CS M152A - Lab 3, Designing a Parking Meter

Name: Sum Li

UID: 505146702

Due Date: November 29, 2020

TA: Mohit Garg

1) Introduction and Requirement

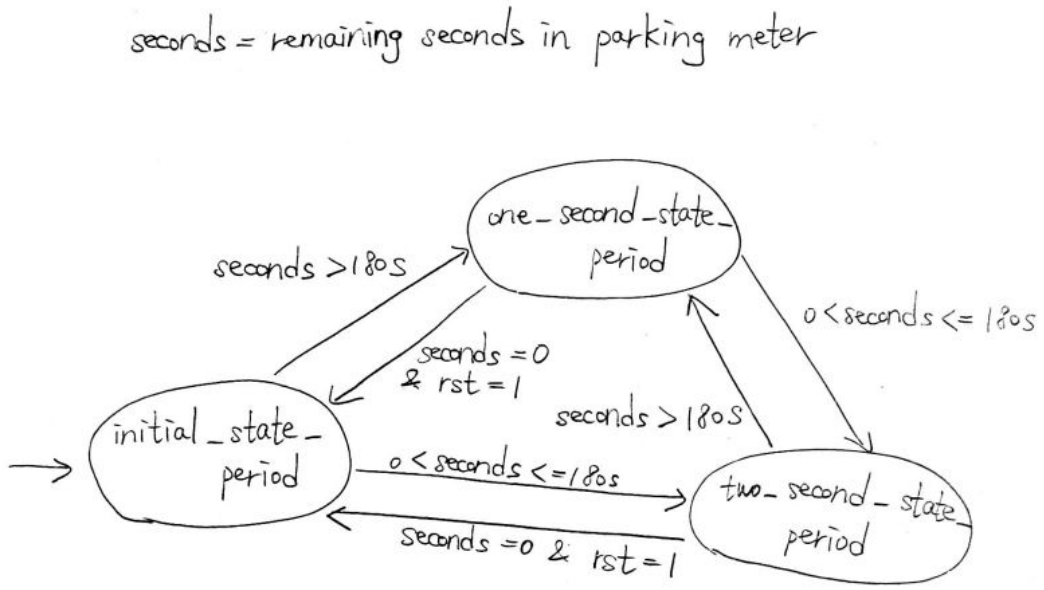
For this lab, the goal is to the Xilinx program to design and test a Finite State Machines (FSMs) which simulates as a parking meter when users added coins to update and display the remaining seconds on the display screen. The system specification divides into input and output modules. The input module consists of 8 input buttons which are add1, add2, add3, add4, rst1, rst2, clk and rst. Each of the buttons represents different coin denominations to add a specific number of seconds to the parking meter. Button add1 adds 60 seconds to the parking meter. Button add2 adds 120 seconds to the parking meter. Button add3 adds 180 seconds to the parking meter. Button add4 adds 300 seconds to the parking meter. Button rst1 resets the parking meter to 16 seconds. Button rst2 resets the parking meter to 150 seconds. Button rst resets the parking meter to the initial state. Input global clock clk drives the parking meter with frequency of 100 Hz.

The output module is formed by a seven segment vector called led_seg which is a 7 bit vector. There are a total of 4 output ports called val3, val2, val1, val0 and each of the ports is a 4 bit vector. Each of the output ports display the digit in binary coded decimal form which then correspond among the 4 segments. There are a total of 4 anodes called a3, a2, a1, a0 and each of the anodes is a 1 bit signal. Each anode selects the correct screen to display the corresponding digit. For example, anode a3 displays the most significant digit (leftmost digit) and a0 displays the least significant digit (rightmost digit).

There are several requirements of the lab. First, if the users press any button among add1, add2, add3, add4, then the corresponding seconds will be added to the meter. In the usual case, the meter starts counting down by decrement 1 at a time. Second, There are three different flashing periods. The first case is when the parking meter seconds is 0, then the meter should flash 0000 with period 1 second and 50% duty cycle. The second case is when the parking meter seconds is between 1 second and 180 seconds (1 sec <= meter <= 180 sec), then the meter should flash with period 2 second and 50% duty cycle. The third case is when the parking meter second is greater than 180 seconds (> 180 sec), then the meter should flash with period 1 second and 50% duty cycle. Third, the maximum value of the meter is 9999. If the user attempt to add in more than 9999 seconds, the meter will stay on displaying 9999 seconds and start counting down from there.

2) Design Descriptions

FSM Diagram and Explanation of the States and Transitions



The inputs of the FSM are the reset signal (rst) of the parking meter and the remaining seconds in the parking meter. There are a total of 3 states in the FSM diagram. The first state (start state) is called “initial_period_state”; the second state is called “one_second_state”; the third state is called “two_second_state”. The output of the FSM is the current state of the parking meter which is among one of the 3 states in the diagram. If the reset signal is being pressed (rst = 1), the parking meter has to move back to “inital_period_state” to indicate the remaining seconds are being reset to 0. When the parking meter seconds is 0 (if rst = 1, the parking meter is also equal to 0 seconds), FSM moves to “initial_state_period” and the meter will flash with period 1 second and 50% duty cycle. When the parking meter seconds is (1 second <= parking meter seconds <= 180 seconds), FSM moves to “two _second_state_period” and the mater will flash with period 2 second and 50% duty cycle. When the parking meter seconds is greater than 180 seconds, FSM moves to “one_second_state_period” and the meter will flash with period 1 second and 50% duty cycle. Based on the FSM states, the parking meter will flash with the corresponding period on the display.

Implementation of period 1 second clock & period 2 seconds clock with 50% duty cycle (flashing period)

In order to create a period 1 second clock with 50% duty cycle. I created a counter called “hundred_counter" that will count up to 100. The counter will divide the input clock (100Hz) and transform that into a period 1 second clock (1Hz). In order to flash with 50% duty cycle which means it is on for 0.5 seconds and off for 0.5 seconds. When the “hundred_counter" is between 0 and 49 (0 <= hundred_counter <= 49), then the seven segment display is on and “flash” (not the actual flashing since the results are not showing on the hardware). When the “hundred_counter” is greater than 49 (hundred_counter > 49), then the seven segment display is off and not flash. The parking meter will only “flash” with period 1 second clock with 50% duty cycle when it is either in “inital_state_period” or “one_second_state_period” from the FSM state.

