CS M152A Project 4 Report: Design a Parking Meter

Belle Lerdworatawee, Lab 1

TA: Ananya Ravikumar

Mar 15, 2021

Introduction and Requirement:

In this lab, students incorporate the knowledge that they've gained over the quarter. The task is to design a finite state machine (FSM) that is a parking meter. A user pushes buttons to add time to the parking meter which displays the remaining time before the meter expires. Different buttons represent different coin denominations; meter time is incremented as soon as a button is pushed and decrements every second. Time is displayed on 4 seven-segment LED displays. The parking meter flashes 0000 every half a second in the initial state since there are 0 seconds remaining. When the time remaining is greater than 0 but less than 180 seconds, the display flashes the time every second and shows only the even seconds. When there are more than 180 seconds left, the time is always displayed. 9999 seconds is the maximum amount of time that can be allotted. The parking meter connects anodes and cathodes to the LED displays in order to show the time. More details on the machine will be covered in the following section.

Design Description

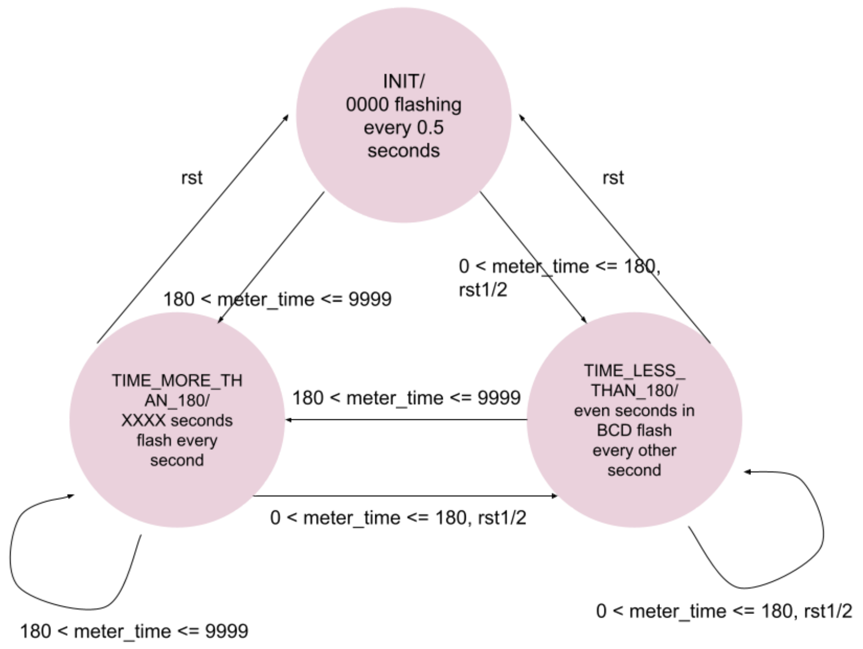


Figure 1: Parking Meter Finite State Machine

I implemented the parking meter using several modules: the top module is parking_meter. The other three submodules I designed were a bcd converter, and 2 counters that count to 100 and

4. I created the submodules in order to keep the parking meter module as clean as possible, and only have code relevant to the functions of a parking meter. This enabled me to distinguish between always blocks that were necessary for the FSM and always blocks that assisted in transmitting extra signals or helped to control timed events. The FSM itself consists of 3 states which I will detail below. In each state, when rst goes high they will return to the INIT state.

States:

1. INIT:
    - machine initializes in this state after rst goes high; represents the state where no one is using the parking meter
    - sets all anodes to high for half a second and then turns all anodes low for a half of a second
    - sets led_seg to 1000000 which is the vector for the number 0
    - waits for button presses (addX) to move to state 1 if the meter time is less than 180 or to state 2 is the time is greater than 180
    - waits for rst1 or rst2 to go high to move to state 1
2. TIME_LESS_THAN_180
    - once a coin has been inputted, the machine counts down every second
    - checks if meter_time is 0 or if rst is high in which it will return to state 0
    - checks if meter_time is even
        i. sets 1 anode at a time to be low for 10ms and rotates which anode is low
        ii. changes led_seg to the corresponding digit
    - moves to state 2 if addX buttons are pushed and meter_time goes above 180
3. TIME_MORE_THAN_180
    - once a coin has been inputted, the machine counts down every second
    - waits for rst1 or rst2 to go high to move to state 1
    - returns to state 1 if meter_time drops below 180
    - sets 1 anode at a time to be low for 10ms and rotates which anode is low
    - changes led_seg to the corresponding digit

Modules and Submodules:

1. parking_meter: This module takes in 8 inputs—clk_in, rst, add1, add2, add3, add4 which represent the clock, reset, and 4 different coins. Then it gives 9 outputs—led_seg (a 7 segment vector), an1-4 (4 anode wires), and val4-1, four 4-bit binary-coded decimal (bcd) values of the time remaining. The segment mappings go from CG to CA, with CG being the most significant bit. This module contains 3 always blocks that form the skeleton of an fsm: an always block that triggers on the positive edge of clk_in to update the state, and two always blocks that trigger upon any input change to determine the next state or determine the outputs. All of the anodes and led_seg are determined in the always block that sets the outputs; flashing depends on the state and I use if

statements to check for various conditions such as meter_time being even and setting the anodes low or led_seg to a specific valX output as such. The valX are entirely determined by the submodule bcd_converter. Additionally, 3 always blocks control timing. As meter_time needs to count down every second, I used the submodule count_to_100 as a 1Hz clock and set a flag count_down to high whenever the submodule finished counting. Then I have another always block that's sensitive to all inputs, and this catches any buttons that are pushed or rstX signals. Upon detecting these inputs, it sets another variable new_meter_time to the correct sum. Lastly, I use an always block that triggers on positive edges of clk to update the meter_time by setting it to 0, adding new input, or subtracting one from itself. This was accomplished by using a series of if statements that checked when the inputs were high and a separate if statement that checked when count_down was high.

