Lab 3: Design a parking meter

CS M152A Fall 2020, Majid Sarrafzadeh

Lab 6: Rajas Mhaskar

Kenta Hayakawa

11/29/2020

Introduction and Requirement

The purpose of this lab was to design a Finite State Machine to model a parking meter which simulates coins being added and displays the appropriate time remaining. FSMs are a powerful tool that can be used to model many real world systems, and are particularly useful for the behavioral modeling of sequential circuits. An FSM can have a finite number of states, and can be in any one of these states at any given time. The machine transitions from one state to another based on the inputs it receives and the state that it is currently in. The machine must begin operation in an initial state. Below are the possible inputs to this parking meter, and the expected outputs:

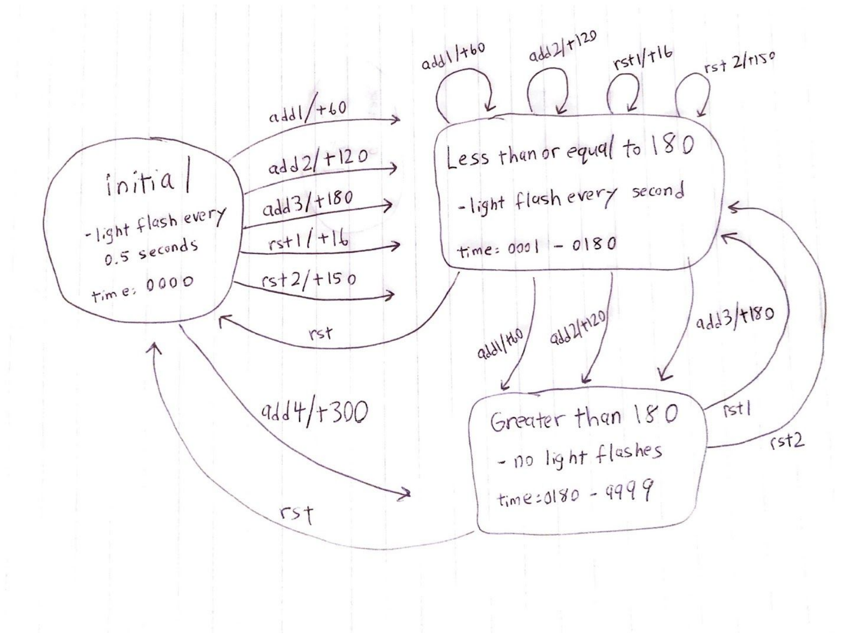
//table

As soon as a button is pushed, the time should be added immediately. The output is modeled as 4 seven segment displays which display the time remaining, and each add and reset button is high for at most one clock cycle.

In the initial state, and anytime the meter reads 0 seconds left, the led segments will flash on and off with a period of 1 sec, meaning 0.5 sec the led will be on and 0.5 sec the led will be off. If the time is 180 sec to 1 seconds, the led will flash for 1 sec and disappear for 1 sec, so the even numbers will be on and the odd numbers will be off. The max time value the meter can have is 9999 seconds. It is assumed that multiple inputs will not be pressed at the same time