In order to create a period 2 second clock, I created a register called “clock_period_two_proceed” with 2 seconds period and it is driven by my “hundred_counter” wich means the clock will flip after my “hundred counter” has run for one clock cycle in order to create the 50% duty cycle. The parking meter will only “flash” with a period 2 seconds clock with 50% duty cycle when it is in “two _second_state_period” from the FSM state.

Implementation of the Parking Meter Counter

In order to record the most updated remaining seconds of the parking meter, I created a 14 bits register variable called “parking_meter_seconds” to store the value. For each clock cycle, the “parking_meter_seconds” counter will decrease by 1 each time. If the user pressed a button (add1, add2, add3, add4), the corresponding value of seconds will be added to the “parking_meter_seconds” counter. However, the parking meter can only read exactly 1 input for each second.

Implementation of the Seven Segment Vector Display

In order to create the seven segment vector display, I wrote a separate module that takes three sets of inputs which are the 4 output ports (val3, val2, val1, val0), the input clock and reset signal. The input variable val3 represents the number at the thousand place; the input variable val2 represents the number at the hundred place; the input variable val1 represents the number at the tens place; the input variable val0 represents the number at the ones place. The input clock is

the same as the global clock of the parking meter module which is a frequency of 100Hz. There are two sets of outputs which are the 4 anodes (a3, a2, a1, a0) and the 7 bit output register which represents the 7 cathodes. According to the Nexys3 reference manual for the seven-segment display, it is driven by the 4 anodes and the 7 cathodes (CA, CB, CC, CD, CE, CF, CG, DP). Each of the 4 anodes will be set to 0 sometimes according to the correct order in order to refresh the display.

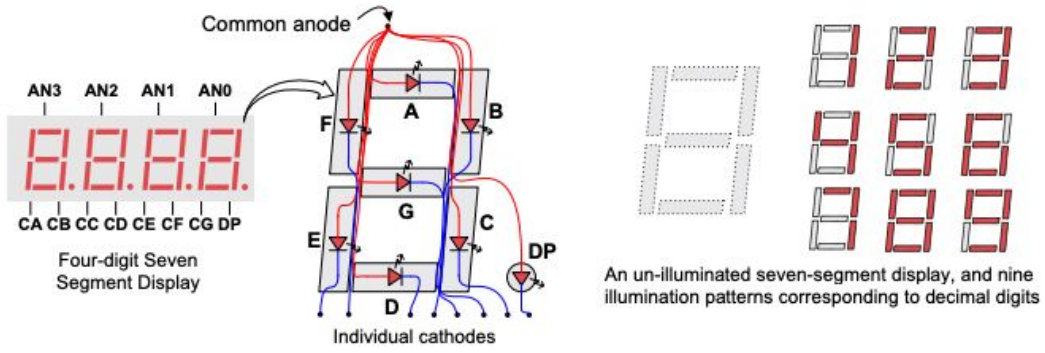


Figure 1: Illustration of the four digit seven segment display

Each digit in binary coded decimal form (4 bits) will be encoded as a specific value for each of the cathodes. The following table shows how the encoding works:

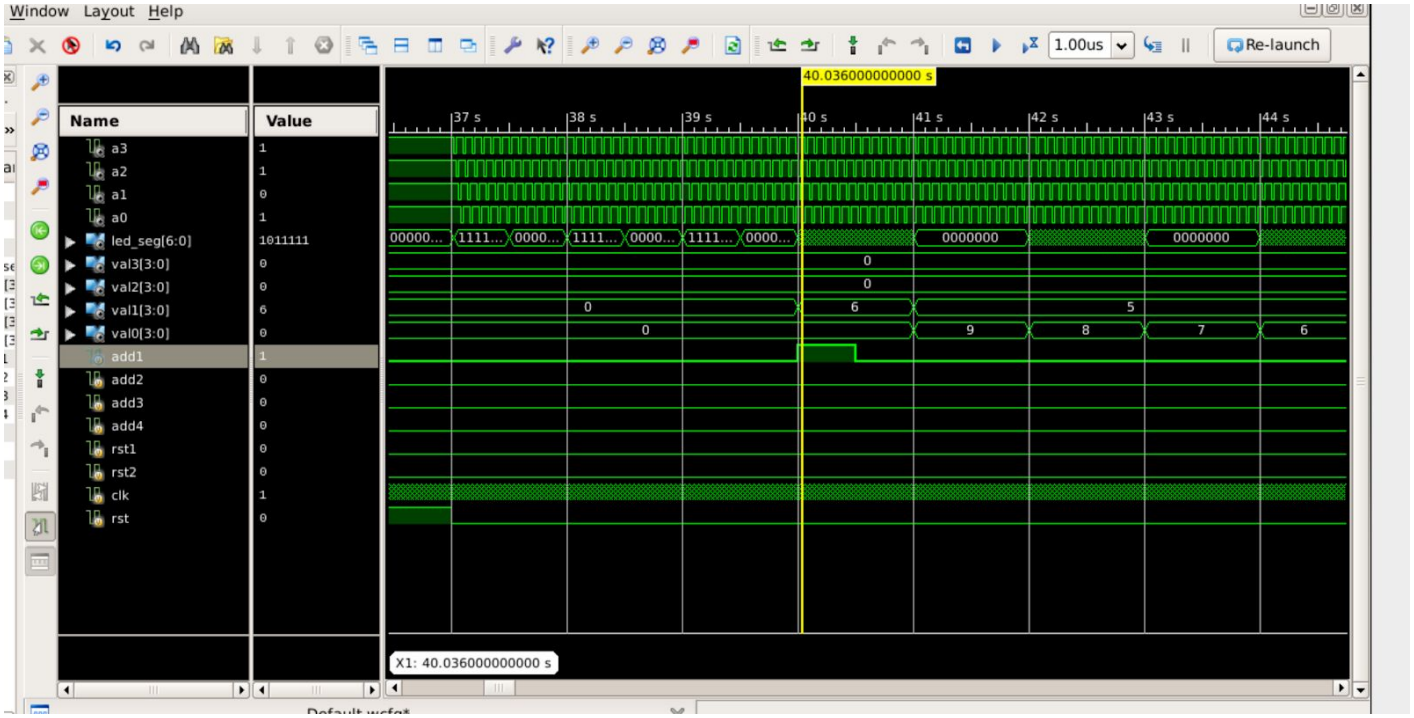
Decimal Digit	BCD Digit	CA	CB	CC	CD	CE	CF	CG
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1

The following is the high level schematics generated from Xilinx ISE for the module parking meter. According to the diagram, there are a total of 8 inputs which are the 4 input buttons (add1, add2, add3, add4), input clock, 3 reset signals (rst, rst1, rst2). According to the diagram, there are a total 9 outputs which are 4 output ports (val3, val2, val1, val0), 4 anodes (a3, a2, a1, a0) and the 7 bits led_seg.

