CS M152A Project 3 Report: Finite State Machine Design — Vending Machine

Belle Lerdworatawee, Lab 1
TA: Ananya Ravikumar
Feb 28, 2021

Introduction and Requirement:

This lab focuses on learning how to design, implement, and most importantly test a finite state machine (FSM) that is a vending machine. There are two types of FSMs: Moore whose transitions depend solely on the input and Mealy whose transitions depend on the present state and input. FSMs prove to be very useful in modeling the behavior of real-world systems like sequential circuits. These machines have a certain number of states that they transition to and from; they must be initialized to a state. For the purposes of this lab, the vending machine has these characteristics: 20 snack varieties, each with their own compartment of size 10 and unique item code. A person can only buy 1 item at a time and payment is card-only. More specific details on the vending machine will be covered in the discussion about its states.

Design Description

For the design and implementation, I took inspiration from the turnstile example provided in the project description. I walked through each of the Action Items and made those into states, then added or removed states as necessary. There's only one module in my design, which accepts 8 inputs: clock, reset, reload, card in, item code, key press, valid transaction, and door open. It has 4 outputs: vend, invalid sale, cost, and failed transaction. These inputs and outputs are intended to simulate real life interactions with a vending machine. I designed a Moore machine to capture the behavior of the vending machine, which means that the output depends solely on the state. It has 10 different states, which I will detail below.

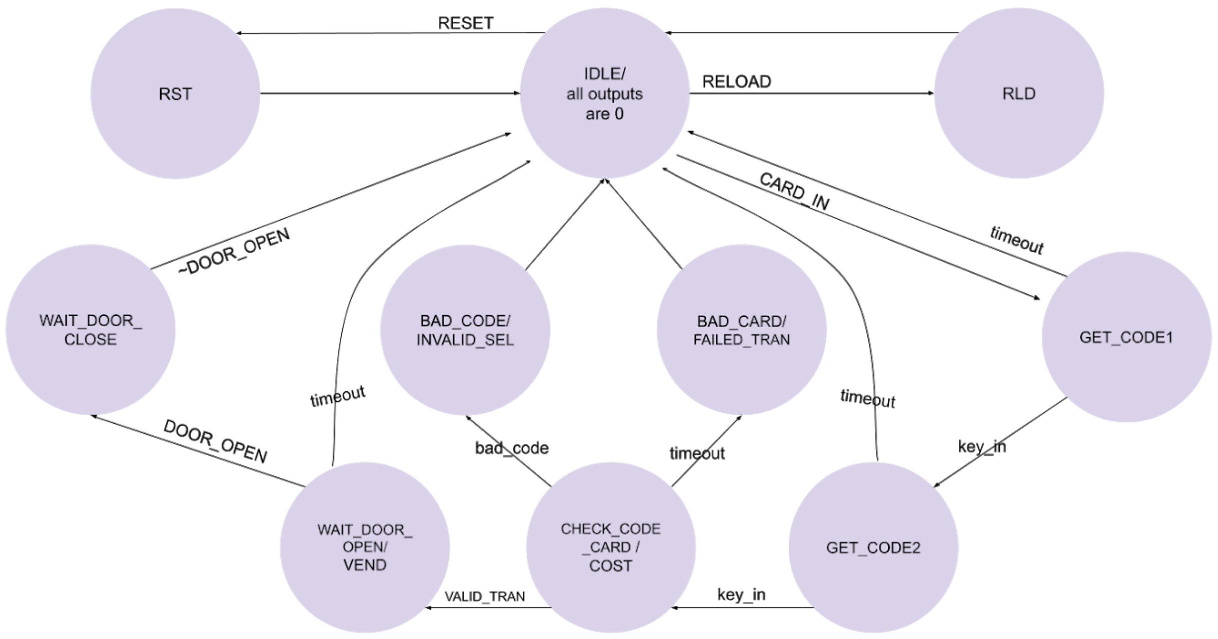
1. IDLE:
    - the machine starts off in this state after RESET goes high
    - sets all the outputs and intermediate flags/variables to 0
    - represents the state where no transaction has been initiated
    - waits for CARD_IN to go high to signify a transaction
2. GET_CODE1
    - once CARD_IN goes high, the machine transits from IDLE to this state
    - resets the 2nd timer to 0, and then uses the 1st timer to count to 5 before setting timeout to high
    - polls for KEY_PRESS and ITEM_CODE
    - when timeout goes high, it returns to IDLE

- otherwise, it detects the KEY_PRESS signal, stores ITEM_CODE, and moves to GET_CODE2

3. GET_CODE2
    - resets the 1st timer to 0, and then uses the 2nd timer to count to 5 before setting timeout to high
    - also polls for KEY_PRESS and ITEM_CODE
    - when timeout goes high, it returns to IDLE
    - otherwise, it detects the KEY_PRESS signal, stores ITEM_CODE, and moves to CHECK_CODE_CARD
4. CHECK_CODE_CARD
    - resets the 2nd timer to 0, and then uses the 1st timer to count to 5 before setting timeout to high
    - performs checks by priority as listed here: checks valid code, if selected snack has snacks to vend, and polls for VALID_TRAN
    - upon detecting an invalid code, it sets an intermediary bad_code to high and then moves to the BAD_CODE state
    - upon detecting a valid code but invalid selection (snack has 0 items), it sets an intermediary bad_code to high and then moves to the BAD_CODE state
        - it will also output the COST of the item in this state
    - upon detecting a valid code and selection, it polls for VALID_TRAN for 5 cycles before setting timeout to high
        - then it outputs FAILED_TRAN, and moves back to the IDLE state
    - if all the above conditions have been met, then it moves to the WAIT_DOOR_OPEN state
5. WAIT_DOOR_OPEN
    - resets the first timer to 0, and then starts the 2nd timer to count to 5 before setting timeout to high
    - polls for DOOR_OPEN to go high while outputting the VEND signal
    - when DOOR_OPEN goes high, it moves to the WAIT_DOOR_CLOSE state
    - otherwise, if 5 cycles have passed, it returns to the IDLE state
6. WAIT_DOOR_CLOSE
    - polls for DOOR_OPEN to go low
    - remains in this state until DOOR_OPEN goes low or RESET to go high, after which it will go to IDLE in the former case and RST in the latter case
    - also will output the VEND signal
7. BAD_CODE
    - this state's purpose is purely to output the INVALID_SEL signal
    - it will automatically move to IDLE after 1 clock cycle
8. BAD_CARD
    - this state's purpose is purely to output the FAILED_TRAN signal
    - it will automatically move to IDLE after 1 clock cycle
9. RLD
    - this state automatically moves to IDLE after RESET goes low
    - the purpose of this state is to set all snack counters to 10

10. RST

- this state automatically moves to IDLE after RESET goes low
- the purpose of this state is to set all snack counters to 10



Assume that inputs on transitions are equal to 1.
Assume that outputs shown in the states are equal to 1, and all outputs not shown are equal to 0
Assume that if RST goes high in any state, the machine will transition to go to the RST state
(arrows not included to make the figure simpler)

Figure 1: Vending Machine Finite State Machine diagram

To implement the described FSM above in Verilog, I created the vending machine top module that accepts the same 8 inputs and 4 outputs as listed in an earlier paragraph. The structure of the module is broken into 5 key component always blocks. They each are responsible for separate tasks: one always block triggered on the positive edge of CLK will update the current state to be the next state. Two trigger on the positive edge of CLK are responsible for counting cycles. Then another one triggered on all the wires considers what is the current state, and then determines the value of next_state by checking for specific inputs or flags that have been set. The most important always block is also triggered by its wires, and this determines the output of each state as well as what flags should be set when.