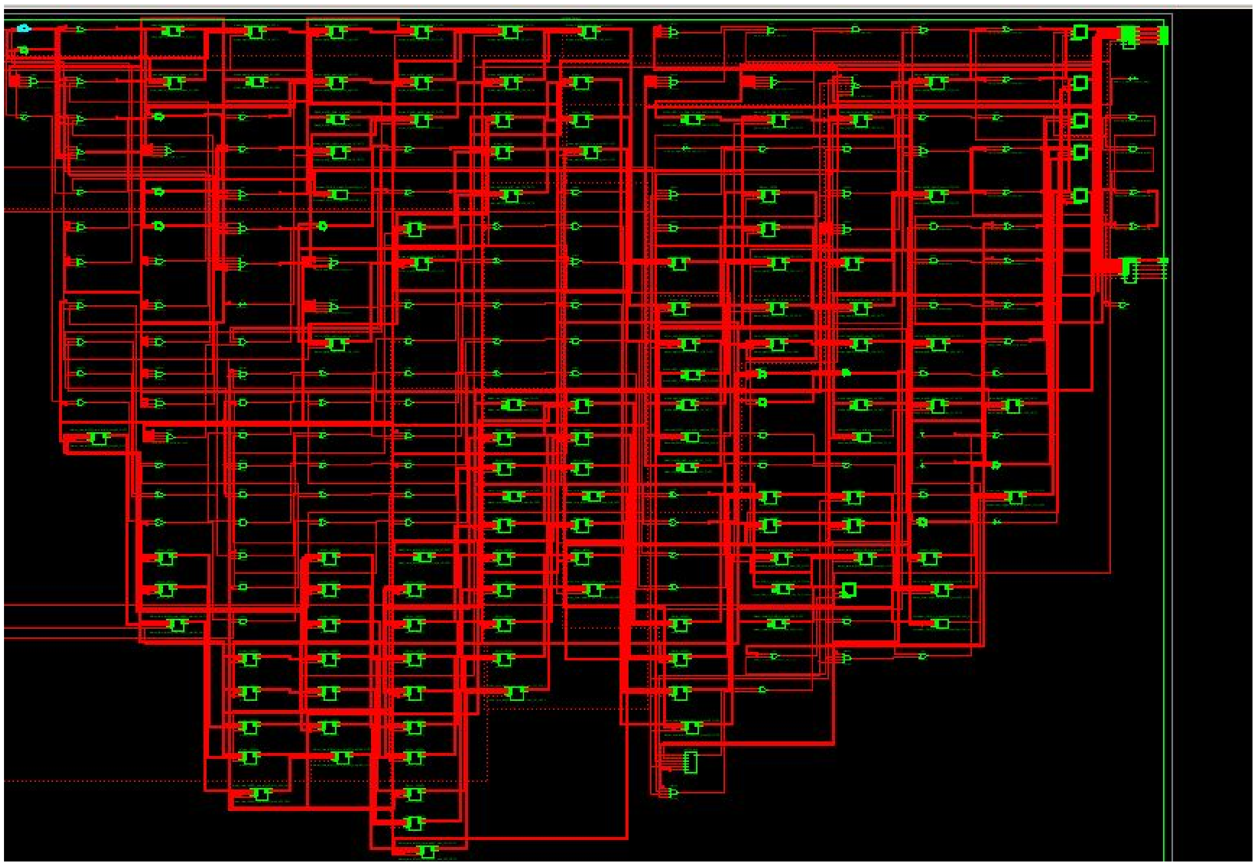
Design Description

The finite state machine for this design is given below:



The states were categorized mainly into three states: the initial state, the state less than or equal to 180 seconds, and the states representing times beyond 180. Depending on the combination of inputs, the remaining time will be in one of these three states at all times, and it also represents the different behaviors of flashing as well.

The schematic for this design is shown below:



The schematic is very complex, and this shows that my implementation was not very modular, since I only used two submodules, and they can be seen at the very right of the schematic (very small, lime green color). If I had more modules the schematic would look more organized. The inputs come in from the left side of the schematic, and most of the calculations are done here (hence the complicated circuit). Then, the outputs are sent to the submodules, which output the final outputs.

The code was divided into three modules:

1) The top module puts every module together, and includes the main logic for the calculation of the remaining times given an input and the automatic

decrement of time every second. The basic logic of the decrement is that a "round" variable was used to count every edge of the clock, and at a certain value of "round", the time decreased. The effect of the input was also triggered at certain values of this variable as well.