Figure 2: High level schematic of the parking meter module

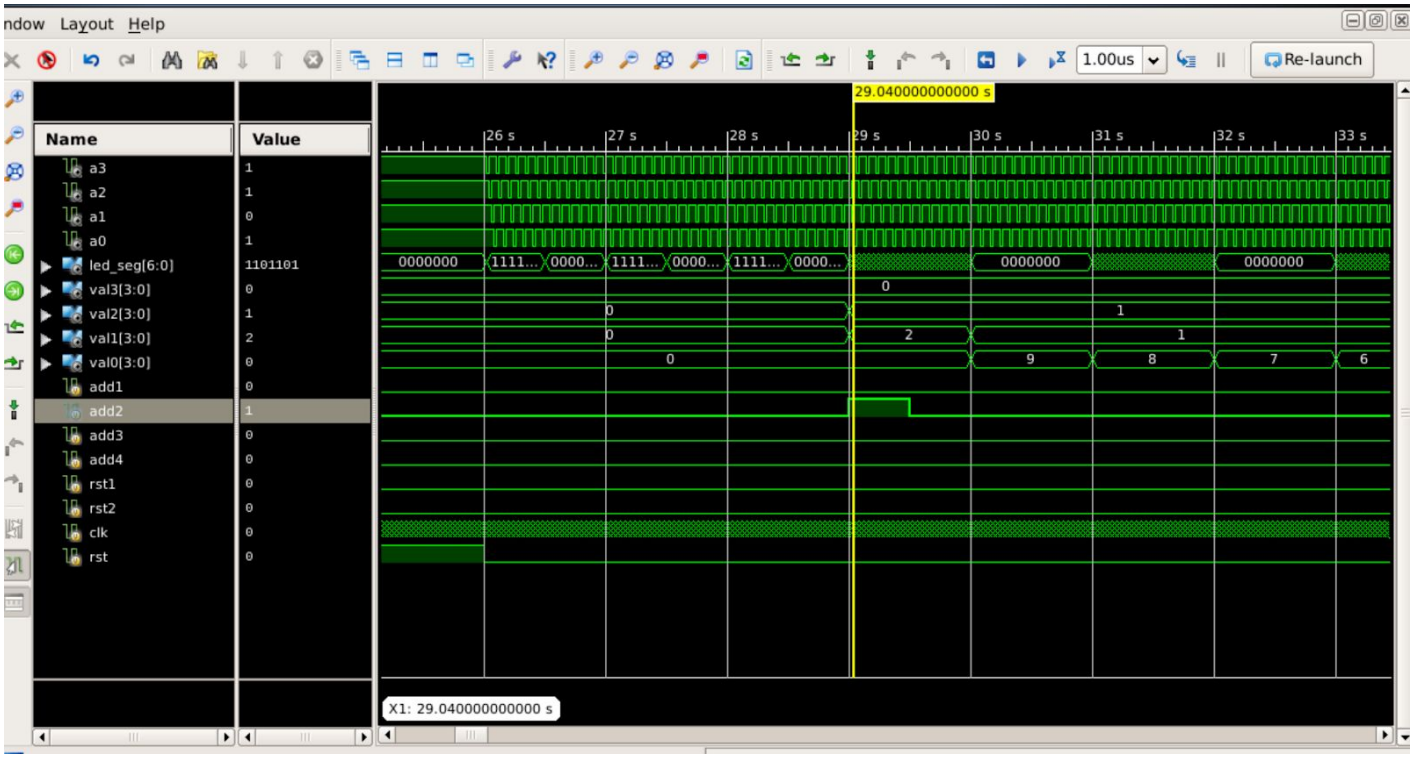
3) Simulation Documents

Testing of input add1



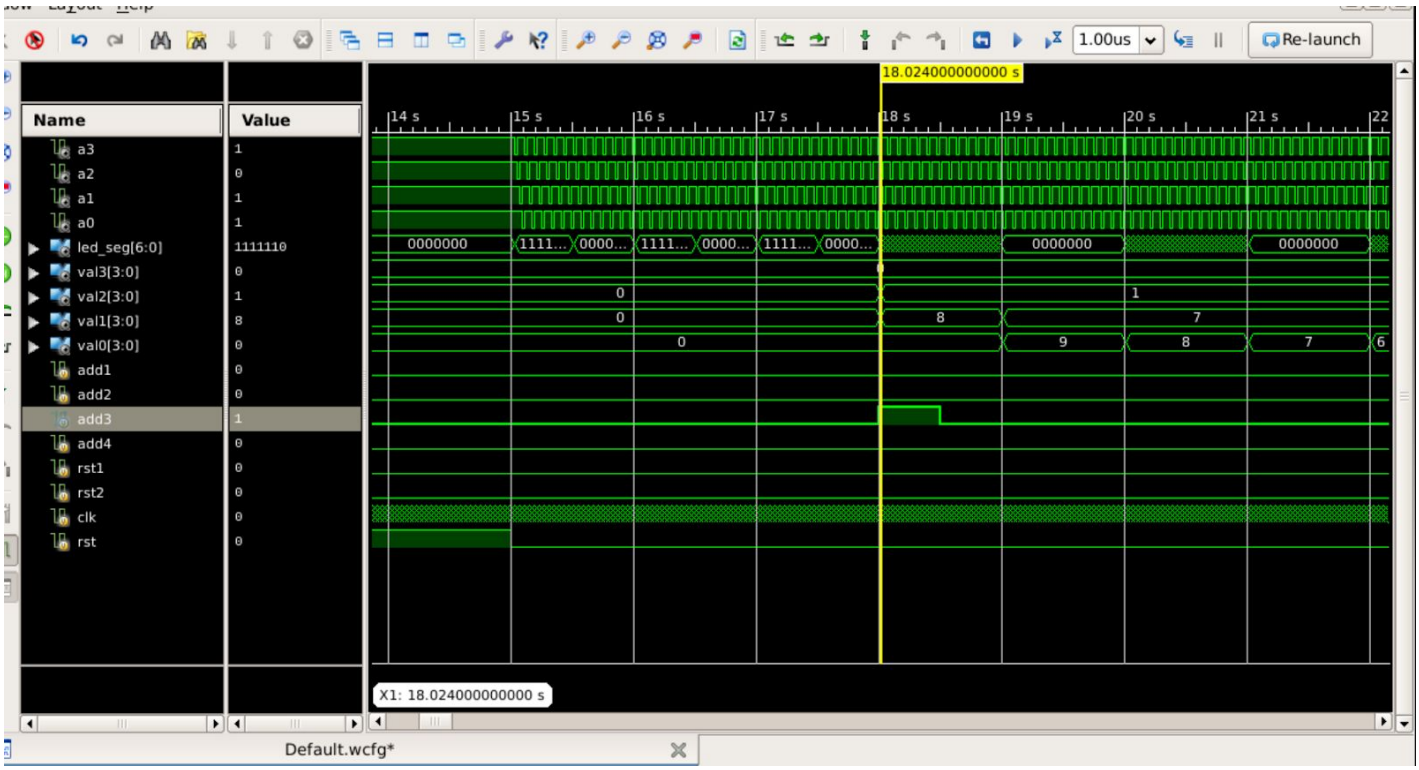
The above waveform diagram shows when input signal “add1” = 1 which means to add 60 seconds to the parking meter. When time = 40 seconds, the signal of add1 becomes high (1), then the parking meter increases from 0000 (val3 = 0, val2 = 0, val1 = 0, val0 = 0) to 0060 (val3 = 0, val2 = 0, val1 = 6, val0 = 0). After add1 becomes low (0), then the parking meter starts to decrement by 1 each second from 60 seconds, for example, 60 seconds, 59 seconds, 58 seconds, etc. Since the remaining seconds is less than 180 seconds, the 7 bits led_seg is flashing with a period 2 seconds with 50% duty cycle.