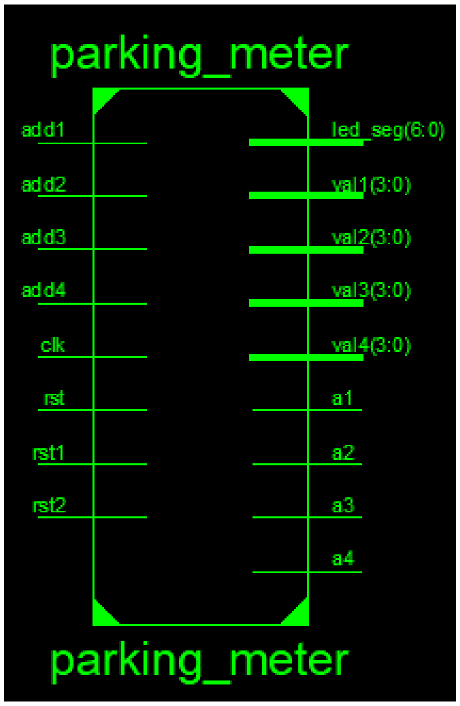


Figure 2: ISE generated schematic of the top module, parking_meter

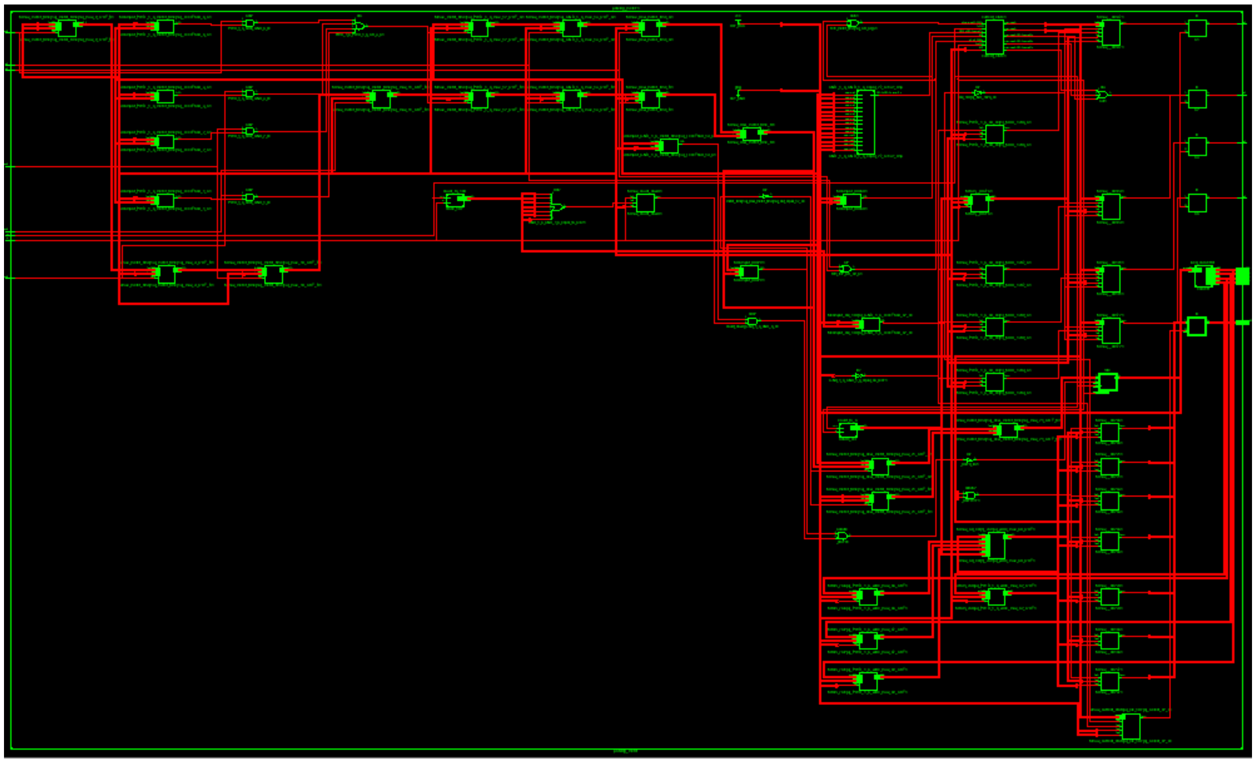


Figure 3: RTL schematic of the top module, parking_meter

2. bcd_converter: The input for this module is a 14-bit decimal number and the outputs are four 4-bit binary-coded decimals. The input needs to be 14 bits as 9999 is the largest time that can be allotted to the parking meter and there needs to be 4 outputs as 9999 has 4 digits. This module is responsible for simply converting one decimal number to four 4-bit binary-coded decimals; it achieves this through combinational logic and makes use of the division and modulus operators to extract each digit from meter_time.

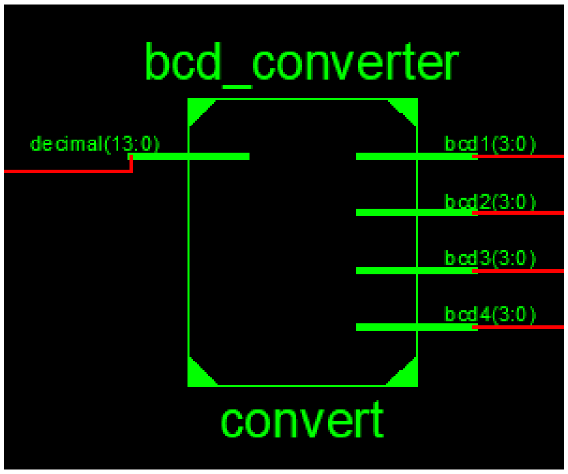


Figure 4: ISE generated schematic of the submodule, bcd_converter

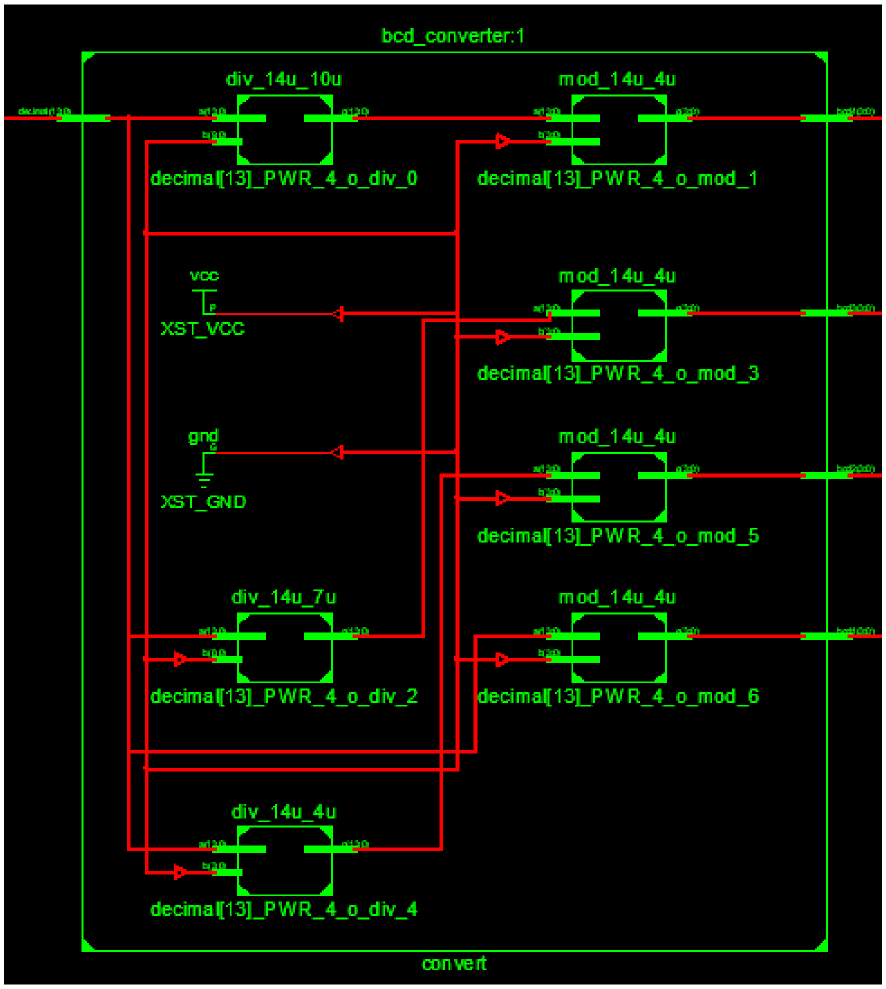

bcd_converter:1
div_14u_10u
decimal[13]_PWR_4_o_div_0
mod_14u_4u
decimal[13]_PWR_4_o_mod_1
vcc
XST_VCC
mod_14u_4u
decimal[13]_PWR_4_o_mod_3
gnd
XST_GND
mod_14u_4u
decimal[13]_PWR_4_o_mod_5
div_14u_7u
decimal[13]_PWR_4_o_div_2
mod_14u_4u
decimal[13]_PWR_4_o_mod_6
div_14u_4u
decimal[13]_PWR_4_o_div_4
convert

Figure 5: RTL schematic of the submodule, bcd_converter

3. count_to_100: Next, this module serves as a 1Hz clock. Its input is clk_in and rst from the top module, and it has a single output—a 7-bit counter. The counter will count from 0 to 99, which needs 7 binary bits. I used a single always block that is clocked to the input clk. The module has a series of if statements that reset the counter to 0 if rst is high; otherwise it checks if the counter is equal to 99 at which it resets the counter back to 0. By default, the counter adds 1 to itself.