2) The sevenseg module has two main functions: using the remaining time as input and generating an LED segment value using those digits, and to trigger the anodes based on the input clock. For the cases less than 180, a special counting variable was used to determine when to turn the led segment off (1111111), and in this case it was every second. Using a similar technique, the initial state was coded so that a counting variable would aid in detecting half a second, so the led segment would flash. For cases greater than 180, no flashing algorithm was used, and the led segment simply displayed the output as is.
3) The final module was the BCD encoder, which takes in the remaining time and generates 4 values that represent the actually decimal values shown on the display of the parking meter. Since I calculated these decimal values in the top module (by accident), I simply used those digits calculated in the top module as the inputs to the module, and outputted those digits exactly out. The output is val1, val2, val3, and val4, each with a binary equivalent of the decimal output of the meter.

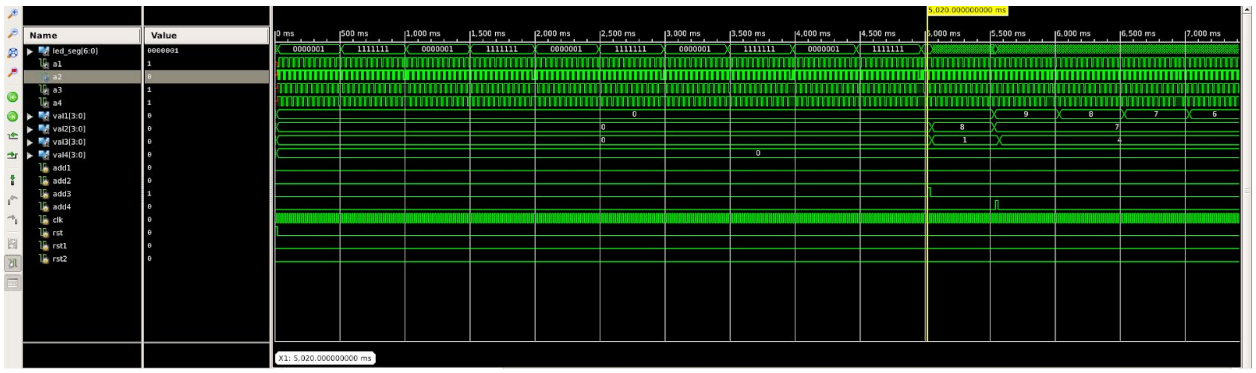
Simulation Documentation

The testbench for this project included all 10 possible inputs: add1, add2, add3, add4, rst1, rst2, rst, and clk. I made my time unit 1ms for convenience, and a 100hz clock was generated by giving a 10 unit time delay every clock pulse. The reset signal created the initial state with all 0s as the led outputs, and since this goes on for the first five seconds, the 0s flash off every 0.5 seconds, and flash on the other 0.5 seconds. I used a fairly random sequence of inputs: add3 was inputted first after the initial 5 seconds, followed by an add4 a half second later. A rst1 input is then inputted 2 seconds later, taking the clock all the way down to 16 seconds left. An add1 input is given a second later, and a rst2 input another half second later. The state transitions during this stage are very quick. Finally, an add2 input is given a second later, followed by a reset button another second later that takes the remaining time back to 0. Every input signal is only high for one clock cycle, or 20ms.

Throughout this simulation, whenever the time dipped below 180 seconds, the led segments flash every second, meaning the odd values below 180 are disappeared on the screen. This can be seen with the odd values having a led segment value of 1111111, representing an off state. The same led segment value is seen every half second when the remaining time is 0.

The a1, a2, a3, and a4 values give a waveform that is almost constant throughout the entire simulation (meaning that the meter is constantly outputting an LED output. The lights are basically flashing in order extremely fast, so that it looks like the led segments are being outputted simultaneously). The input clock of 100hz was used to time the anode clockings. The val1, val2, val3, and val4 are the BCD encodings representing the decimal value of the output time. In this simulation documentation, the BCD encodings are shown in unsigned decimal form for convenience (by switching the radix setting in the simulation page), but these are in binary form by default.

