs into this project.

The most difficult aspect of this project for me was implementing the flashing logic, and making sure that state transitions would react properly to the many different button presses. Writing out all the button functions and drawing out their relations helped tremendously.

Overall, I thought this lab was a great final project for this class. I've learned a lot about practical digital design and how FPGA design works in conjunction with Verilog. From this class, I think programming from a hardware perspective is the skill I gained the most. Thank you Mohit for being a great and supportive TA for all these projects. I really appreciated your prompt and helpful replies to all my questions.