Looking at the data structures, I initialized parameters to represent all the states with a unique 4-bit binary number. Then I have two registers to hold the current state and next state. Instead of having 20 separate registers to represent the snack compartment, I initialized an array of 20

4-bit values. Then I initialize registers to control time events: I created 2 registers to signify when to start the timer, and two variables to actually be the timer itself. I ran into a lot of errors while trying to rely on just 1 timer as there wasn't enough time for the timer to reset during transitions. And so the next state would use a timer that didn't start at 0. To fix this, I use 2 timers and alternate their usage so that one can reset while the other is being used. Next, I have 3 registers to handle the ITEM_CODE input. Because I didn't want to have 20 if statements to check if each ITEM_CODE combination was valid, I instead converted the 2 hex input codes to 1 decimal number. This was accomplished by first storing the 2 ITEM_CODEs into two different 6-bit registers. In the GET_CODE1 state and output always block, I use an if statement to check if the first ITEM_CODE is 1 or 0. I set the intermediary item_code1 to 10 in the first case, 0 in the second case, and 20 if neither of the cases are seen. Then in the GET_CODE2 state, I simply set item_code2 equal to ITEM_CODE. Lastly, in the CHECK_CODE_CARD state, I add the two item_code intermediaries to get 1 decimal number that I can then use for direct, simple comparisons. For instance, I can check for INVALID_SEL by checking if the number is less than 19. I can also directly access the snack compartment register by using this number to index its compartment, and easily subtract one snack or check for an empty compartment. The registers are 6 bits just to ensure that no truncation will occur. Lastly, I created various flags like bad_code, timeout, key_in, and valid to serve as flags that will trigger conditions in the always block that determines next_state. To give a brief overview, *bad_code* is set when the machine detects a code larger than 19 or an empty snack compartment. *timeout* is set when the machine detects that the timer has counted to 5. *key_in* is set whenever the machine detects KEY_PRESSED in the GET_CODE1 or GET_CODE2 states. *valid* is set when the machine has determined the VALID_CARD is high.

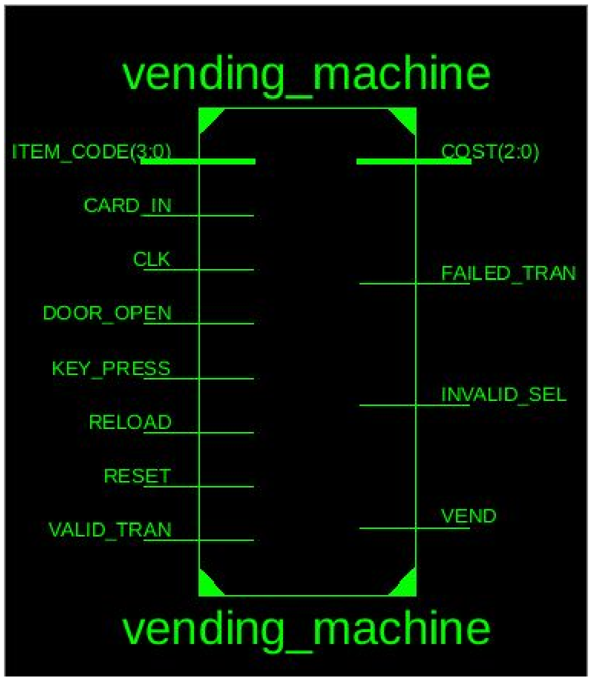


Figure 2: ISE generated schematics of the top module, vending_machine

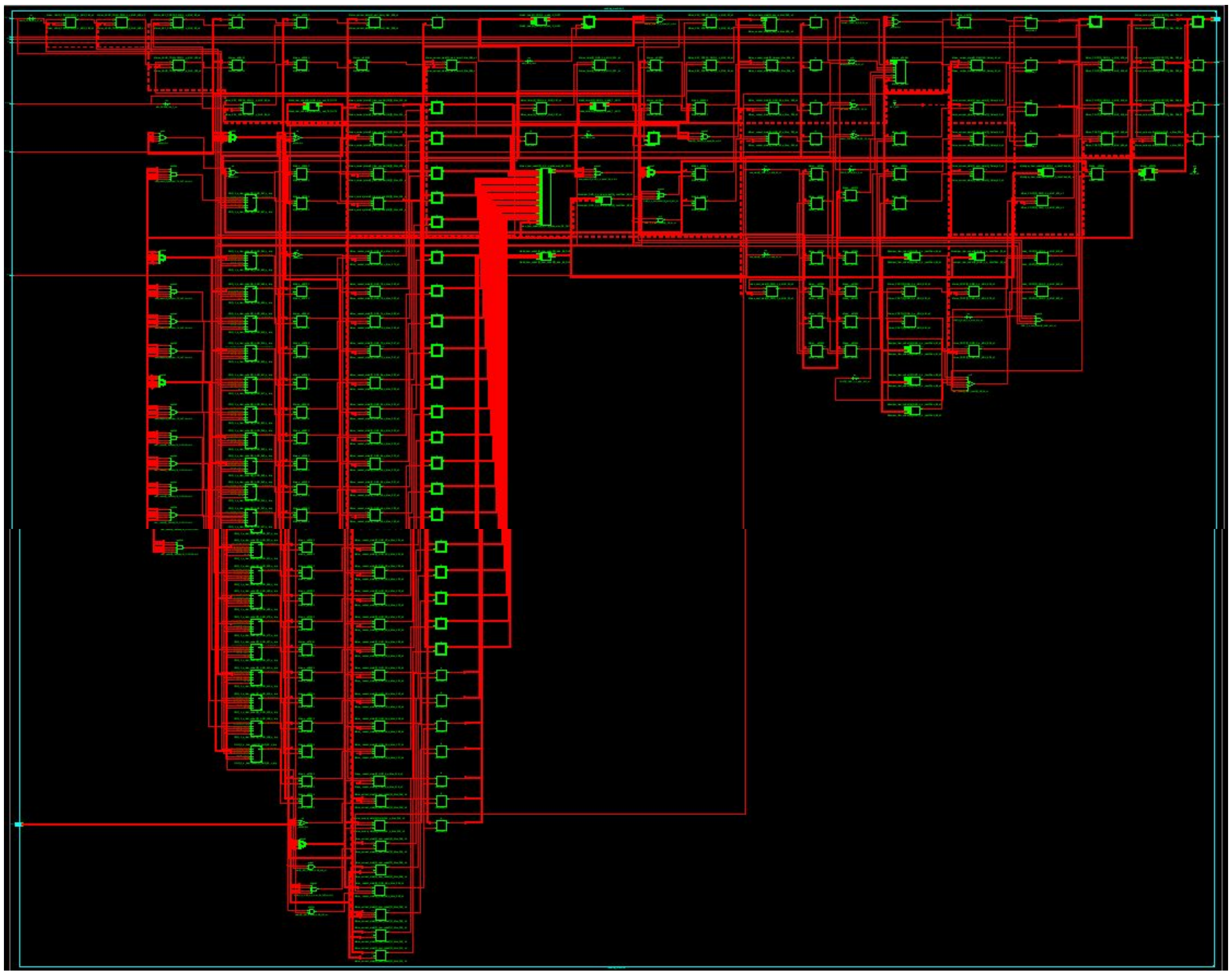


Figure 3: RTL schematic of the top module, vending_machine

Looking at the schematic, we can observe that the diagram displays many muxes, counters, and a lot of registers. My design relies heavily on registers so this makes sense as the FSM needs to keep track of the intermediate flags, the snack compartments, and the states. Additionally, the vending machine needs to perform combinational logic to calculate the next state, time, and the outputs, hence the counters.

Simulation Documentation

To determine if the vending_machine module works as expected, I ran tests on all the possible paths it can take in the state diagram. To create the test cases, I started from the IDLE state and then determined all possible paths from each state after IDLE. For instance, when I move from IDLE to GET_CODE1, I have 2 possible states to move to next: IDLE or GET_CODE2. Each of these became a test case that I needed to check. I have written more on each test case below, and I will briefly outline the important signals in the waveform simulations. The first case will be discussed more extensively than the others.