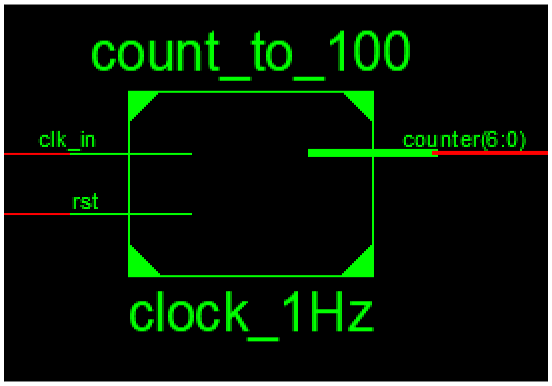


Figure 6: ISE generated schematic of the submodule, count_to_100

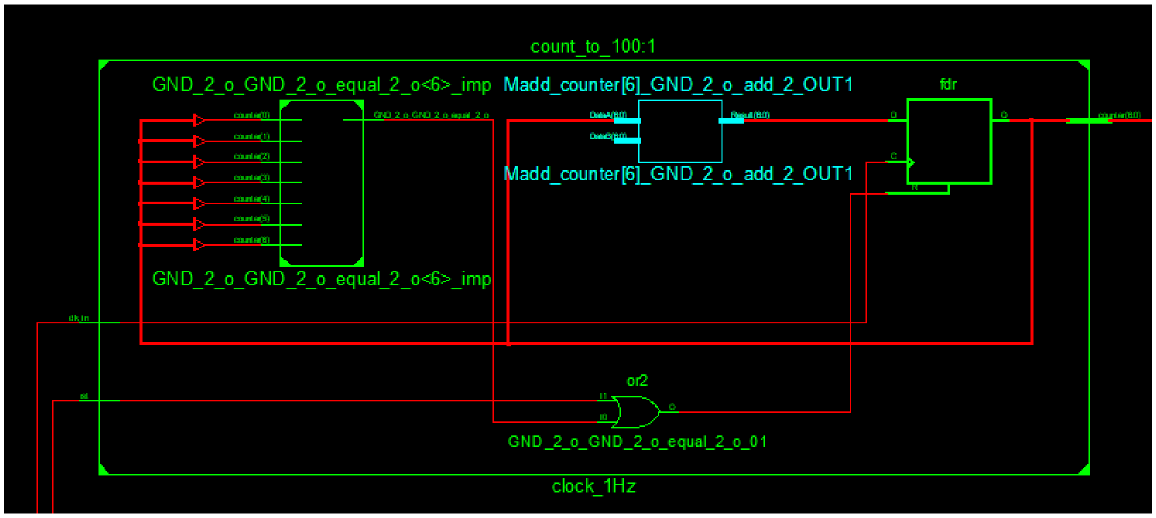

count_to_100:1
GND_2_o_GND_2_o_equal_2_o<6>_imp
Madd_counter[6]_GND_2_o_add_2_OUT1
fdr
Madd_counter[6]_GND_2_o_add_2_OUT1
GND_2_o_GND_2_o_equal_2_o<6>_imp
or2
GND_2_o_GND_2_o_equal_2_o1
clock_1Hz

Figure 7: RTL schematic of the submodule, count_to_100

4. count_to_4: Lastly, I created this module to select which anodes become low. It has the inputs clk_in and rst and outputs a 2-bit counter. The counter counts from 0 to 3. I use a single always block sensitive to all inputs in this module. Similar to the module above, there's an if statement that checks for rst to set the counter to 0. The default case is the counter adding 1 to itself. No additional check is necessary since the counter will truncate any number above 3.

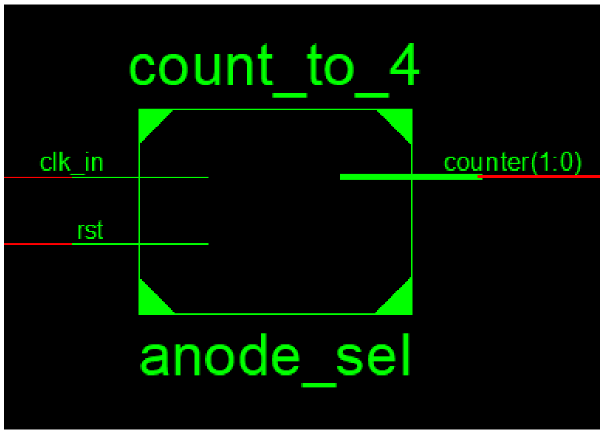


Figure 8: ISE generated schematic of the submodule, count_to_4

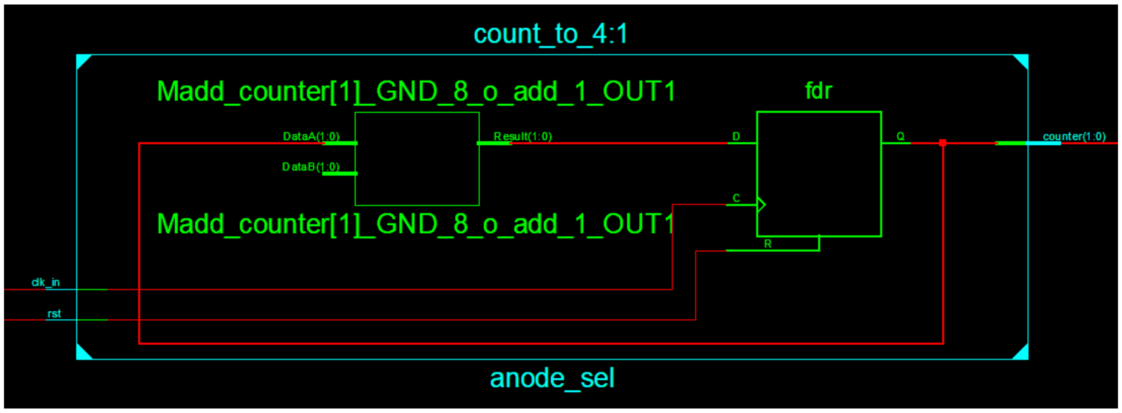


Figure 9: RTL schematic of the submodule, count_to_4

Looking at the RTL schematics, we can observe that the top module has many registers which makes sense as it needs to store many values for meter_time. There's also many MUXes for the various if statements in the always blocks. The bulk of the bcd_converter module consists of submodules that perform the mod operation as well as divide. This is due to the fact that bcd_converter doesn't store any values so it doesn't require registers. Then the two counter submodules have a module inside them that counts.

Simulation Documentation

In order to produce a functioning parking meter, I performed rigorous tests on the parking_meter module to detect and correct any bugs. I created test cases by running through all possible paths in the state diagram. Each test case is detailed below, and I outline the important signals in the waveform simulations. The testbench contains an always block to generate a 100Hz clock signal, and another always block to change the inputs. Tests are performed one after another without necessarily resetting the machine.

1. The first test shows rst initializing the machine and all the outputs. The behavior is simulated by setting rst high for 10ms before setting it low again. meter_time is 0 and thus we can observe that val4-1 are each 0 which is expected. The anodes are all low for 0.5 seconds which means that 0000 is displayed for that period of time. Then the anodes are high for the remaining 0.5 seconds. This indicates a successful flashing with a period of 1s and 50% duty cycle. Lastly led_seg is 7'b1000000 which corresponds to the digit 0. Additionally, note that the meter_time doesn't decrease in the INIT state even though a second has passed.

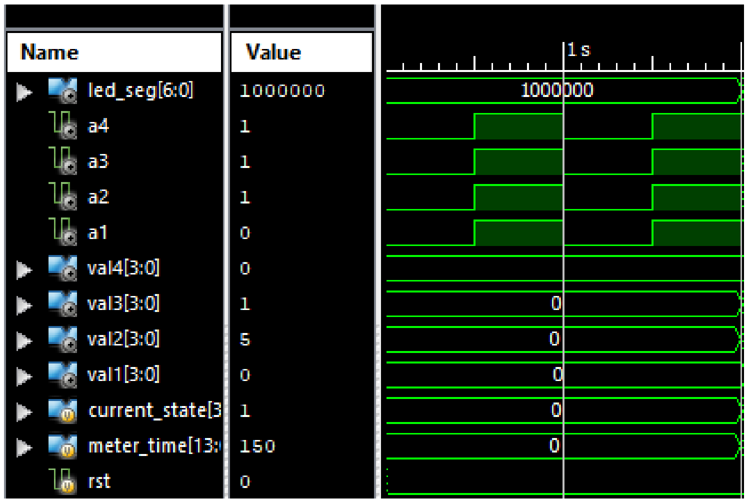


Figure 10: rst initializes parking_meter

2. This simulation focuses on the inputs and a correct summation of the inputs. I turn the buttons add1-4 on for 10ms each in cascading order. meter_time becomes 60 after add1 is pushed which is correct. Next, add2 adds 120 to 60 resulting in 180 which matches the above figure. Subsequently, add3 adds 180 to 180 (results in 360) and add4 adds 300 to 360 (results in 660), both of which agree with the waveform display. Additionally, rst1 sets meter_time to 16 and rst2 sets meter_time to 150 (the number is displayed right next to meter_time as it was cut out).
There is a small bug after add1 is pressed. There's a slight delay of 10ms between state transitions. So at 2,000ms, we see a delay of 10ms where all the anodes are low and led_seg shows a 0. This is due to the current state still being INIT at that time. However, 10ms is also too small of a time slice for the human eye to notice