Testing of input add2



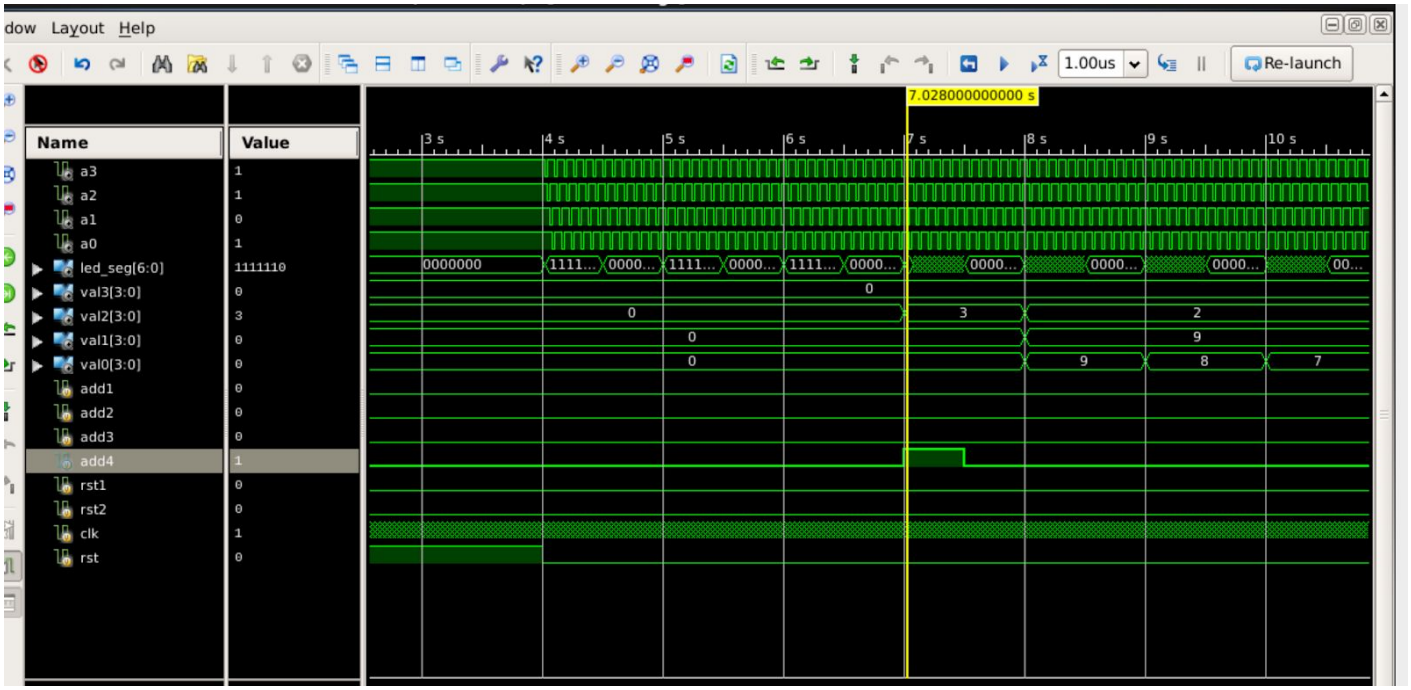
The above waveform diagram shows when input signal “add2” = 1 which means to add 120 seconds to the parking meter. When time = 29 seconds, the signal of add2 becomes high (1), then the parking meter increases from 0000 (val3 = 0, val2 = 0, val1 = 0, val0 = 0) to 0120 (val3 = 0, val2 = 1, val1 = 2, val0 = 0). After “add2” becomes low (0), then the parking meter starts to decrease by 1 each second from 120 seconds, for example, 120 seconds, 119 seconds, 118 seconds, etc. Since the remaining seconds is less than 180 seconds, the 7 bits led_seg is flashing with a period 2 seconds with 50% duty cycle.