The testbench contains an always block to generate a 100MHz clock signal, and then it contains another always block to change the inputs. Tests are performed one after the other without resetting the machine.

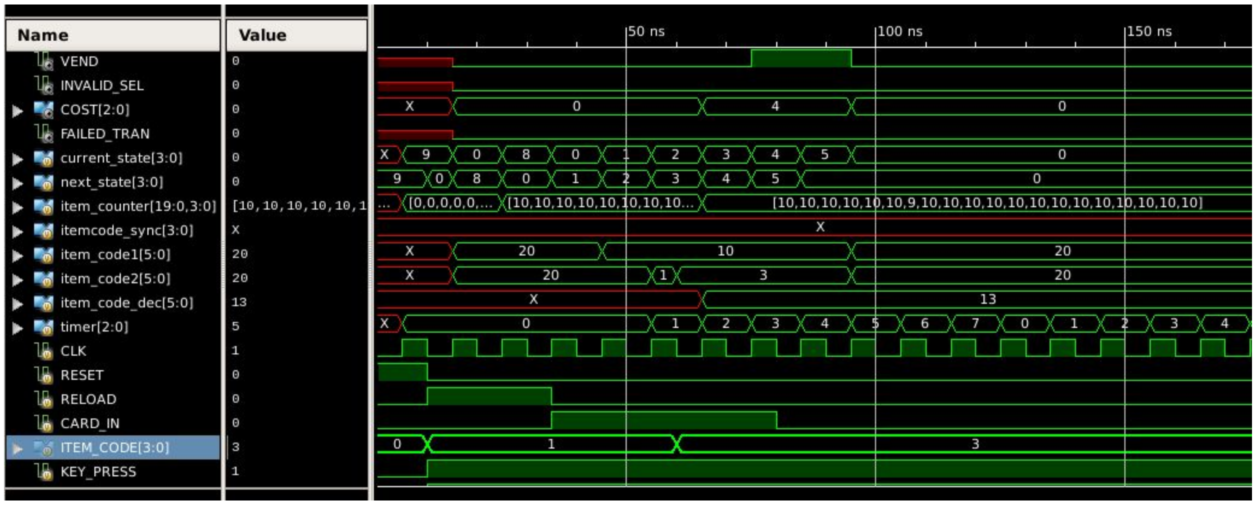


Figure 4: Successful transaction with ITEM_CODE: 13

1. This is a runthrough of a successful transaction from the vending machine selecting snack number 13. The behavior is simulated by first initializing CARD_IN and VALID_CARD to high and setting ITEM_CODE to 0001. Then after item_code1 has been set, I changed ITEM_CODE to 0101. The relevant intermediate signals from inside the module are displayed. We can observe that initially the RESET and RELOAD signal set the item_counter array to 0 and 10 respectively for all the compartments, which is as expected. Then item_code1 becomes 10, item_code2 becomes 3, and item_code_dec is 13. The COST displayed is 4, and the VEND signal goes high. The 13th element of the array is also decremented by one; all these signals demonstrate a successful transaction. Lastly, the states flow from 0 to 5 in order which means that all the state transitions were correct, as I designed the FSM to work that way. The item_codes also start out as bad values, 20, which is what they were initialized to and is good.

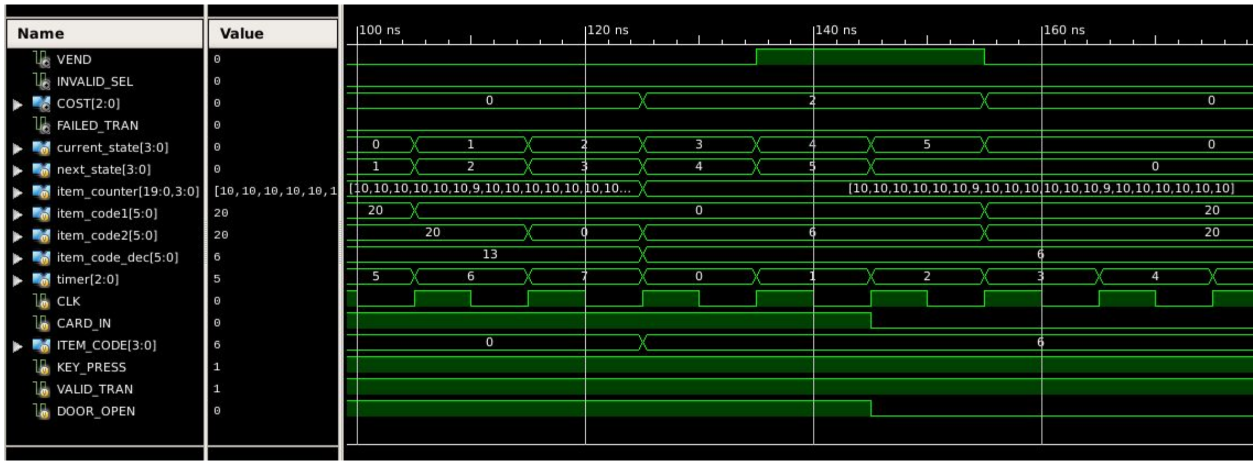
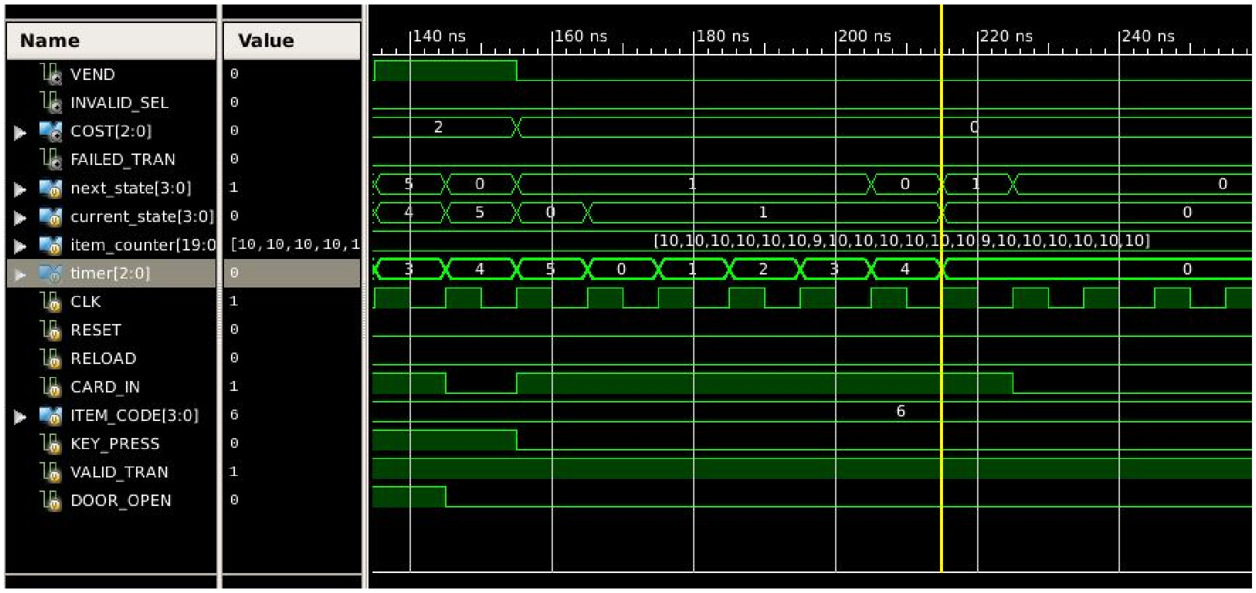


Figure 5: Successful transaction with ITEM_CODE: 06

2. This is a runthrough of another successful transaction from the vending machine selecting snack number 06. I wanted to make sure that the way I implemented the GET_CODE states were correct. The behavior is simulated by first initializing CARD_IN and VALID_CARD to high and setting ITEM_CODE to 0000. After item_code1 has been set, I changed ITEM_CODE to 0110. item_code1 becomes 0, item_code2 becomes 6, and item_code_dec is 6. The COST displayed is 2, and the 6th element of the array is also decremented by one. The states still flow in order which means the behavior is as expected.