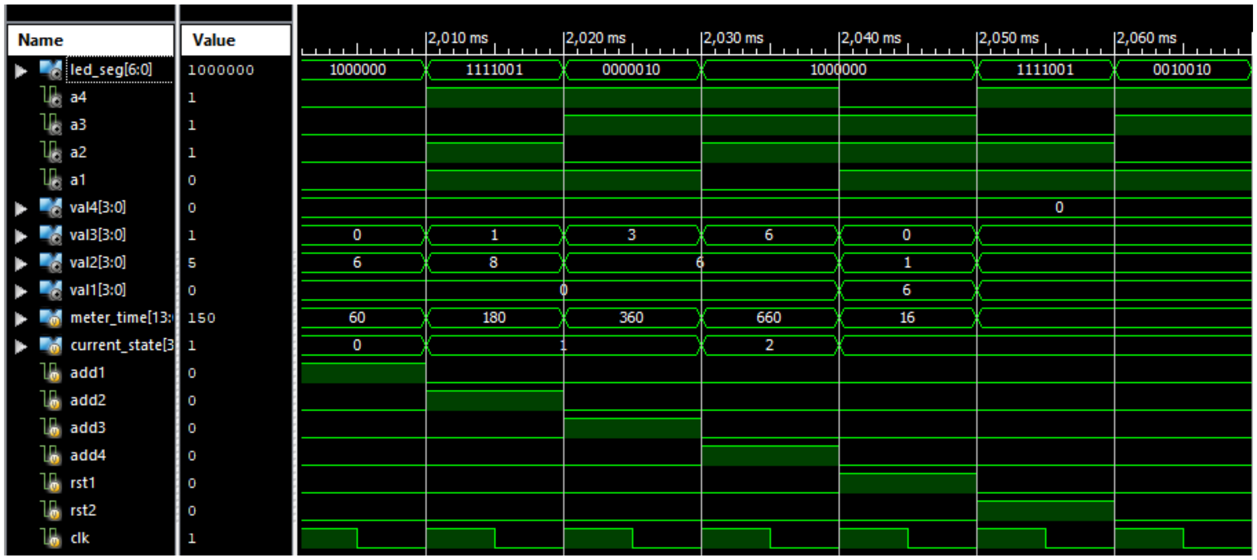


Figure 11: add1-4 and rst1-2 are pressed

3. This runthrough checks if meter_time counts down every second to represent the meter time expiring. I didn't set any inputs high and simply let the parking meter run. It's in the correct state, TIME_LESS_THAN_180, and val4-1 each show the correct digit. The time does indeed count down every second. The last piece of this waveform that I would like to highlight are the outputs: an4-1. The anodes, as we can see from the blurry green waves, alternate their low periods when meter_time is even, and are all high when meter_time is odd. Since the time flashes with a period of 2 seconds and 50% duty cycle, showing only the even values, this is correct behavior.

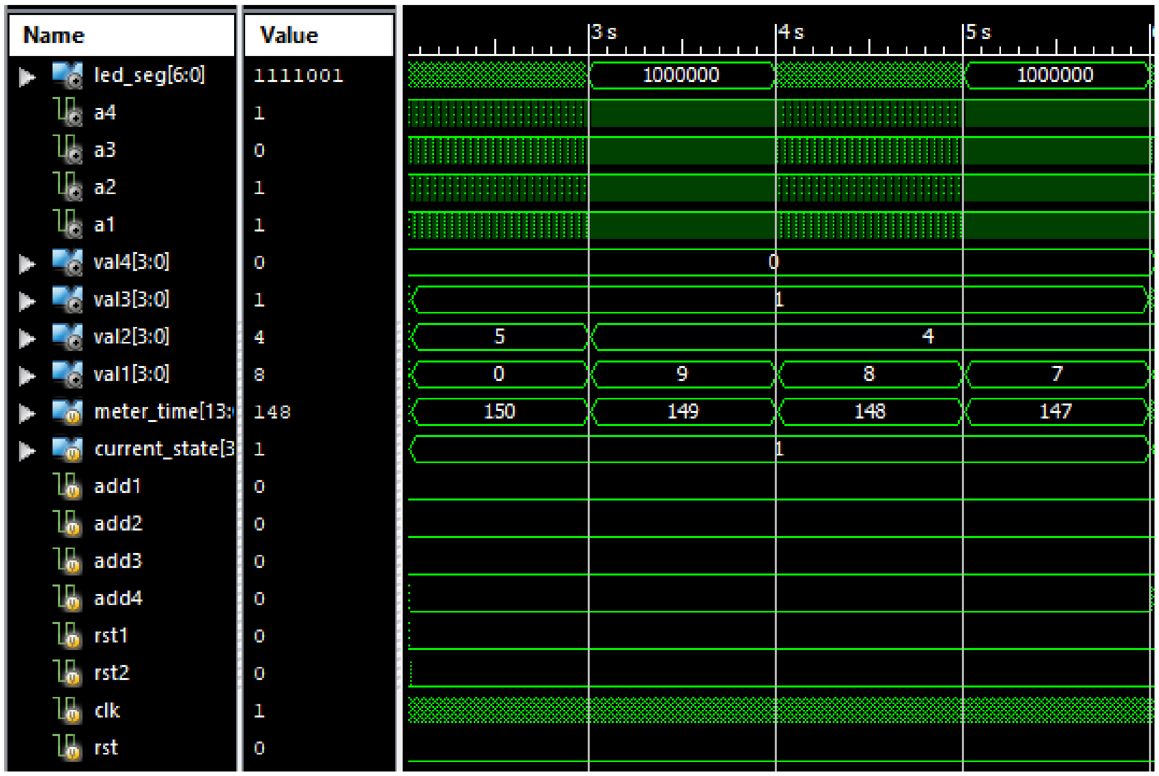
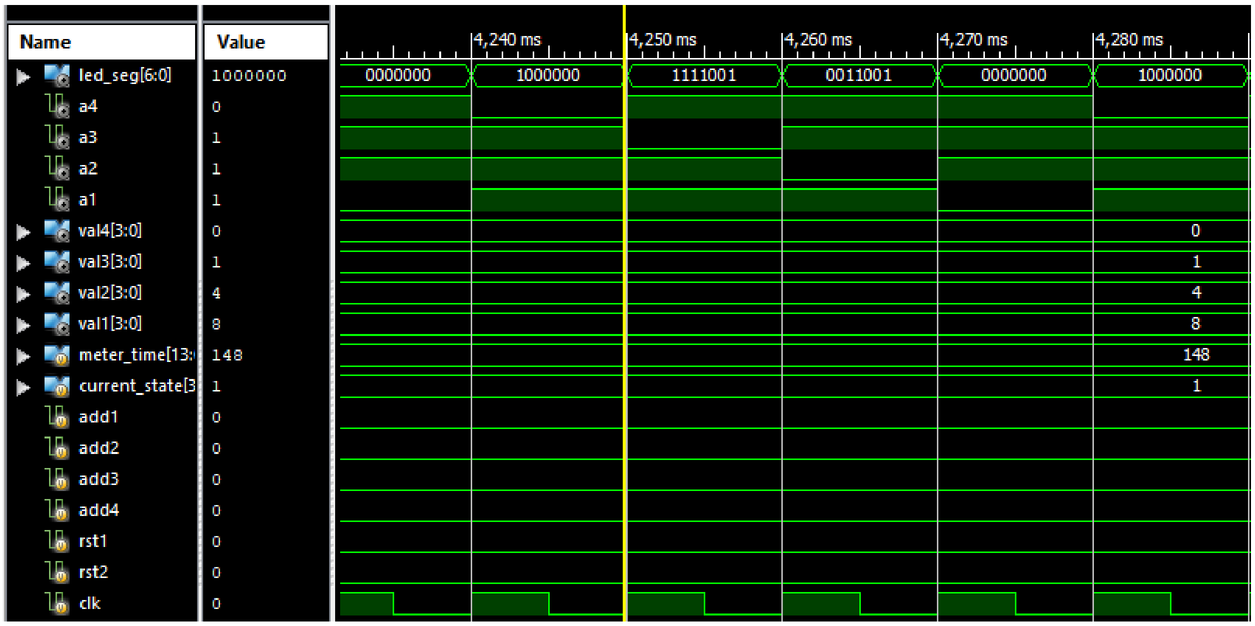


Figure 12: meter_time counts down every second; led_seg displays even seconds and flashes every second