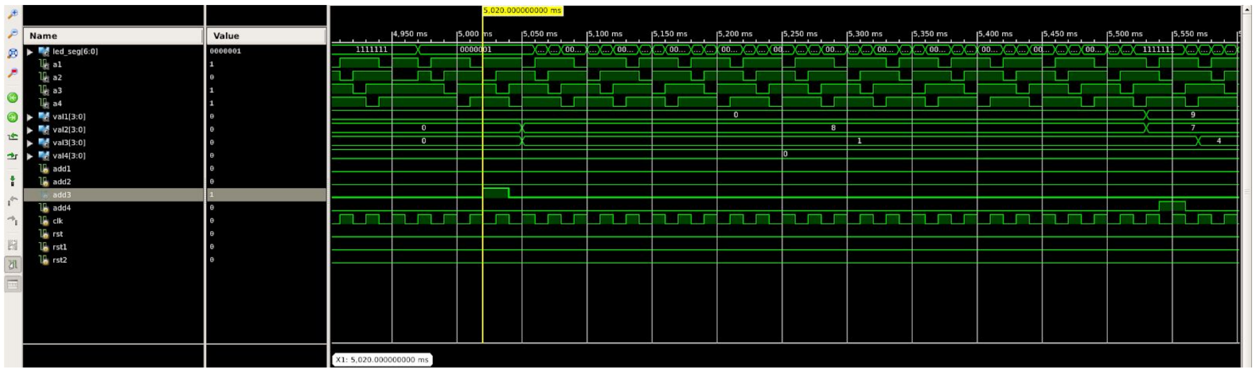
The output below shows the initial state. A reset is triggered, and the anode signals start taking in inputs. As seen here, each anode period is 10ms, since the input clock is 100hz. So this meter has an anode refresh period of 40ms. Every 500ms, or half a second, the led segments switch to 1111111, signifying that the led is flashing off, then turns back to 0000001, signifying that the led is flashing on. The val values are all 0 as expected, so the meter is showing the value 0000.



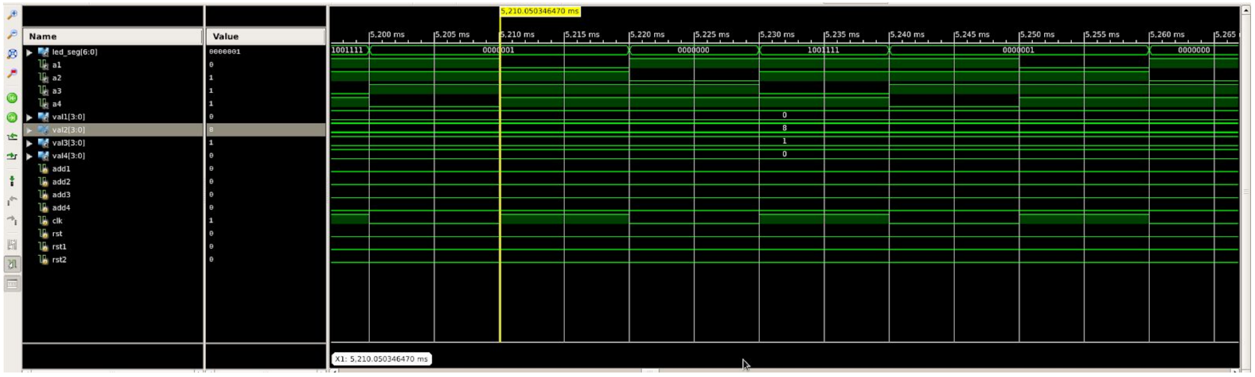
Name	Value
led_seg[6:0]	0000001
a1	1
a2	0
a3	1
a4	1
val1[3:0]	0
val2[3:0]	0
val3[3:0]	0
val4[3:0]	0
add1	0
add2	0
add3	1
add4	0
clk	0
rst	0
rst1	0
rst2	0

X1: 5,020.000000000 ms

The first output below shows the next state, when the add3 input is given. Since the reset went to 0 at 20ms, the add3 input is triggered not exactly at 5000ms, but at 5020 ms. The val displays the numbers that will be displayed on the meter, which is 0180. The led segments all read 180 in 7 bit led segment form, as shown in the second output below.