Testing of input add3



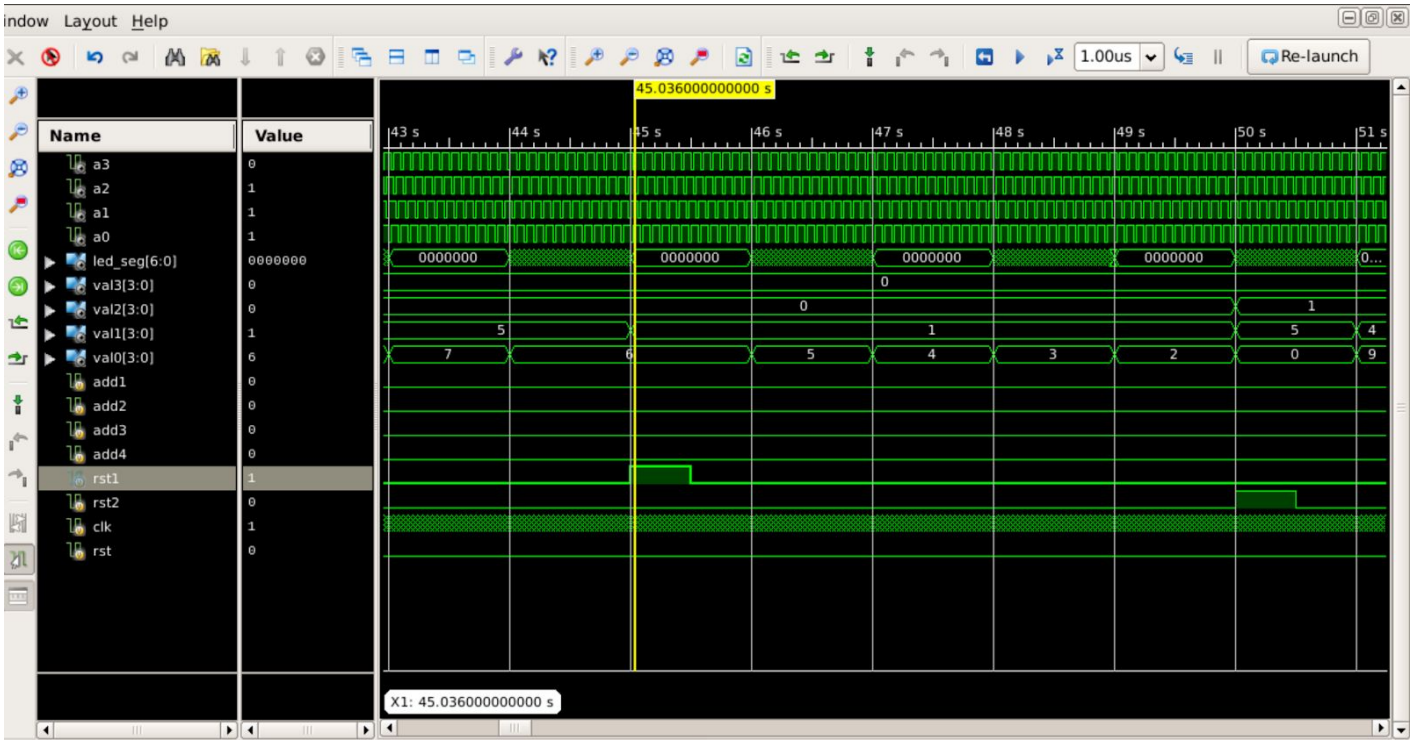
The above waveform diagram shows when input signal “add3” = 1 which means to add 180 seconds to the parking meter. When time = 18 seconds, the signal of add3 becomes high (1), then the parking meter increases from 0000 (val3 = 0, val2 = 0, val1 = 0, val0 = 0) to 0180 (val3 = 0, val2 = 1, val1 = 8, val0 = 0). After “add3” becomes low (0), then the parking meter decreases by 1 each second from 180 seconds, for example, 180 seconds, 179 seconds, 178 seconds, etc. Since the remaining seconds is equal to 180 seconds, the 7 bits led_seg is flashing with a period 2 seconds with 50% duty cycle.