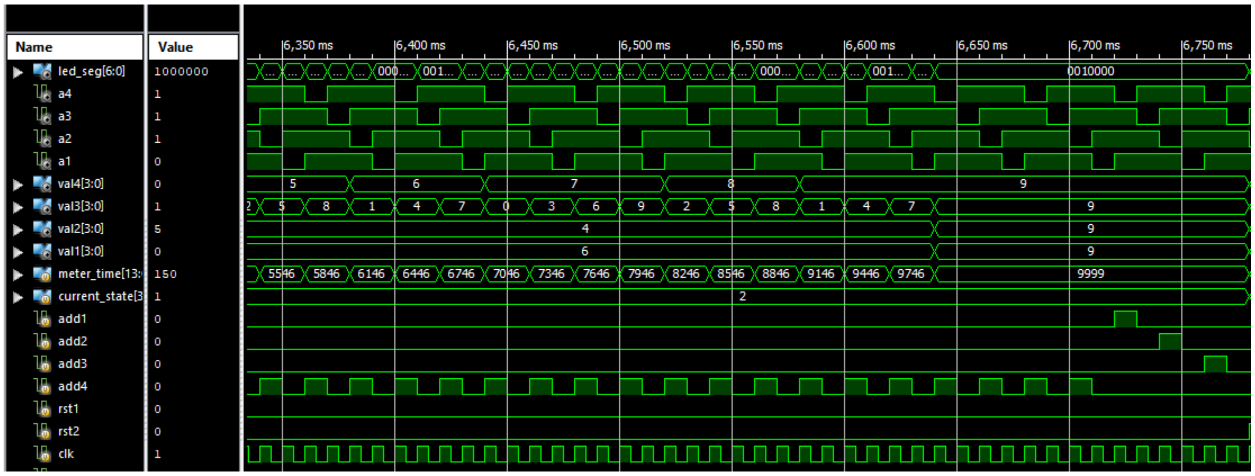
4. Next, I wanted to ensure that all 9 outputs are set correctly. In this test, meter_time is 148s, so val4-1 should be 0148 respectively. This matches with the waveform simulation below. Then focusing on the time between 4,240 and 4,280ms, the outputs an4-1 are staggered with each one low for 10ms. When an4 is low, that means that digit 4 will be displayed and so we expect the led_seg to show the sequence for a 0. This matches with what we observe below. Similarly for the other 3 anodes, led_seg shows the sequence for a 1 when an3 is low, the sequence for a 4 when an2 is low, and the sequence for an 8 when an1 is low.



Name	Value
led_seg[6:0]	1000000
a4	0
a3	1
a2	1
a1	1
val4[3:0]	0
val3[3:0]	1
val2[3:0]	4
val1[3:0]	8
meter_time[13:	148
current_state[3	1
add1	0
add2	0
add3	0
add4	0
rst1	0
rst2	0
clk	0

Figure 13: led_seg and val4-1 displays the correct binary sequence; a4-1 are staggered

5. After this, I checked if meter_time would stop incrementing when it is 9999s. In my testbench, I use a for loop to turn add4 on and off until the meter_time has incremented to 9999s. The numbers are a bit small, but we can clearly see that on the right side of the waveform, meter_time does not increment anymore despite asserting add1-4.



Name	Value
led_seg[6:0]	1000000
a4	1
a3	1
a2	1
a1	0
val4[3:0]	0
val3[3:0]	1
val2[3:0]	5
val1[3:0]	0
meter_time[13:	150
current_state[3	1
add1	0
add2	0
add3	0
add4	0
rst1	0
rst2	0
clk	1

Figure 14: meter_time is capped at 9999s despite what coins are inputted

6. This test is to make sure that rst2 will reset the machine to the TIME_LESS_THAN_180 state when the machine is originally in the TIME_MORE_THAN_180 state. I made rst2 high for 10ms and we can see that current_state transitions from 2 to 1. meter_time also changes from 9999 to 150 which is correct. I had also tried this when the original state was INIT instead of TIME_LESS_THAN_180 and the behavior was the exact same.

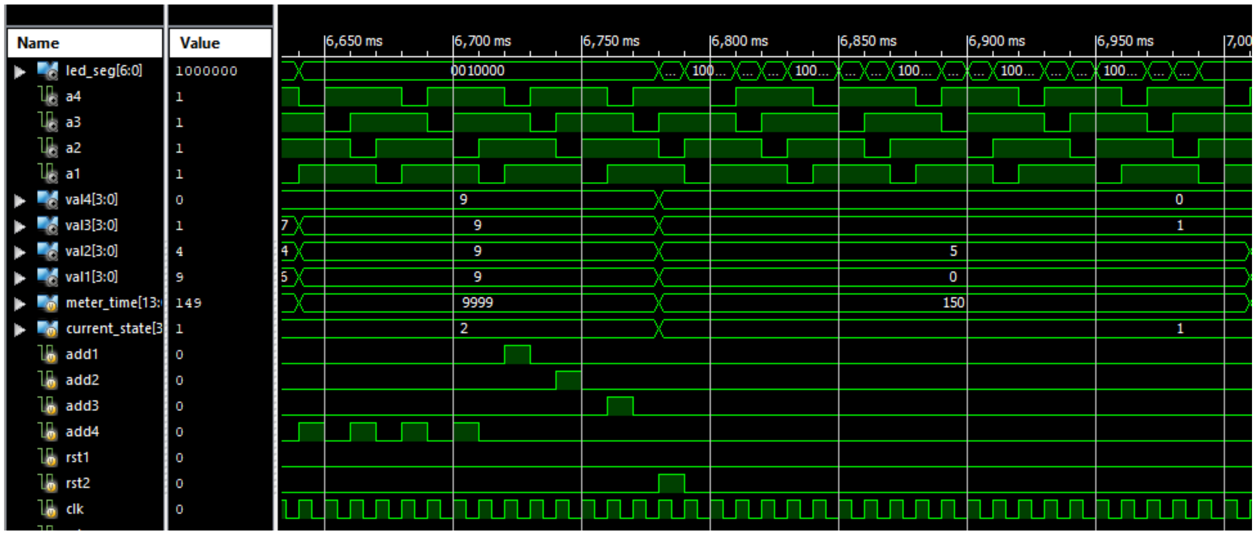


Figure 15; rst2 in state 2 will cause the machine to go to state 1

7. This following runthrough determines if the machine will work correctly when rst is high. The machine is initially in the TIME_LESS_THAN_180 state before I assert rst for 10ms. Afterwards, the current_state immediately becomes 0 (INIT state) so this works as expected. The rst signal does indeed reset the machine despite its state. Similarly, I tried asserting rst when the current_state was TIME_MORE_THAN_180 and it transitioned to the correct state of TIME_LESS_THAN_180.

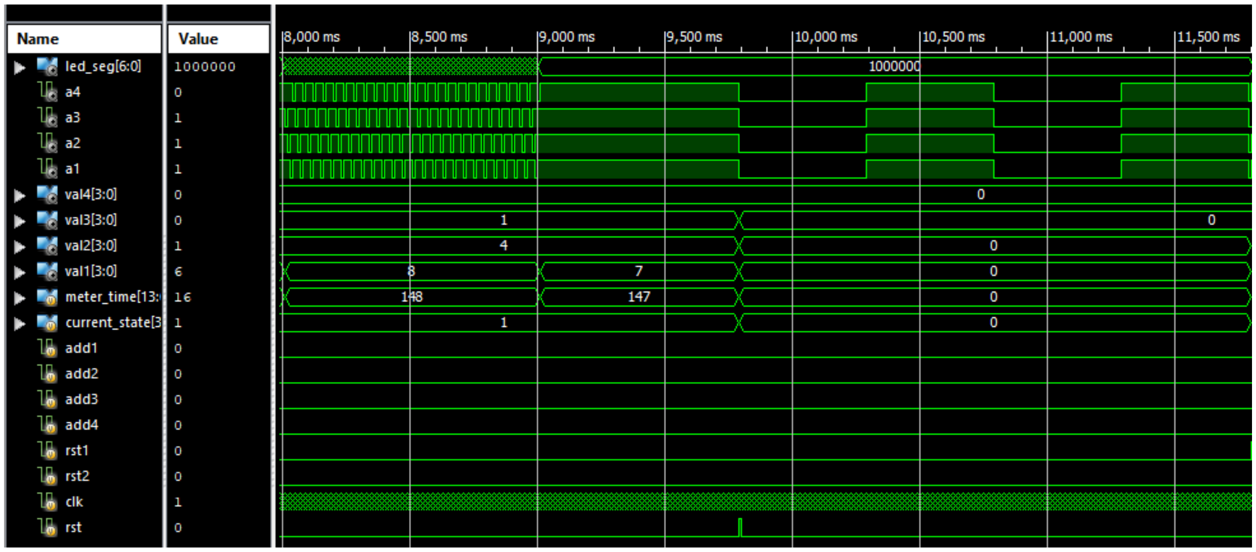


Figure 16: rst in state 1 will cause machine to transition to state 0

8. The next test case focuses on rst1. I made rst1 high for 10ms when the current_state was INIT. This immediately changed meter_time to 150 and the current_state to TIME_LESS_THAN_180 which is expected. The signal caused the machine to transition

states and the correct amount of time is displayed. Moreover, I tested the rst1 signal in the current_state of TIME_MORE_THAN_180 and the result was like this.