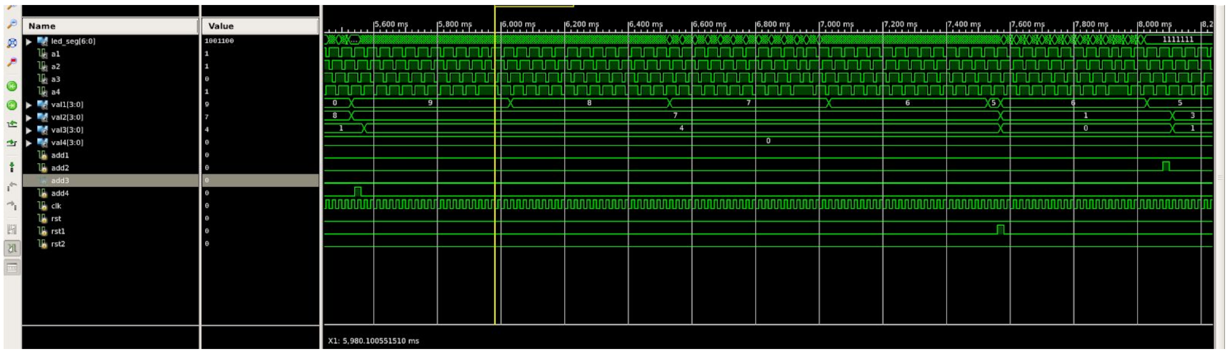


-add3 is inputted

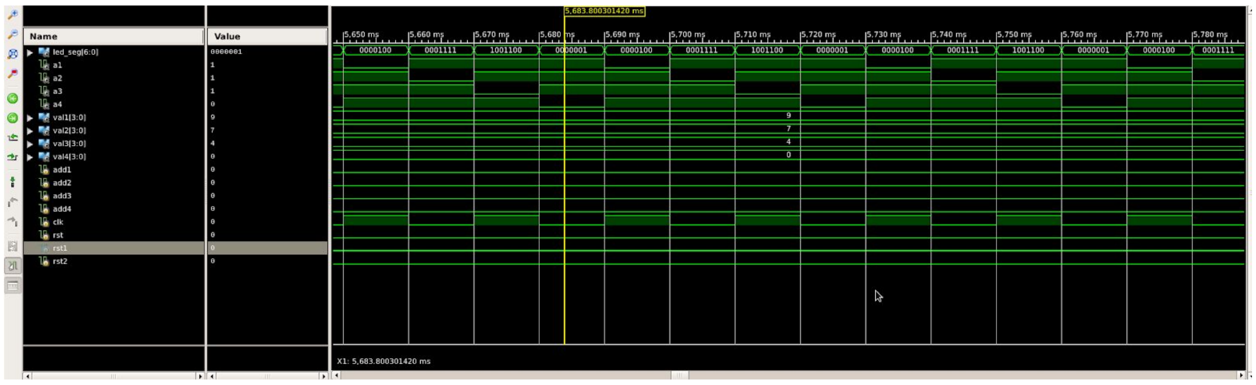


-lights are flashing

The first output below shows the next state, when add4 is inputted. Since the last time that was outputted on the meter briefly before switching states was 179, add4 adds 300 to that time, so the meter now outputs 479, or 0479 as the output. From there, it counts down until the next input, so the time goes from 0479 down to a brief moment of 0475. Since the time is not below 180, the lights will not flash. The second output below shows how the led segments will not show 1111111.