Name	Value
VEND	0
INVALID_SEL	0
COST[2:0]	0
FAILED_TRAN	0
next_state[3:0]	1
current_state[3:0]	0
item_counter[19:0	[10,10,10,10,1
timer[2:0]	0
CLK	1
RESET	0
RELOAD	0
CARD_IN	1
ITEM_CODE[3:0]	6
KEY_PRESS	0
VALID_TRAN	1
DOOR_OPEN	0

Figure 6: timeout for item_code1

3. Next, I wanted to test the GET_CODE1 state and make sure it was transitioning back to IDLE after 5 cycles. I simulated the behavior by enabling CARD_IN, but disabling KEY_PRESS to signify that no key is pressed despite the ITEM_CODE. Looking at the waveforms, we can see that current_state remains 1 for 5 cycles and the timer counts from 0 to 4 which is the intended behavior. No outputs are high, and the timer also resets after the state goes from 1 (GET_CODE1) to 0 (IDLE).

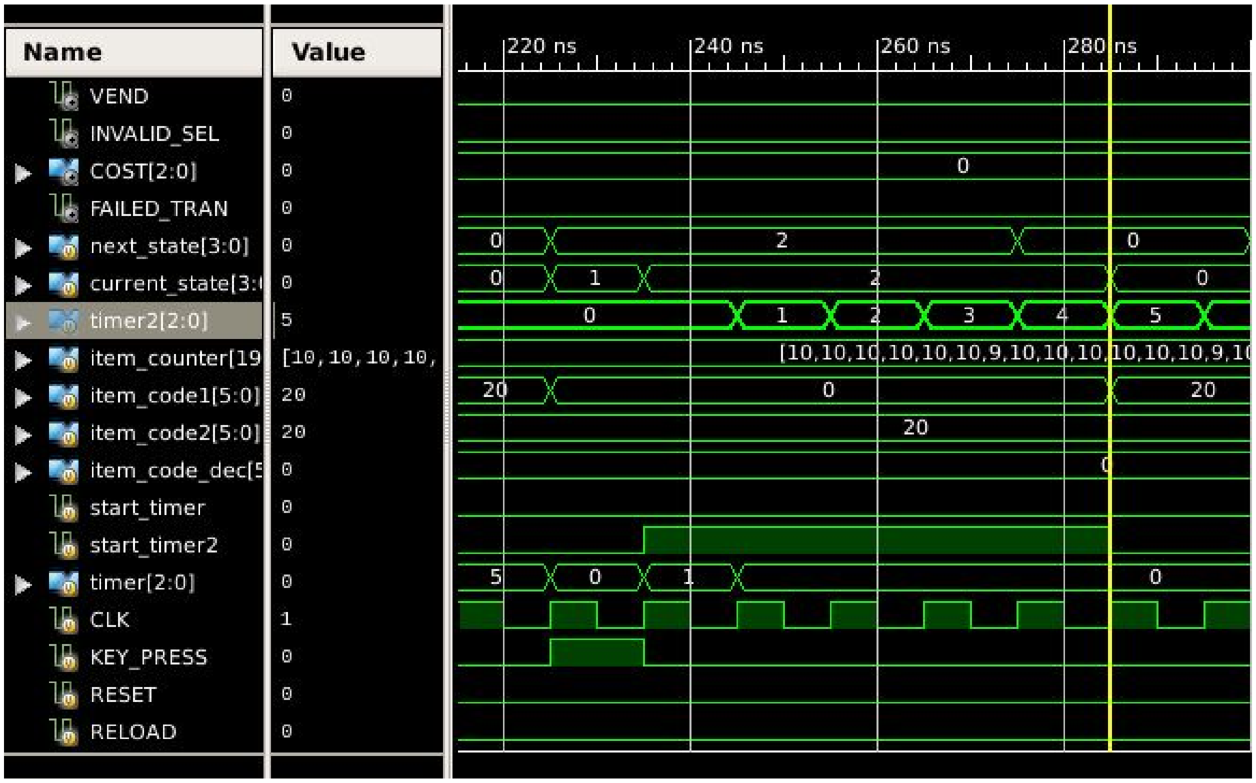


Figure 7: timeout for item_code2

4. Afterwards, I wanted to make sure that the same behavior for timeout occurs in the GET_CODE2 state. The behavior is simulated by keeping the same inputs high as last time, but also enabling KEY_PRESS until the end of GET_CODE1. Similarly, we can see that current_state remains 2 (GET_CODE2) for 5 clock cycles and that timer2 counts from 0 to 4. After this, the current state goes from 2 (GET_CODE2) to 0 (IDLE) which is correct behavior.

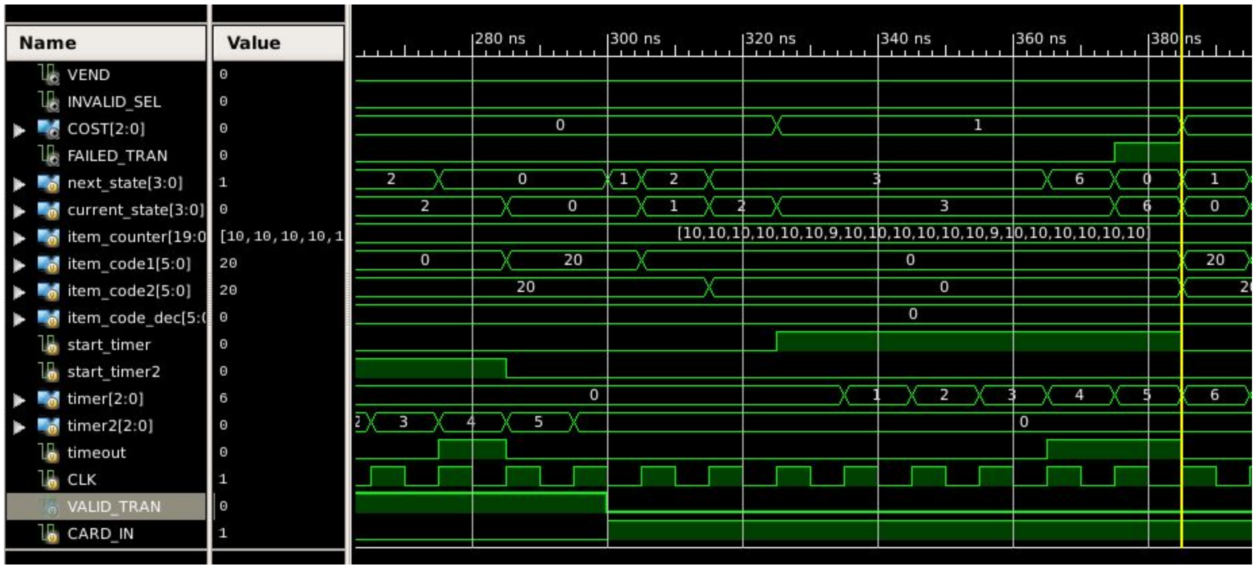


Figure 8: timeout for VALID_TRAN

5. The test after this is to check the timeout functionality in the CHECK_CODE_CARD state. I implemented all of the timeouts the same way and so I tested them roughly in the same manner. With all the inputs kept the same as last time, I left KEY_PRESS on high for the entirety of this test and turned VALID_TRAN to low. From the waveforms, the machine stays in state 3 (CHECK_CODE_CARD) for 5 cycles before moving to state 6 (BAD_CARD). During this period, the FAILED_TRAN signal is also triggered which is correct. Furthermore, ITEM_CODE is 00 which outputs the correct COST of 1.