Testing of input add4



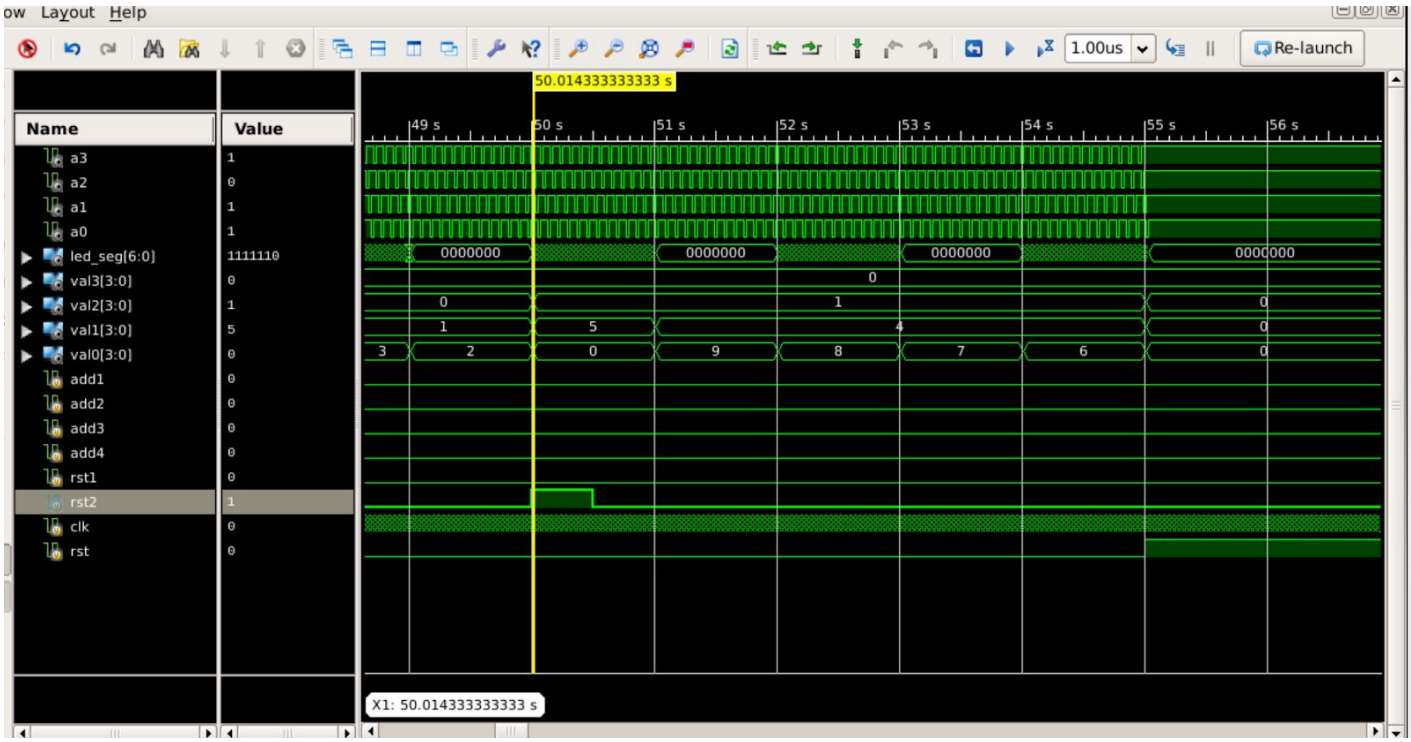
The above waveform diagram shows when input signal “add4” = 1 which means to add 300 seconds to the parking meter. When time = 7 seconds, the signal of “add4” becomes high (1), then the parking meter increases from 0000 (val3 = 0, val2 = 0, val1 = 0, val0 = 0) to 0300 (val3 = 0, val2 = 3, val1 = 0, val0 = 0). After “add4” becomes low (0), then the parking meter decreases by 1 each second from 300 seconds, for example, 300 seconds, 299 seconds, 298 seconds, etc. Since the remaining seconds is greater than 180 seconds, the 7 bits led_seg is flashing with a period 1 second with 50% duty cycle.

Testing of input rst1



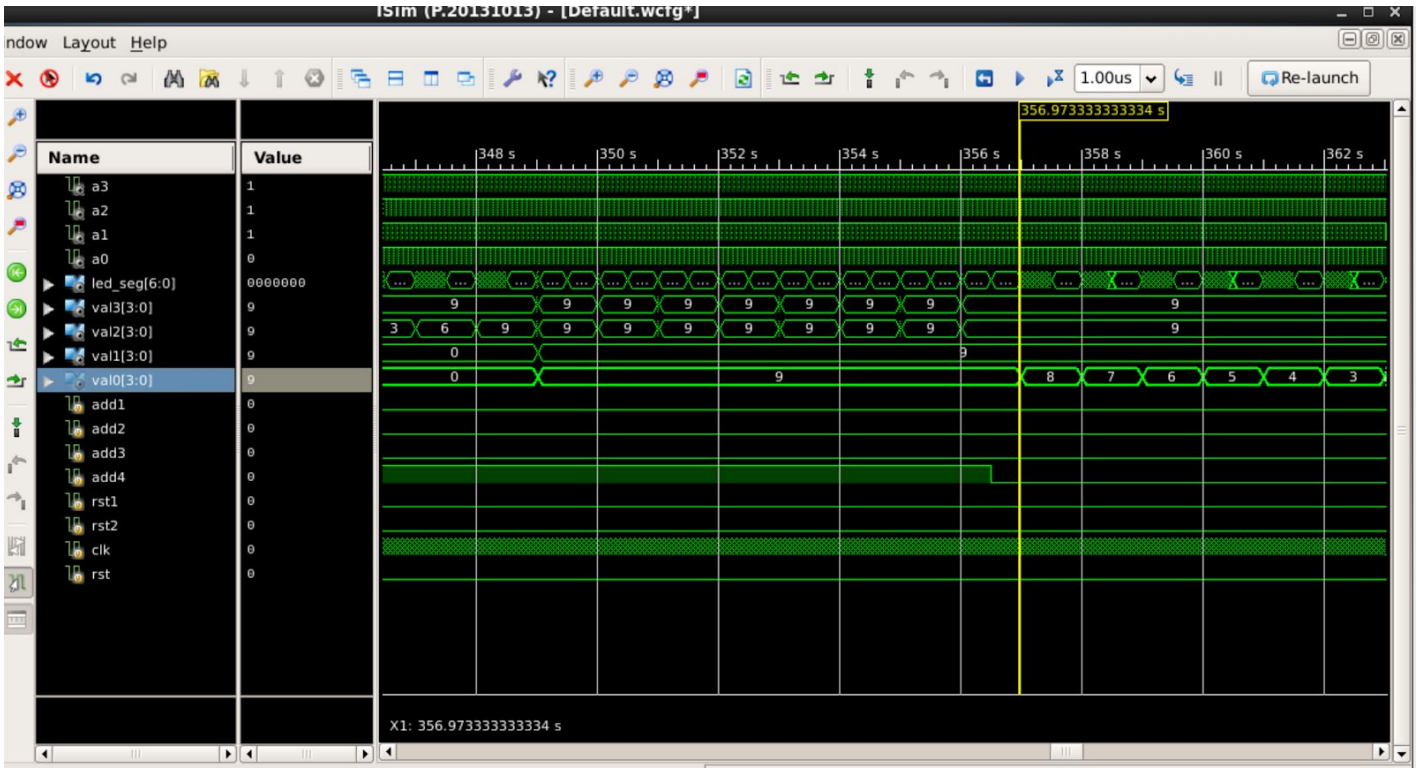
The above waveform diagram shows when input signal “rst1” = 1 which means to reset the parking meter seconds back to 16 seconds. When time = 45 seconds, the signal of “rst1” becomes high (1), then the parking meter makes a jump to 0016 (val3 = 0, val2 = 0, val1 = 1, val0 = 6). After “rst1” becomes low (0), then the parking meter decreases by 1 each second from 16 seconds, for example, 16 seconds, 15 seconds, 14 seconds, etc. Since the remaining seconds is less than 180 seconds, the 7 bits led_seg is flashing with a period 2 seconds with 50% duty cycle.