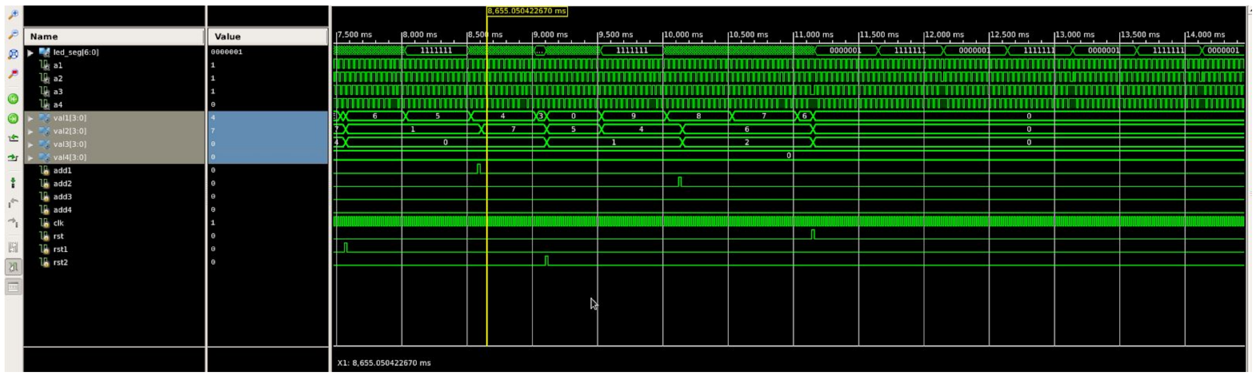


-add4 is inputted



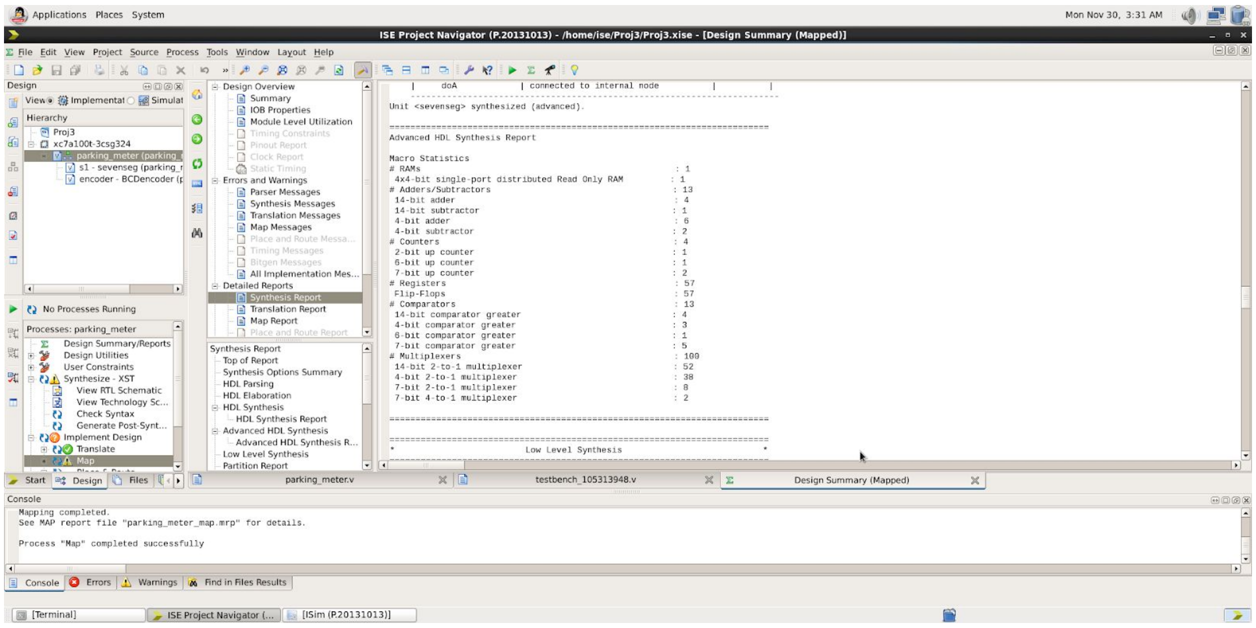
-lights are not flashing

The first output below shows the next few states, since they are all inputted in quick succession. To the far left, a rst1 input turns the meter all the way down to 16 seconds left, and almost a half second later another input add1 is given. Since the previous time on the meter was a brief moment of 14 seconds, the add1 signal adds 60 seconds to that, giving an output of 74 seconds. Since both states are less than 180 seconds, it flashes every second, as seen in the led segments (every other state is 1111111, which is off). The rst2 button then resets the time to 150, and starts counting down. Similar to previous states, the time is less than 180 seconds, so it is flashing every second. After 2 seconds, an add2 input is given which adds 120 seconds to the current time, making the remaining time up to 268 seconds (since the previous state ended at 148 seconds). It counts down, and since it is above 180 seconds, no blinking occurs. Finally, a reset signal brings the time down to 0 seconds. Similar to the initial state, the led segments blink every 0.5 seconds. Since no input is given afterwards, it maintains that 0000 state.



-final few outputs until end

The following is the design summary given from the synthesis report. It gives how many of each digital component was utilized for the synthesis of the design. From this, it can be seen that the most used type of component was the multiplexer, with around 100, which makes sense because a lot of multiplexing went on, especially during the encodings. From the map report, it can be seen that 0 errors were made, which is important, and a lot of sliced LUTs, or look-up tables, were used. This makes sense because a lot of repeated values were used during calculations, so it is more convenient to look those values up when needed, to save resources.