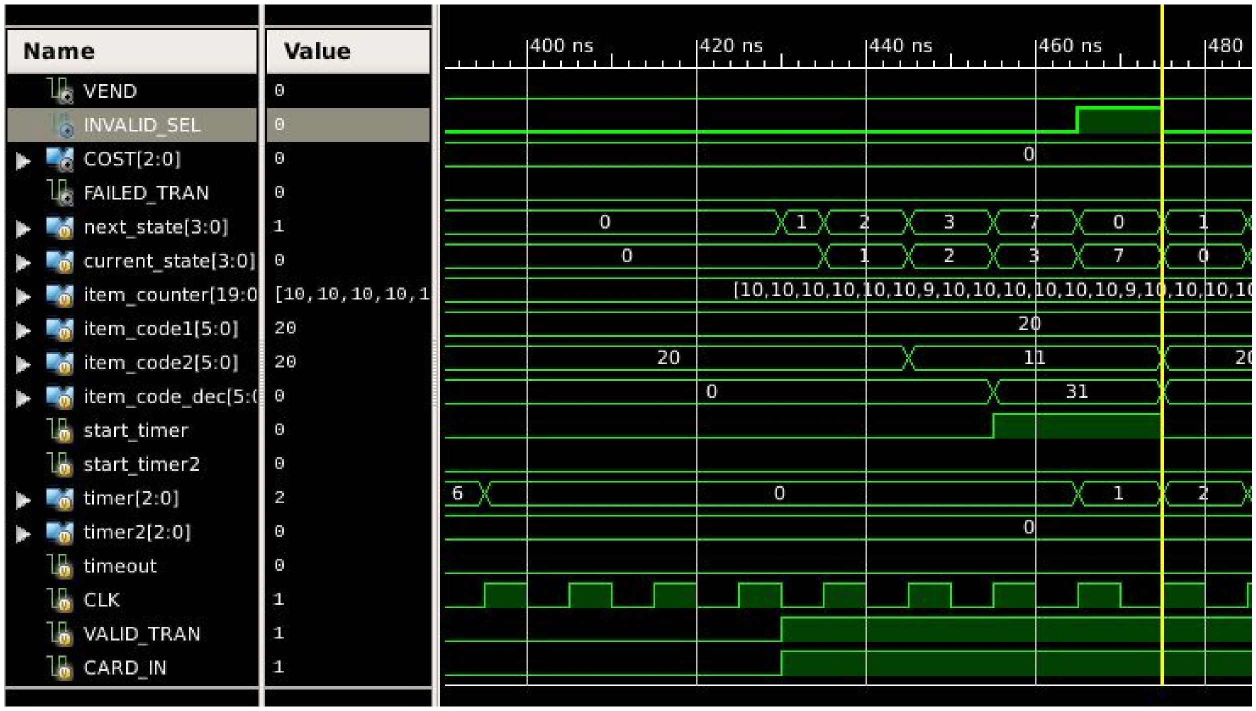
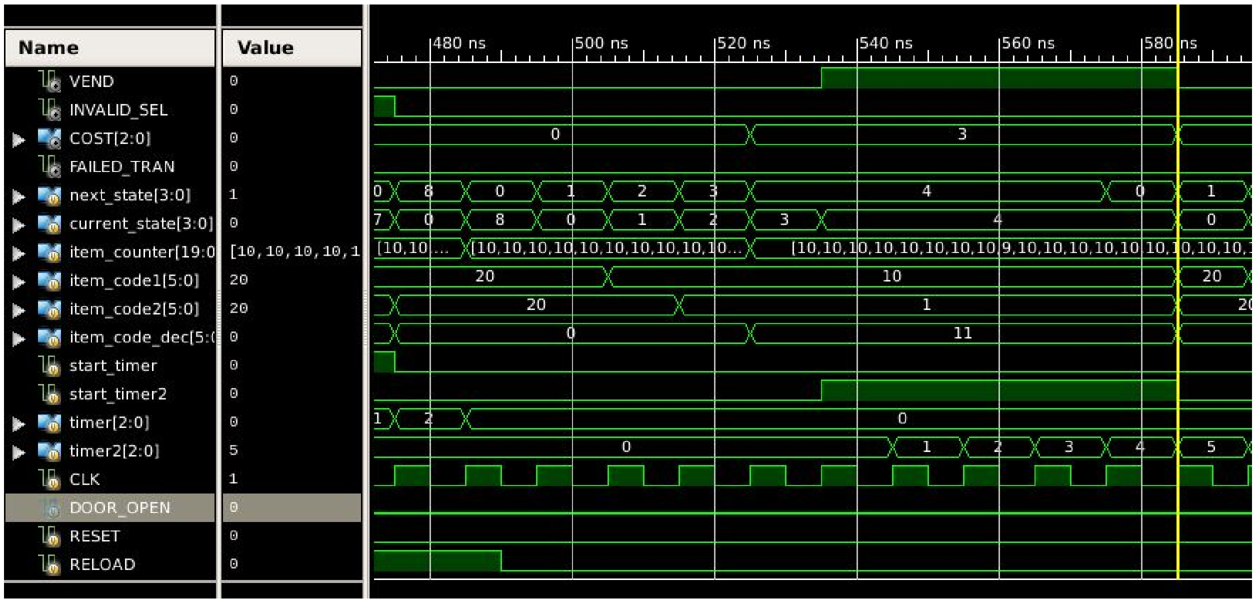


Figure 9: INVALID_SEL because item_code was larger than 19

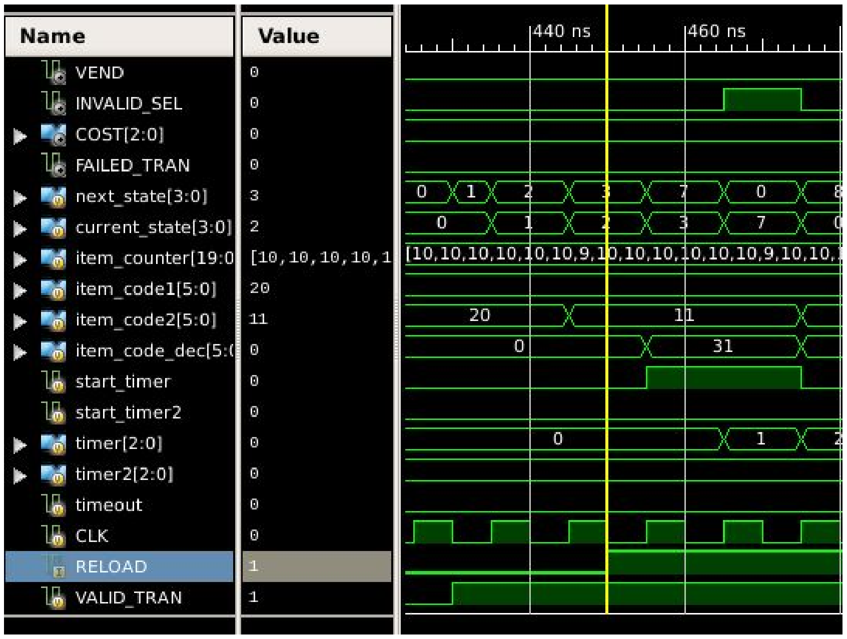
6. This following test case sees if an invalid item code will trigger the INVALID_SEL output. I simulate the behavior by keeping the same inputs as last time, but change ITEM_CODE to 22 which is too large. We can see that when the FSM enters state 3 (CHECK_CODE_CARD), it immediately goes to state 7 (BAD_CODE). During this period, it also sets INVALID_SEL to high so this functionality is as intended.