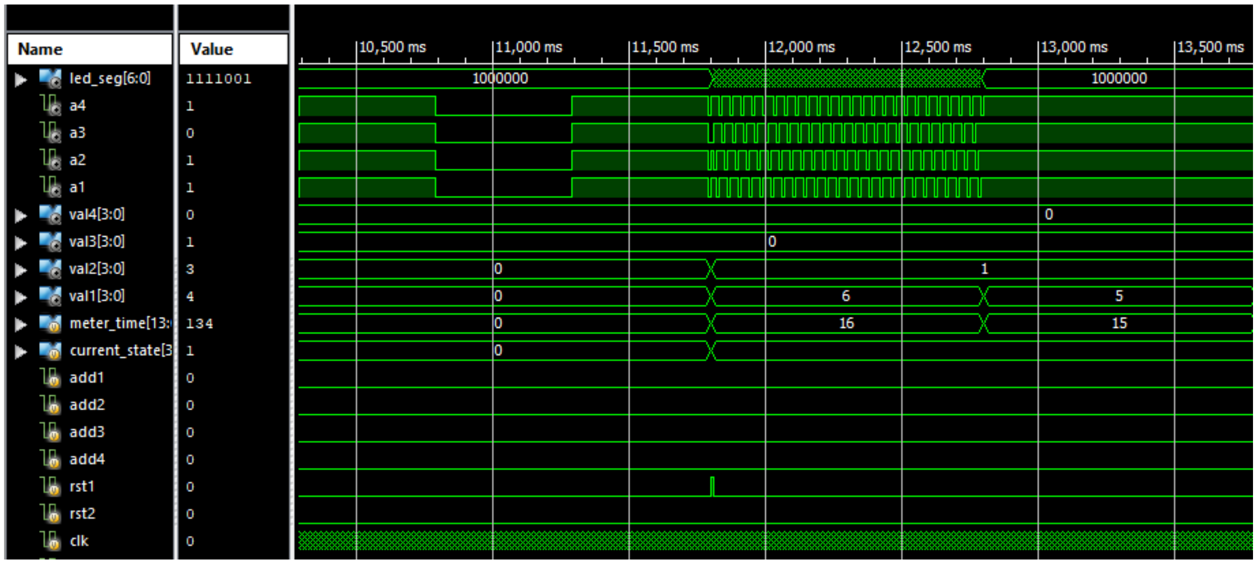


Figure 17: rst1 in state 0 will cause machine to transition to state 1

9. This test case demonstrates what happens when an input signal is high for less than 10ms. In all the previous test cases, I had kept the inputs high for one clock cycle. I wanted to make sure that the always @(*) block in the parking_meter module would still detect incoming button presses at any time, and add it to meter_time in a correct manner. Below, we can observe that add2 is asserted for 5ms instead of 10ms. 120 is added to meter_time immediately, changing it from 14 to 134s. This is correct.

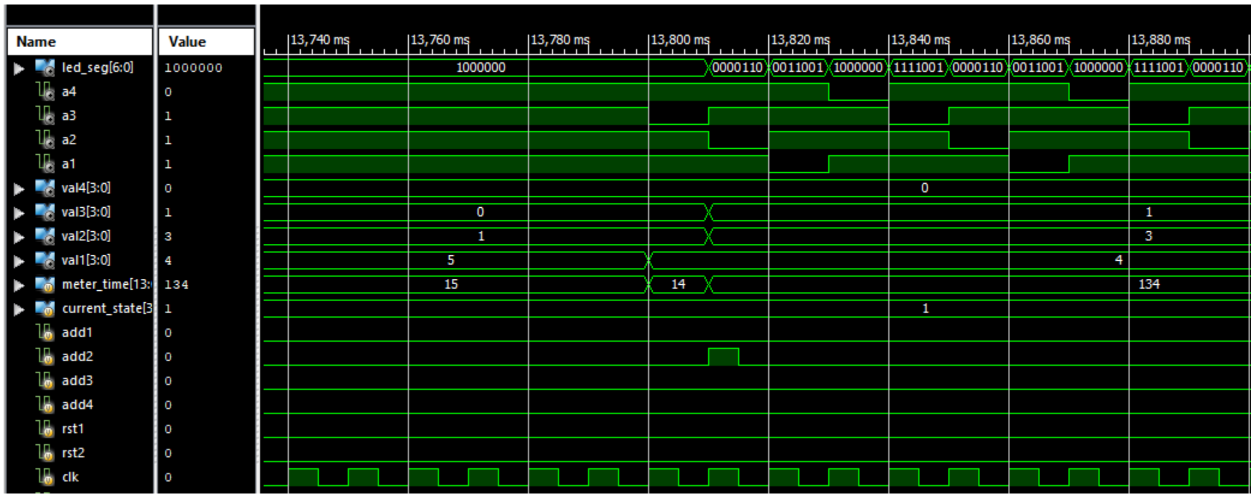


Figure 18: an input of less than a full cycle will still trigger a transition

10. Lastly, this runthrough focuses on the edge case of when a button is pressed just as the time is about to decrease. The meter_time should both add the time and also count

down. Below, we can see that add1 is asserted right as the time counts down and 60 added to 133 but meter_time also subtracts 1 resulting in 192. If add1 had not gone high, then meter_time should have been 132s but now it displays 192s. 193s would have been an incorrect display as 60s was added to 132 not 133s. But the output below is what we expect.

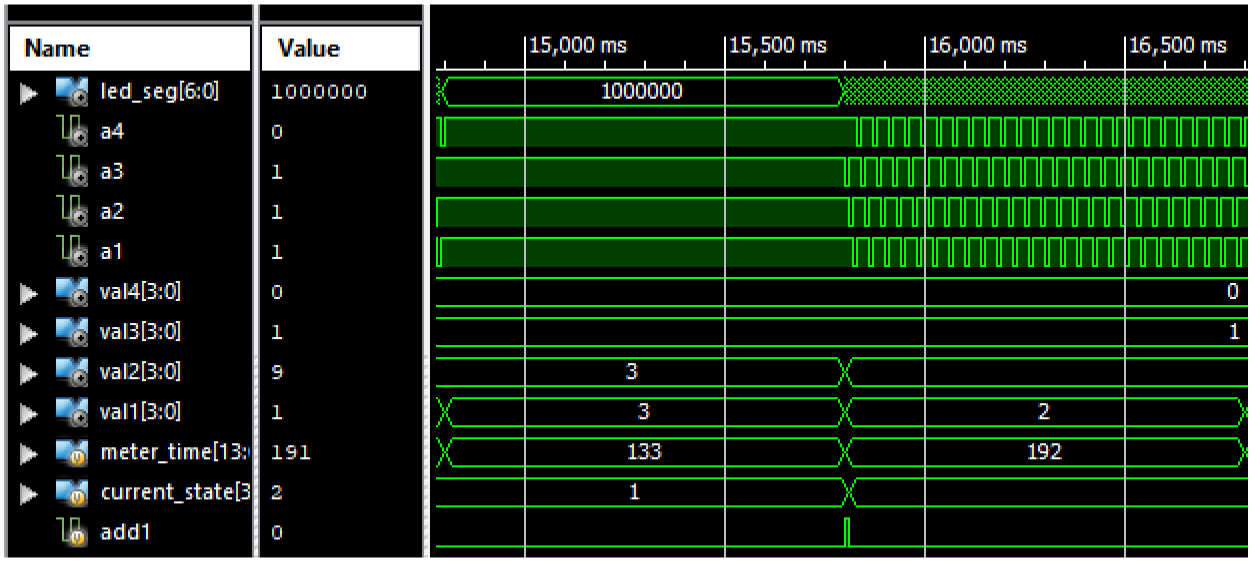
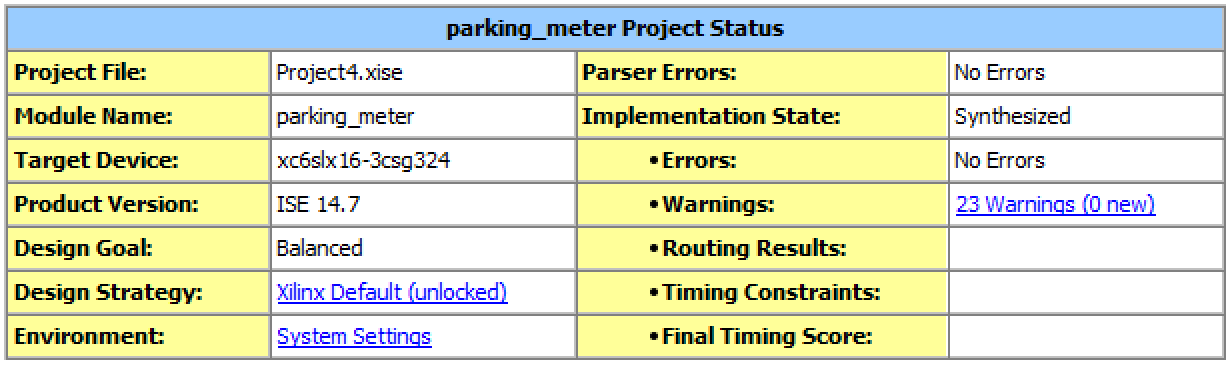
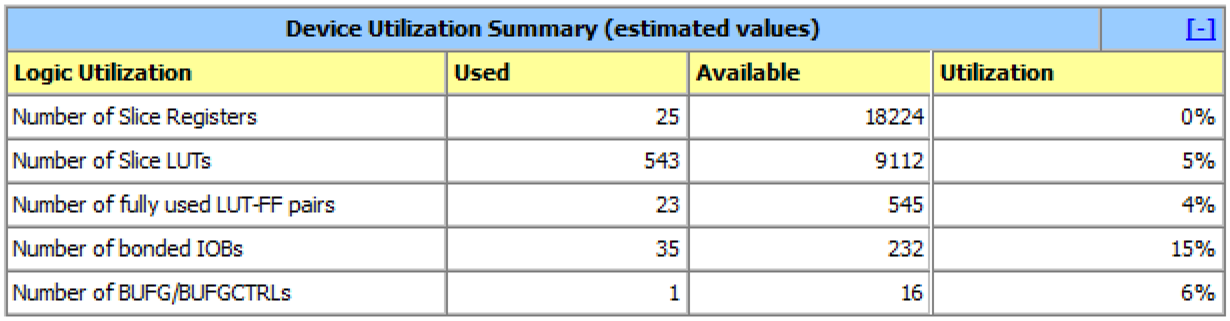