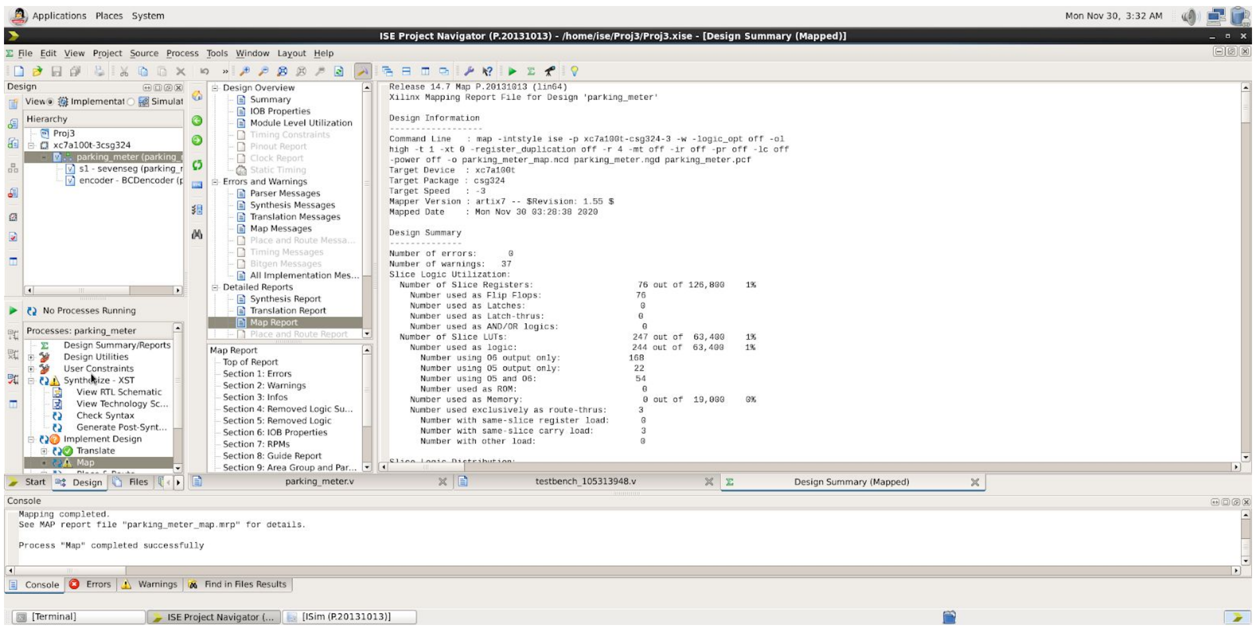


ISE Project Navigator (P.20131013) - /home/ise/Proj3/Proj3.xise - [Design Summary (Mapped)]
Mon Nov 30, 3:31 AM

```
    |      doA            | connected to internal node          |           |
    -----------------------------------------------------------------------------
Unit <sevenseg> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                  : 1
 4x4-bit single-port distributed Read Only RAM         : 1
# Adders/Subtractors                                    : 13
 14-bit adder                                           : 4
 14-bit subtractor                                      : 1
 4-bit adder                                            : 6
 4-bit subtractor                                       : 2
# Counters                                              : 4
 2-bit up counter                                       : 1
 6-bit up counter                                       : 1
 7-bit up counter                                       : 2
# Registers                                             : 57
 Flip-Flops                                             : 57
# Comparators                                           : 13
 14-bit comparator greater                              : 4
 4-bit comparator greater                               : 3
 6-bit comparator greater                               : 1
 7-bit comparator greater                               : 5
# Multiplexers                                          : 100
 14-bit 2-to-1 multiplexer                              : 52
 4-bit 2-to-1 multiplexer                               : 38
 7-bit 2-to-1 multiplexer                               : 8
 7-bit 4-to-1 multiplexer                               : 2

=========================================================================

=========================================================================
*                          Low Level Synthesis                          *
```

```
Mapping completed.
See MAP report file "parking_meter_map.mrp" for details.

Process "Map" completed successfully
```

-design summary



```
Release 14.7 Map P.20131013 (lin64)
Xilinx Mapping Report File for Design 'parking_meter'

Design Information
------------------
Command Line   : map -intstyle ise -p xc7a100t-csg324-3 -w -logic_opt off -ol