Name	Value
VEND	0
INVALID_SEL	0
COST[2:0]	0
FAILED_TRAN	0
next_state[3:0]	1
current_state[3:0]	0
item_counter[19:0	[10,10,10,10,1
item_code1[5:0]	20
item_code2[5:0]	20
item_code_dec[5:(	0
start_timer	0
start_timer2	0
timer[2:0]	0
timer2[2:0]	5
CLK	1
DOOR_OPEN	0
RESET	0
RELOAD	0

Figure 10: timeout for DOOR_WAIT_OPEN

7. Then I tested timeout in the DOOR_WAIT_OPEN state. This state follows the CHECK_CODE_CARD state and represents a successful transaction and item ready state. I kept all the inputs the same as last time, and changed ITEM_CODE to be 0001 so that way the final code will be 11. The current_state remains at 4 (DOOR_WAIT_OPEN) for 5 cycles before returning to 0 (IDLE). The VEND signal remains high and this is correct.



Name	Value
VEND	0
INVALID_SEL	0
COST[2:0]	0
FAILED_TRAN	0
next_state[3:0]	3
current_state[3:0]	2
item_counter[19:0	[10,10,10,10,1
item_code1[5:0]	20
item_code2[5:0]	11
item_code_dec[5:(	0
start_timer	0
start_timer2	0
timer[2:0]	0
timer2[2:0]	0
timeout	0
CLK	0
RELOAD	1
VALID_TRAN	1

Figure 11: RELOAD goes high when not in IDLE state

8. This completes our tests through the state diagram, and now I move on to test miscellaneous things. In this simple test, I made RELOAD go high while the current state was not IDLE to see if next_state would become 0 (IDLE) and it did not.

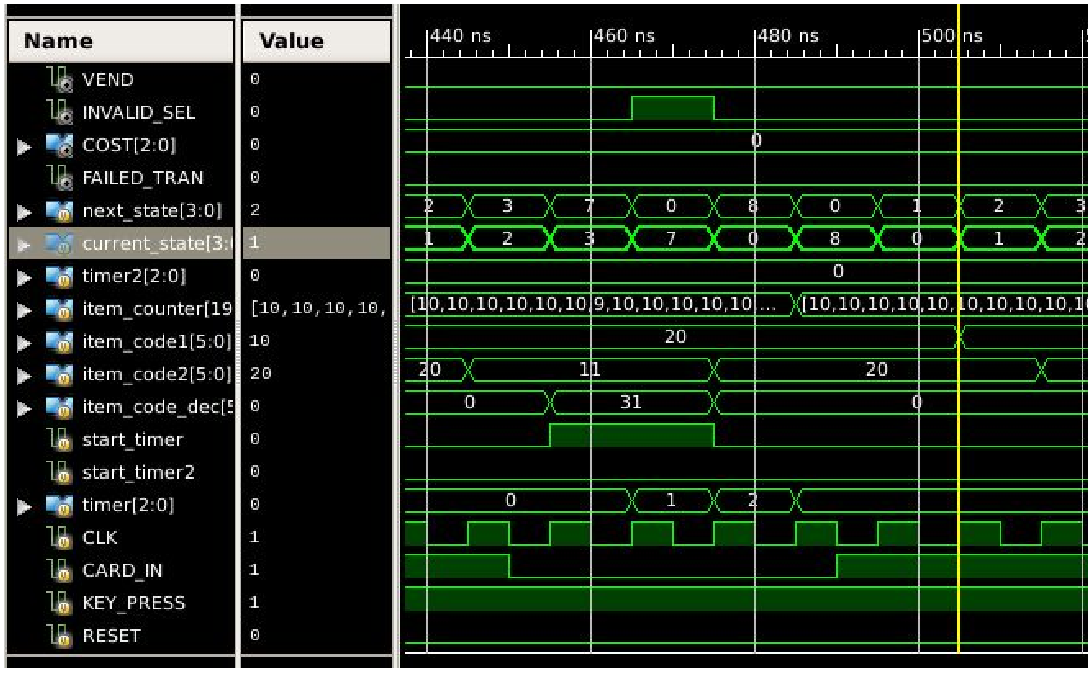


Figure 12: CARD_IN goes high during RELOAD

9. Subsequently, I wanted to test the other requirement which was that the machine should not start a transaction if CARD_IN goes high during RELOAD. To create this behavior, I made CARD_IN equal to 1 while current_state is equal to 8 (RLD). The signal goes high in the middle of the state and remains high, however, the state machine still goes to state 0 (IDLE) first before starting the transaction and moving to GET_CODE1.

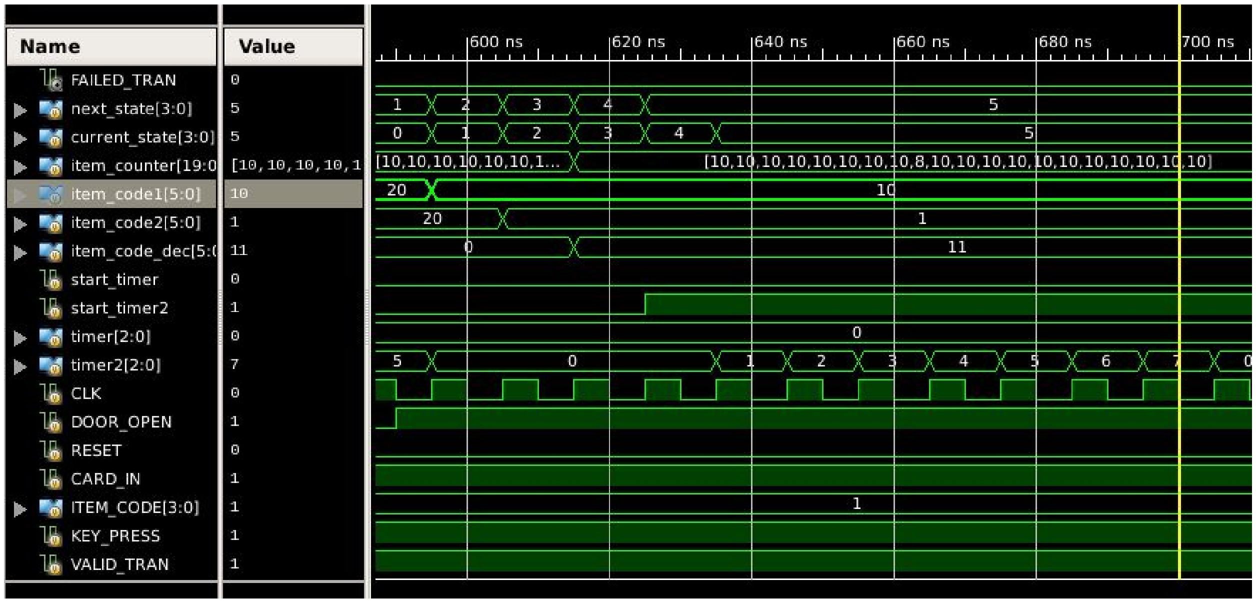


Figure 13: DOOR_OPEN never goes low

10. Then, I simulated the behavior of a vending machine with its door stuck open. Keeping all signals as before, the only modification I make it to keep DOOR_OPEN as 1. After

state 4 (WAIT_DOOR_OPEN), the signal remains high. From timer2, we can observe that the vending machine is stuck in state 5 (WAIT_DOOR_CLOSE) for more than 5 cycles because the signal for DOOR_OPEN has not gone low yet. This is expected.