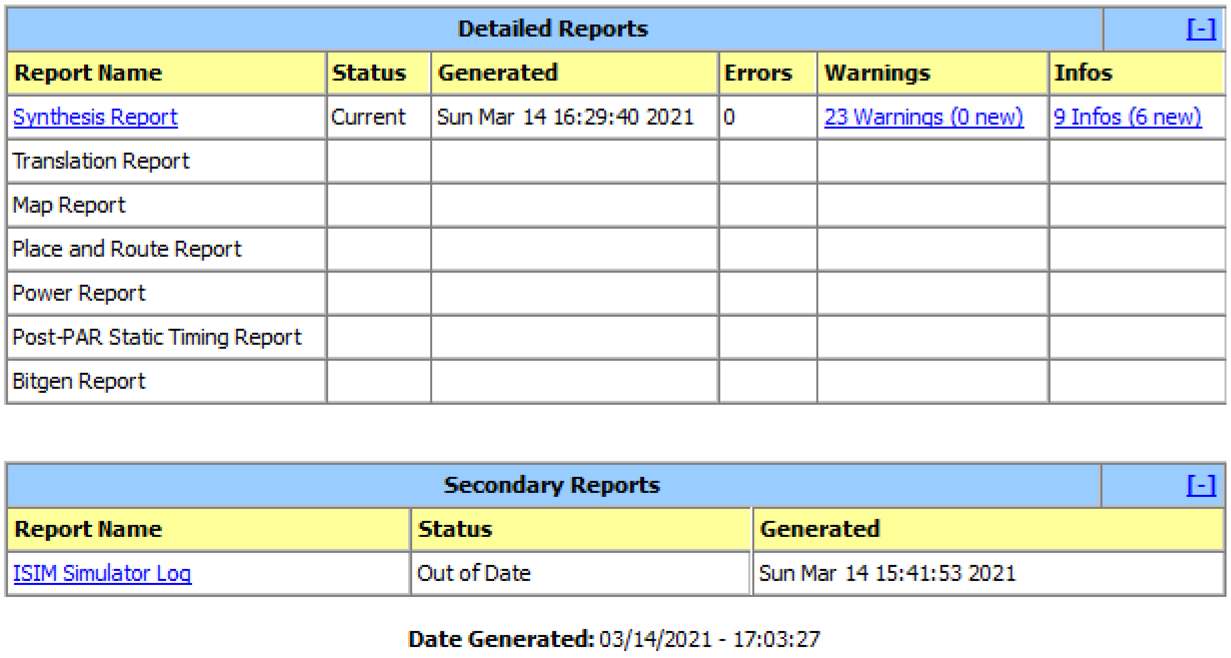
Figure 18: add1 is asserted as meter_time counts down



parking_meter Project Status			
Project File:	Project4.xise	**Parser Errors:**	No Errors
Module Name:	parking_meter	**Implementation State:**	Synthesized
Target Device:	xc6slx16-3csg324	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	23 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	25	18224	0%
Number of Slice LUTs	543	9112	5%
Number of fully used LUT-FF pairs	23	545	4%
Number of bonded IOBs	35	232	15%
Number of BUFG/BUFGCTRLs	1	16	6%



Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sun Mar 14 16:29:40 2021	0	23 Warnings (0 new)	9 Infos (6 new)
Translation Report					
Map Report					
Place and Route Report					
Power Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports		[-]
Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Sun Mar 14 15:41:53 2021

Date Generated: 03/14/2021 - 17:03:27

Figure 20: ISE Design Overview Summary Report

The summary report displays no errors and shows that my design is synthesizable. There are a few warnings about truncation and latches. In my code, I make sure to include default cases for switch statements and to leave an "else" at the end of the if statements to catch any latches. For the truncation errors, I suspect that these are due to the counters; however I ensure that the bits won't be truncated because I terminate the count_to_100 counter early. As for the count_to_4, I need it to truncate the bits.

Synthesis and Implementation Report

The summaries in the Synthesis and Map Report are attached at the end of this report. The 'Design Summary' piece of the Synthesis Report allows us to get an overview of the components that the parking meter machine utilizes. The majority of the components are registers and MUXes, and some flip-flops. I rely on switch statements and if statements to create clocks and determine flags, outputs, and states, all of which needed the former stated components. The Map Report displays no errors, but there are warnings about a gated clock and how one pin is sourced by combinatorial logic. This is most likely due to the counter that I use as a 1Hz clock. Xilinx issues these warnings as gated clocks can cause glitches or increased clock delays, which are undesirable. Originally, I used clk to generate a divide-by-100 50% duty clock. However, I wasn't able to successfully use it to control the time and the counter worked better. Since this counter is clocked by the input clk and can't be changed by anything else, it shouldn't cause glitches or clock delays.

Conclusion

In conclusion, I designed the parking meter as a Moore Machine with 3 states. To implement the machine, I used always blocks to control specific events and within each always block, I utilize case and if-else statements to determine state transitions and outputs. The most challenging part of this lab for me was figuring out how to add time to the parking meter and count down every second. I ran into issues where the time was adding double the actual amount or it wouldn't count down at all. I fixed this by using 2 always blocks to determine the new meter_time or flag that 1 second had passed. Then I used a 3rd always block triggered at the positive edge of the input clock to change meter_time based on the count_down flag and new_meter_time. In doing so, I was able to synchronize the inputs to a clock. My TA was really helpful in answering students' questions about the timing as that part of the lab description was mildly confusing. I really enjoyed this lab and found that it had the right amount of challenge. It was fun incorporating the knowledge we gained throughout the quarter.