Testing of input rst2



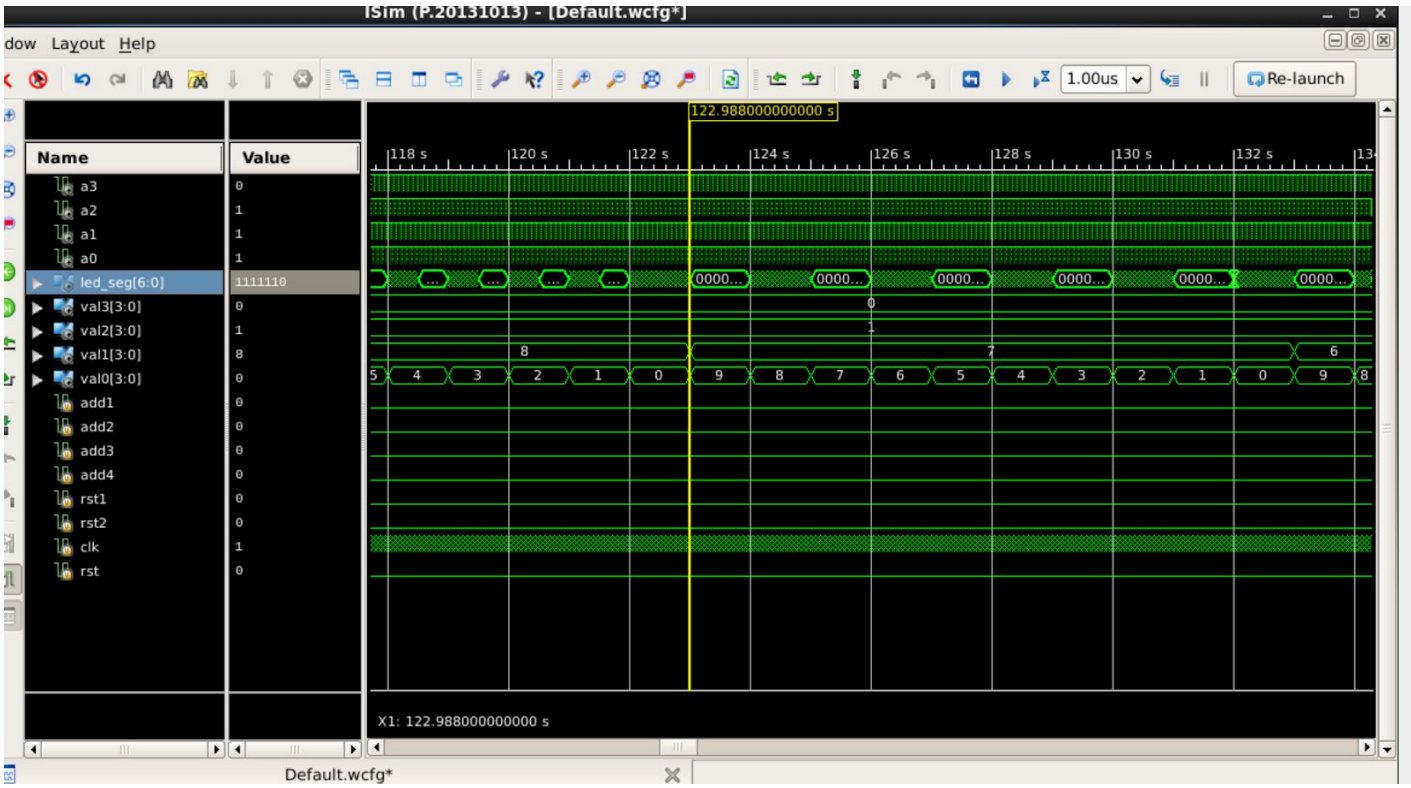
The above waveform diagram shows when input signal “rst2” = 1 which means to reset the parking meter seconds back to 150 seconds. When time = 50 seconds, the signal of “rst2” becomes high (1), then the parking meter makes a jump to 0150 (val3 = 0, val2 = 1, val1 = 5, val0 = 0). After “rst2” becomes low (0), then the parking meter decreases by 1 each second from 150 seconds, for example, 150 seconds, 149 seconds, 148 seconds, etc. Since the remaining seconds is less than 180 seconds, the 7 bits led_seg is flashing with a period 2 seconds with 50% duty cycle.