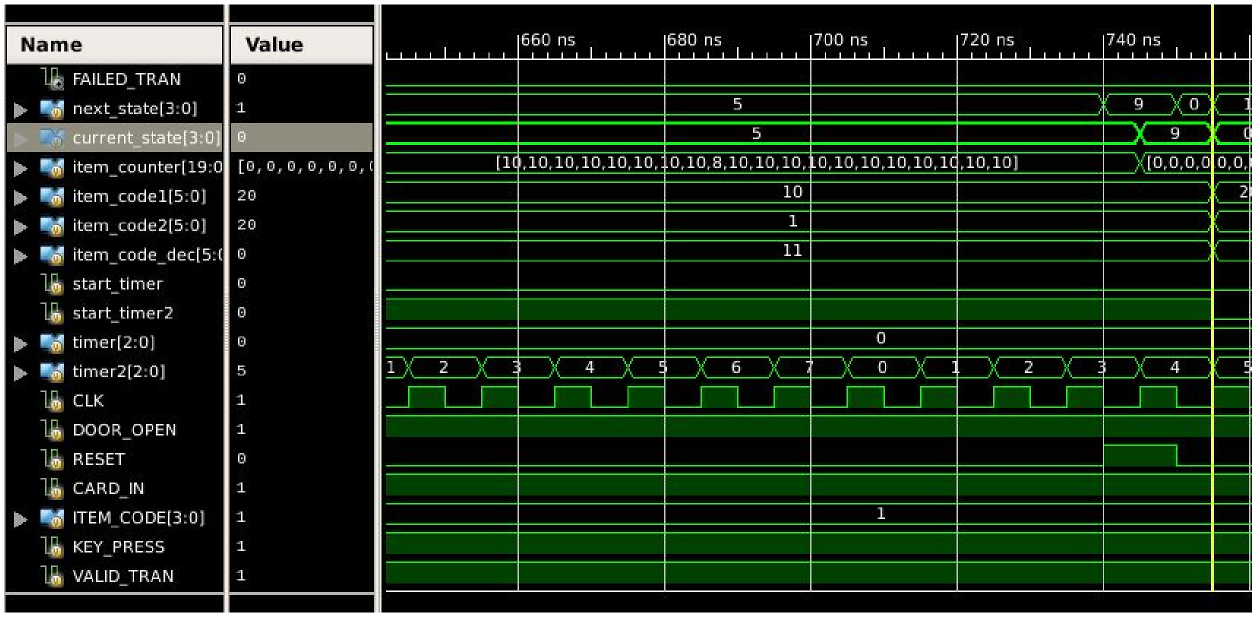


Figure 14: RESET while WAIT_DOOR_CLOSE

11. Using the same test case as before, I only tweaked one signal: RESET. The RESET signal should be able to override any other signal in any state. So I simulated a machine with its door being stuck open and then being reset. At 740ns, we can see that as soon as the RESET signal goes high, the next_state changes to 9 (RST) and then current_state follows it. In this state, all the item_counters go from 10 to 0 and there are no outputs that are high. This means that the machine simulates this successfully.

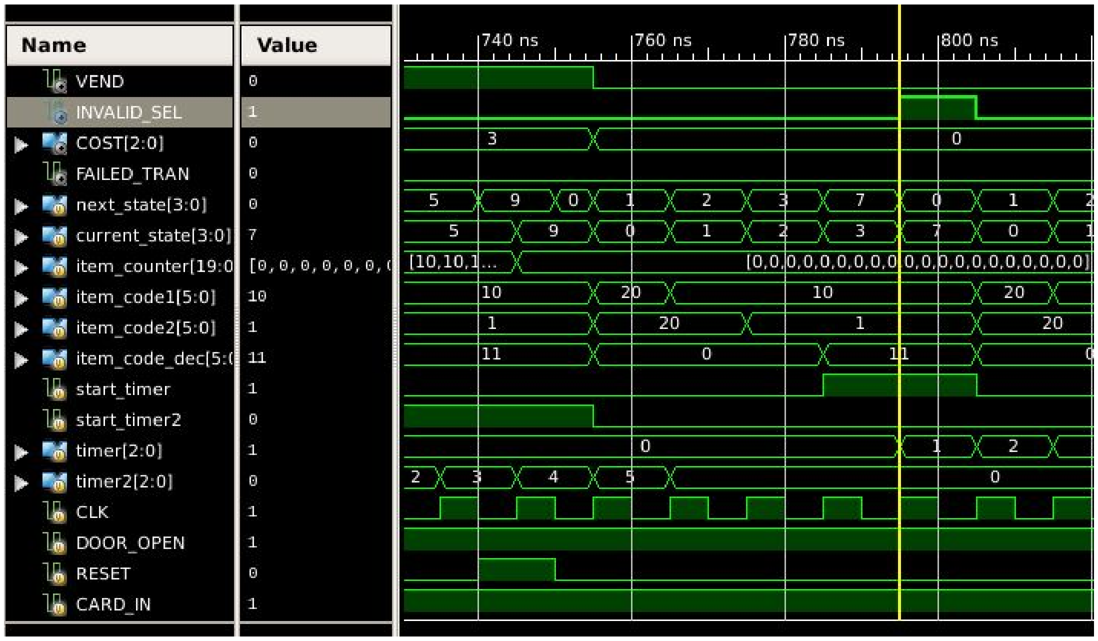
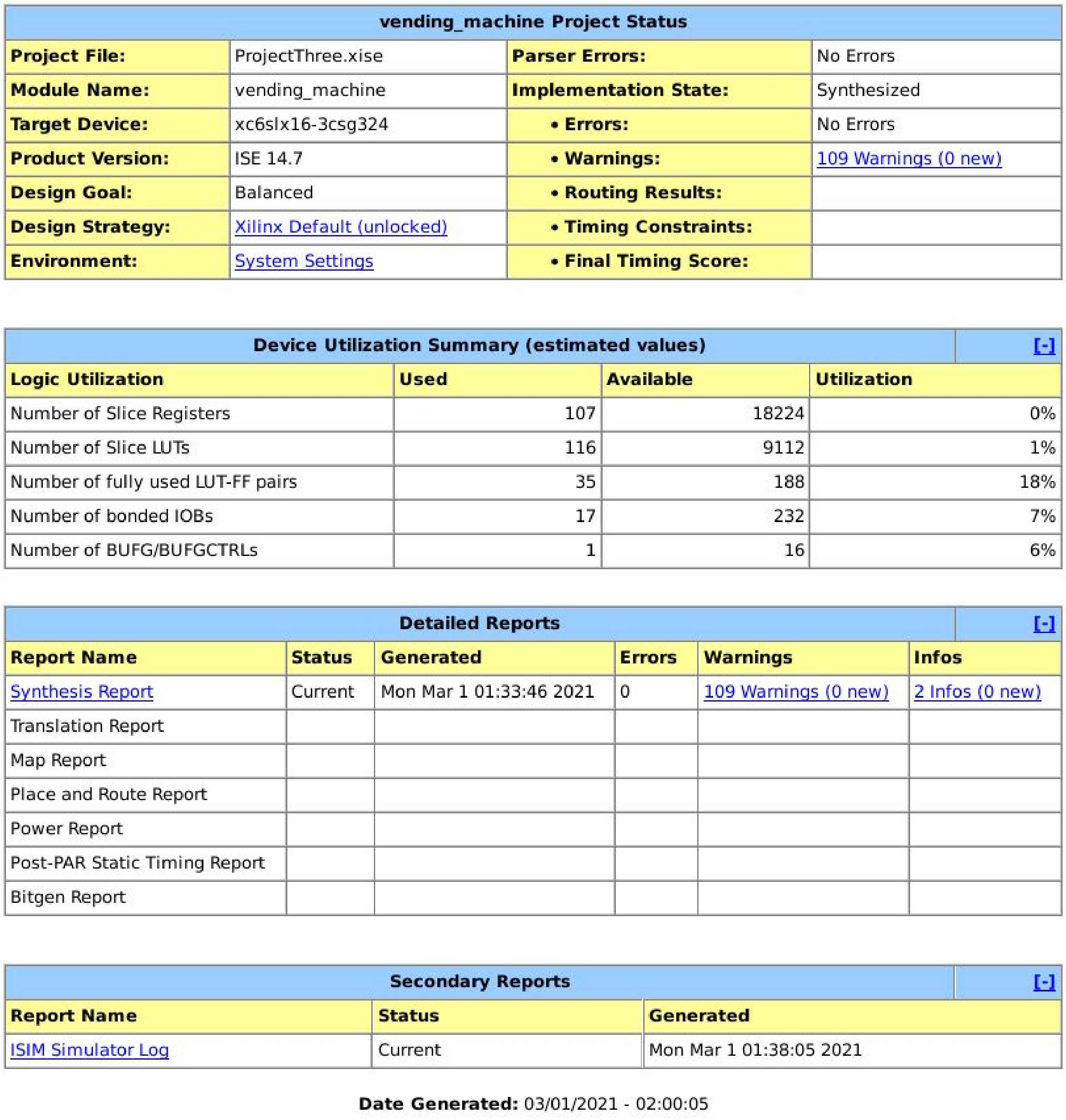