high -t 1 -xt 0 -register_duplication off -r 4 -mt off -ir off -pr off -lc off
-power off -o parking_meter_map.ncd parking_meter.ngd parking_meter.pcf
Target Device  : xc7a100t
Target Package : csg324
Target Speed   : -3
Mapper Version : artix7 -- $Revision: 1.55 $
Mapped Date    : Mon Nov 30 03:28:38 2020

Design Summary
--------------
Number of errors:      0
Number of warnings:   37
Slice Logic Utilization:
  Number of Slice Registers:                    76 out of 126,800    1%
    Number used as Flip Flops:                  76
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                        247 out of  63,400    1%
    Number used as logic:                      244 out of  63,400    1%
      Number using O6 output only:             168
      Number using O5 output only:              22
      Number using O5 and O6:                   54
      Number used as ROM:                        0
    Number used as Memory:                       0 out of  19,000    0%
    Number used exclusively as route-thrus:      3
      Number with same-slice register load:      0
      Number with same-slice carry load:         3
      Number with other load:                    0
```

```
Mapping completed.
See MAP report file "parking_meter_map.mrp" for details.

Process "Map" completed successfully
```

-map report

Conclusion

This lab gave me a good experience with coding finite state machines using verilog and xilinx. We designed a parking meter that changes time remaining depending on the input, with 9 possible inputs. Since finite state machines are used everywhere in the real world, I believe having a good practice with it is important, and I think I got a solid first step into this kind of coding. An obstacle I faced is that the numbers would sometimes not add properly, and the 9 would go to a 10. This happened mainly when I changed my code to properly synthesize, and some cases still do fail, which I could not get to all of them. Another obstacle I faced was the blinking mechanism. Especially for cases less than 180 seconds, it was a challenge to code it in a way so that it blinks every second.

Overall, I believe this was a good introduction to finite state machines, and I plan on using the skills I learned in this project to take on my next project, which also uses a finite state machine.