Testing of maximum value of stored seconds



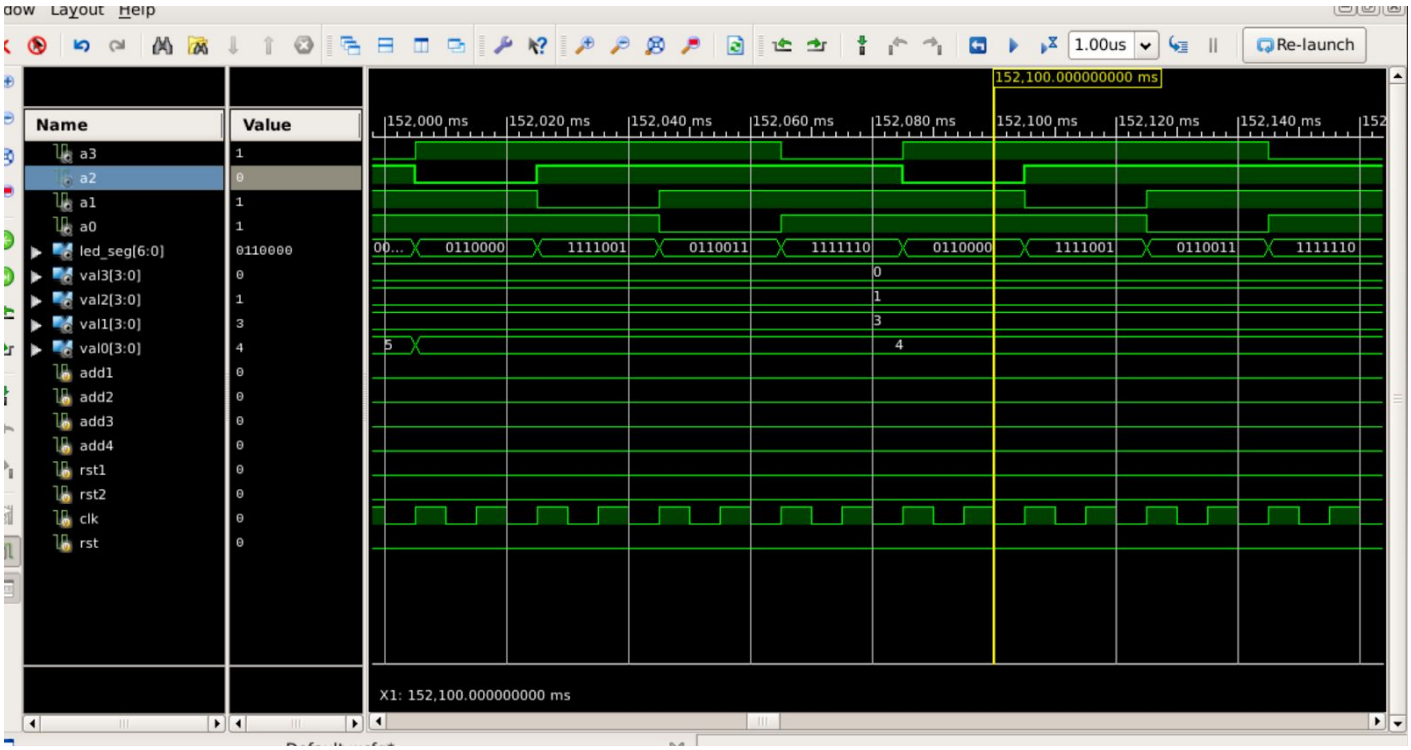
The above waveform diagram shows when the parking meter once reaches 9999 seconds and the value will stay at the maximum value of seconds 9999. When time = 357 seconds after the add button becomes low, the parking meter then the parking meter decreases by 1 each second from 9999 seconds, for example, 9999 seconds, 9998 seconds, 9997 seconds, etc. Since the remaining seconds is greater than 180 seconds, the 7 bits led_seg is flashing with a period 1 seconds with 50% duty cycle.

Testing different flashing period (less than 180 seconds and greater than 180 seconds)



The above waveform diagram shows when the parking meter seconds are <= 180 seconds, it is flashing with period 2 second clock with 50% duty cycle. At time = 123 seconds, the parking meter seconds are 180 seconds so it flashes with the corresponding pattern. At time = 120 seconds, the parking meter seconds are > 180 seconds such as 182 seconds, so the parking meter flashes with period 1 second clock.

Testing of the Seven Segment Vector Display



The above waveform diagram shows the four scenarios of the flashing display which are (a3 = 0, a2 = 1, a1 = 1, a0 = 1), (a3 = 1, a2 = 0, a1 = 1, a0 = 1), (a3 = 1, a2 = 1, a1 = 0, a0 = 1), (a3 = 1, a2 = 1, a1 = 1, a0 = 0). For example, when the signal of a3 = 1, a2 = 0, a1 = 1, a0 = 1 and the output register is displaying 0110000. The second screen (counted from the leftmost side) is flashing with the digit 1 which is the same value of the output port val2. When the signal of a3 = 1, a2 = 1, a1 = 0, a0 = 1 and the output register is displaying 1111001. The third screen (counted from the leftmost side) is flashing with the digit 3 which is the same value of the output port val1.

ISE Design Summary Report

parking_meter Project Status (11/25/2020 - 05:52:04)			
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:**	7 Warnings (2 new)
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	84	18,224	1%	
Number used as Flip Flops	83			
Number used as Latches	1			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	445	9,112	4%	
Number used as logic	443	9,112	4%	
Number using O6 output only	411			
Number using O5 output only	0			
Number using O5 and O6	32			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number using O5 output only	0			
Number using O5 and O6	32			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number used exclusively as route-thrus	2			
Number with same-slice register load	1			
Number with same-slice carry load	1			
Number with other load	0			
Number of occupied Slices	163	2,278	7%	
Number of MUXCYs used	28	4,556	1%	
Number of LUT Flip Flop pairs used	470			
Number with an unused Flip Flop	390	470	82%	
Number with an unused LUT	25	470	5%	
Number of fully used LUT-FF pairs	55	470	11%	
Number of unique control sets	9			
Number of slice register sites lost to control set restrictions	36	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 slice register sites lost to control set restrictions	36	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	4.92			

Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	4.92			

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	Wed Nov 25 05:49:29 2020	0	5 Warnings (0 new)	2 Infos (0 new)
Translation Report	Current	Wed Nov 25 05:51:23 2020	0	0	0
Map Report	Current	Wed Nov 25 05:51:40 2020	0	1 Warning (1 new)	6 Infos (6 new)
Place and Route Report	Current	Wed Nov 25 05:51:49 2020	0	0	3 Infos (3 new)
Power Report					
Post-PAR Static Timing Report	Current	Wed Nov 25 05:51:54 2020	0	0	4 Infos (4 new)
Bitgen Report	Current	Wed Nov 25 05:52:02 2020	0	1 Warning (1 new)	0

Secondary Reports		[-]
Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Wed Nov 25 05:46:14 2020
WebTalk Report	Current	Wed Nov 25 05:52:03 2020
WebTalk Log File	Current	Wed Nov 25 05:52:03 2020

Date Generated: 11/25/2020 - 05:52:47

According to the summary and utilization report, I have used 84 registers for storing variables, output and 83 flip flops. I have also used if else statements which are shown by the number of comparators.

4) Conclusion

After I have finished this lab, I understand how to use Verilog and Xilinx ISE programs to design an FSM model to simulate the real world system which is a parking meter in this project. Through the lab, I had an opportunity to understand the concepts behind Finite State Machines (FSMs) and how to use the idea to model a machine such that the outputs depend on

both the current state and current input. One of the difficulties that I have encountered is to figure out how to switch different flashing periods under different numbers of remaining seconds in the parking meter. Second, it also takes me a while to figure out the encoding from the 4 bit BCD number to the 7 bits cathode number. However, I am able to figure out with the help of the reference manual and the QA sessions.