Synthesis Report

```
=========================================================================
*                          Design Summary                               *
=========================================================================

Top Level Output File Name         : parking_meter.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 722
#      GND                         : 1
#      INV                         : 2
#      LUT1                        : 1
#      LUT2                        : 16
#      LUT3                        : 49
#      LUT4                        : 84
#      LUT5                        : 142
#      LUT6                        : 249
#      MUXCY                       : 85
#      MUXF7                       : 6
#      VCC                         : 1
#      XORCY                       : 86
# FlipFlops/Latches                : 32
#      FDR                         : 9
#      FDRE                        : 16
#      LD                          : 7
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 34
#      IBUF                        : 7
#      OBUF                        : 27
```

```
Device utilization summary:
---------------------------

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:            25  out of  18224      0%
 Number of Slice LUTs:                 543  out of   9112      5%
           Number used as Logic:              543  out of   9112      5%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:          545
   Number with an unused Flip Flop:  520  out of      545      95%
   Number with an unused LUT:         2  out of         545      0%
   Number of fully used LUT-FF pairs:          23  out of        545      4%
   Number of unique control sets:       4

IO Utilization:
 Number of IOs:                        35
 Number of bonded IOBs:                35  out of       232      15%
           IOB Flip Flops/Latches:                 7

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                    1  out of          16      6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------
```

Map Report

```
Design Summary
--------------
Number of errors:       0
Number of warnings:  1
Slice Logic Utilization:
  Number of Slice Registers:                   25 out of  18,224        1%
           Number used as Flip Flops:                25
```

```
        Number used as Latches:                 0
        Number used as Latch-thrus:             0
        Number used as AND/OR logics:           0
  Number of Slice LUTs:                474 out of   9,112       5%
        Number used as logic:                474 out of   9,112       5%
        Number using O6 output only:         351
        Number using O5 output only:         1
        Number using O5 and O6:              122
        Number used as ROM:                  0
        Number used as Memory:                  0 out of   2,176         0%

Slice Logic Distribution:
  Number of occupied Slices:                 177 out of   2,278       7%
  Number of MUXCYs used:                     96 out of   4,556        2%
  Number of LUT Flip Flop pairs used:        474
        Number with an unused Flip Flop:        451 out of       474   95%
        Number with an unused LUT:              0 out of         474     0%
        Number of fully used LUT-FF pairs:      23 out of        474     4%
        Number of unique control sets:          3
        Number of slice register sites lost
        to control set restrictions:         7 out of  18,224         1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
  one Flip Flop within a slice.  A control set is a unique combination of
  clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

IO Utilization:
  Number of bonded IOBs:                     35 out of        232   15%
        IOB Latches:                         7

Specific Feature Utilization:
  Number of RAMB16BWERs:                     0 out of          32       0%
  Number of RAMB8BWERs:                      0 out of          64       0%
  Number of BUFIO2/BUFIO2_2CLKs:                0 out of          32       0%
  Number of BUFIO2FB/BUFIO2FB_2CLKs:            0 out of          32       0%
  Number of BUFG/BUFGMUXs:                   1 out of          16       6%
        Number used as BUFGs:                   1
        Number used as BUFGMUX:                 0
  Number of DCM/DCM_CLKGENs:                    0 out of          4        0%
  Number of ILOGIC2/ISERDES2s:               0 out of          248      0%
  Number of IODELAY2/IODRP2/IODRP2_MCBs:               0 out of          248      0%
  Number of OLOGIC2/OSERDES2s:                  7 out of          248      2%
```

```
          Number used as OLOGIC2s:                    7
          Number used as OSERDES2s:                   0
  Number of BSCANs:                           0 out of         4        0%
  Number of BUFHs:                            0 out of         128      0%
  Number of BUFPLLs:                          0 out of         8        0%
  Number of BUFPLL_MCBs:                      0 out of         4        0%
  Number of DSP48A1s:                         0 out of         32       0%
  Number of ICAPs:                    0 out of         1        0%
  Number of MCBs:                     0 out of         2        0%
  Number of PCILOGICSEs:                      0 out of         2        0%
  Number of PLL_ADVs:                         0 out of         2        0%
  Number of PMVs:                     0 out of         1        0%
  Number of STARTUPs:                         0 out of         1        0%
  Number of SUSPEND_SYNCs:                            0 out of         1        0%

Average Fanout of Non-Clock Nets:             4.31

Peak Memory Usage:  4529 MB
Total REAL time to MAP completion:  18 secs
Total CPU time to MAP completion:   12 secs

Table of Contents
-----------------
Section 1 - Errors
Section 2 - Warnings
Section 3 - Informational
Section 4 - Removed Logic Summary
Section 5 - Removed Logic
Section 6 - IOB Properties
Section 7 - RPMs
Section 8 - Guide Report
Section 9 - Area Group and Partition Summary
Section 10 - Timing Report
Section 11 - Configuration String Information
Section 12 - Control Set Information
Section 13 - Utilization by Hierarchy

Section 1 - Errors
------------------

Section 2 - Warnings
--------------------
WARNING:PhysDesignRules:372 - Gated clock. Clock net
   current_state[3]_clk_1Hz[6]_Select_82_o is sourced by a combinatorial pin.
   This is not good design practice. Use the CE pin to control the loading of
   data into the flip-flop.

Section 3 - Informational
------------------------
INFO:MapLib:562 - No environment variables are currently set.
INFO:LIT:244 - All of the single ended outputs in this design are using slew
   rate limited output drivers. The delay on speed critical single ended outputs
   can be dramatically reduced by designating them as fast outputs.
INFO:Pack:1716 - Initializing temperature to 85.000 Celsius. (default - Range:
   0.000 to 85.000 Celsius)
INFO:Pack:1720 - Initializing voltage to 1.140 Volts. (default - Range: 1.140 to
   1.260 Volts)
INFO:Map:215 - The Interim Design Summary has been generated in the MAP Report
   (.mrp).
INFO:Pack:1650 - Map created a placed design.

Section 4 - Removed Logic Summary
---------------------------------
   2 block(s) optimized away

Section 5 - Removed Logic
------------------------

Optimized Block(s):
TYPE              BLOCK
GND               XST_GND
VCC     XST_VCC

To enable printing of redundant blocks removed and signals merged, set the
detailed map report option and rerun map.

Section 6 - IOB Properties
--------------------------

+---------------------------------------------------------------------------------------------------------------------------+
| IOB Name                           | Type            | Direction | IO Standard         | Diff  | Drive    | Slew |
Reg (s)          | Resistor | IOB          |
|                              |                  |          |                      | Term  | Strength | Rate |
|          | Delay |
+---------------------------------------------------------------------------------------------------------------------------+
| a1                         | IOB              | OUTPUT       | LVCMOS25             |          | 12      |
SLOW |          |          |          |
| a2                 | IOB          | OUTPUT     | LVCMOS25         |        | 12     |
SLOW |        |        |        |
| a3                 | IOB          | OUTPUT     | LVCMOS25         |        | 12     |
SLOW |        |        |        |
| a4                 | IOB          | OUTPUT     | LVCMOS25         |        | 12     |
SLOW |        |        |        |
| add1                        | IOB          | INPUT      | LVCMOS25         |        |
|        |        |        |        |
| add2                        | IOB          | INPUT      | LVCMOS25         |        |
|        |        |        |        |
| add3                        | IOB          | INPUT      | LVCMOS25         |        |
|        |        |        |        |
| add4                        | IOB          | INPUT      | LVCMOS25         |        |
|        |        |        |        |
| clk                | IOB          | INPUT      | LVCMOS25         |        |        |
|        |        |        |
| led_seg<0>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| led_seg<1>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| led_seg<2>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| led_seg<3>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| led_seg<4>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| led_seg<5>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| led_seg<6>                  | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW | OLATCH    |        |        |
| rst                | IOB          | INPUT      | LVCMOS25         |        |        |
|        |        |        |
| rst1               | IOB          | INPUT      | LVCMOS25         |        |        |
|        |        |        |
| rst2               | IOB          | INPUT      | LVCMOS25         |        |        |
|        |        |        |
| val1<0>                     | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW |           |        |        |
| val1<1>                     | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW |           |        |        |
| val1<2>                     | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW |           |        |        |
| val1<3>                     | IOB          | OUTPUT     | LVCMOS25         |        | 12
| SLOW |           |        |        |
| val2<0>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val2<1>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val2<2>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val2<3>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val3<0>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val3<1>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val3<2>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val3<3>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val4<0>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val4<1>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val4<2>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
| val4<3>                        | IOB            | OUTPUT       | LVCMOS25          |          | 12
| SLOW |                 |          |          |
+-------------------------------------------------------------------------------------------------------------------------+

Section 7 - RPMs
----------------

Section 8 - Guide Report
------------------------
Guide not run on this design.

Section 9 - Area Group and Partition Summary
--------------------------------------------

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

Area Group Information
```

```
---------------------

  No area groups were found in this design.

---------------------

Section 10 - Timing Report
--------------------------
A logic-level (pre-route) timing report can be generated by using Xilinx static
timing analysis tools, Timing Analyzer (GUI) or TRCE (command line), with the
mapped NCD and PCF files. Please note that this timing report will be generated
using estimated delay information. For accurate numbers, please generate a
timing report with the post Place and Route NCD file.

For more information about the Timing Analyzer, consult the Xilinx Timing
Analyzer Reference Manual; for more information about TRCE, consult the Xilinx
Command Line Tools User Guide "TRACE" chapter.

Section 11 - Configuration String Details
-----------------------------------------
Use the "-detail" map option to print out Configuration Strings

Section 12 - Control Set Information
------------------------------------
Use the "-detail" map option to print out Control Set Information.

Section 13 - Utilization by Hierarchy
-------------------------------------
Use the "-detail" map option to print out the Utilization by Hierarchy section.
```