Figure 15: INVALID_SEL when there are 0 snacks

12. Last but not least, since I had reset all the item counters to 0 in the last test run, this was a perfect way to test INVALID_SEL on an invalid selection. I made CARD_IN,KEY_PRESS, VALID_TRAN high and set ITEM_CODE to 0001 to make a code of 11. The current_state flows from 0 (IDLE) to 3 (CHECK_CODE_CARD) smoothly and then changes to 7 (BAD_CODE). This means that the machine handles the case of empty snacks correctly.



vending_machine Project Status			
Project File:	ProjectThree.xise	**Parser Errors:**	No Errors
Module Name:	vending_machine	**Implementation State:**	Synthesized
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	109 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	107	18224	0%
Number of Slice LUTs	116	9112	1%
Number of fully used LUT-FF pairs	35	188	18%
Number of bonded IOBs	17	232	7%
Number of BUFG/BUFGCTRLs	1	16	6%

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Mon Mar 1 01:33:46 2021	0	109 Warnings (0 new)	2 Infos (0 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	Current	Mon Mar 1 01:38:05 2021

Date Generated: 03/01/2021 - 02:00:05

Figure 16: ISE Design Overview Summary Report

As we can see from the summary report, my design has no errors and is synthesizable. There is an alarming number of warnings however, and all of these are latch warnings. This is because in my code, I don’t have an else statement after all the if statements. I omitted these for readability as the code was becoming very long and complex. I made sure that latches won’t occur by clearing defining what inputs will trigger a state transition and then setting those inputs appropriately in the always block that controls outputs. Additionally, in the same always block, I traced through each state and checked that each state will set the correct flags and outputs before transitioning.

Synthesis and Implementation Report

Parts of the Synthesis and Map Report are attached at the end of this report. From the ‘Design Summary’ piece of the Synthesis Report, we can get an overview of the components that the vending machine uses. About half of the components are registers and the majority of the other half are flip-flops. From the Map Report, there are no errors however there are errors about sourcing a MUX by a combinatorial pin. I suspect that this is due to creating flags that trigger the next_state always block. However, I ensure that the flags are set or disabled during very specific moments in the states so that they won’t randomly be set or unset.

Conclusion

In summary, I designed an FSM with 10 states to simulate a vending machine. The vending machine is most complicated in its requirement of waiting 5 cycles for a specific input before returning to the IDLE state. My design is based on a Moore Machine and expands on the turnstile example in the project spec. To implement this, I used always blocks to control specific events and then within each always block, I made use of case statements and if-else statements to determine or calculate the next state. The largest difficulty I had was understanding what inputs were triggered where. It was difficult keeping track of when certain flags were set, especially while considering when inputs would stabilize. Another issue I ran into was implementing the counter to determine the *timeout* flag. I dealt with this by experimenting with a variety of design choices, like having a reset_timer and start_timer button or checking the current state and value of start_timer, before finally settling on alternating between 2 timers. My TA was very helpful in explaining what we needed to do and answering our questions, which I really appreciate. My only feedback would be to see an example of an FSM that depends on a time element.

Synthesis Report

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

Top Level Output File Name          : vending_machine.ngc

Primitive and Black Box Usage:
```

```
------------------------------
# BELS                          : 120
#       INV                     : 4
#       LUT2                    : 10
#       LUT3                    : 7
#       LUT4                    : 16
#       LUT5                    : 16
#       LUT6                    : 63
#       MUXF7                   : 4
# FlipFlops/Latches             : 113
#       FD                      : 4
#       FDR                     : 6
#       LD                      : 99
#       LDC                     : 2
#       LDP                     : 2
# Clock Buffers                 : 1
#       BUFGP                   : 1
# IO Buffers                    : 16
#       IBUF                    : 10
#       OBUF                    : 6

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

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:            107  out of  18224     0%
 Number of Slice LUTs:                 116  out of   9112     1%
      Number used as Logic:                 116  out of   9112      1%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:        188
  Number with an unused Flip Flop:  81  out of       188     43%
  Number with an unused LUT:        72  out of       188     38%
  Number of fully used LUT-FF pairs:        35  out of       188     18%
  Number of unique control sets:    34

IO Utilization:
 Number of IOs:                        17
 Number of bonded IOBs:                17  out of       232     7%
      IOB Flip Flops/Latches:               6
```

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

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

  No Partitions were found in this design.

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

Map Report

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

Design Information
------------------
Command Line   : map -intstyle ise -p xc6slx16-csg324-3 -w -logic_opt off -ol
high -t 1 -xt 0 -register_duplication off -r 4 -global_opt off -mt off -ir off
-pr off -lc off -power off -o vending_machine_map.ncd vending_machine.ngd
vending_machine.pcf
Target Device  : xc6slx16
Target Package : csg324
Target Speed   : -3
Mapper Version : spartan6 -- $Revision: 1.55 $
Mapped Date  : Mon Mar  1 02:16:41 2021

Design Summary
--------------
Number of errors:       0
Number of warnings:   31
Slice Logic Utilization:
  Number of Slice Registers:                 107 out of  18,224      1%
        Number used as Flip Flops:           10
        Number used as Latches:              97
        Number used as Latch-thrus:                0
        Number used as AND/OR logics:              0
  Number of Slice LUTs:                      98 out of   9,112        1%
        Number used as logic:                98 out of   9,112        1%
        Number using O6 output only:         80
        Number using O5 output only:         0
        Number using O5 and O6:              18
        Number used as ROM:                  0
        Number used as Memory:                     0 out of   2,176           0%
```

```
Slice Logic Distribution:
  Number of occupied Slices:                 51 out of   2,278        2%
  Number of MUXCYs used:                     0 out of   4,556         0%
  Number of LUT Flip Flop pairs used:        155
        Number with an unused Flip Flop:            48 out of        155   30%
        Number with an unused LUT:                  57 out of        155   36%
        Number of fully used LUT-FF pairs:          50 out of        155   32%
        Number of unique control sets:              30
        Number of slice register sites lost
        to control set restrictions:         133 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:                     17 out of        232     7%
        IOB Latches:                         6

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:                      6 out of         248      2%
        Number used as OLOGIC2s:                    6
        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:         3.24

Peak Memory Usage:  673 MB
Total REAL time to MAP completion:  23 secs
Total CPU time to MAP completion:   19 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:Security:42 - Your software subscription period has lapsed. Your current
version of Xilinx tools will continue to function, but you no longer qualify for
Xilinx software updates or new releases.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
   current_state[3]_RELOAD_Select_41_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
   current_state[3]_GND_66_o_Mux_239_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
```

```
  current_state[3]_GND_10_o_Mux_127_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_6_o_Mux_119_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_98_o_Mux_303_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_104_o_Mux_315_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net Mram__n05188 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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net Mram__n05186 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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  Mmux_current_state[3]_next_state[3]_Select_44_o12 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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net Mram__n05183 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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net Mram__n05181 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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_93_o_Mux_293_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_58_o_Mux_223_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_54_o_Mux_215_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
```

```
  current_state[3]_GND_50_o_Mux_207_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_78_o_Mux_263_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_18_o_Mux_143_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_74_o_Mux_255_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_92_o_Mux_291_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_96_o_Mux_299_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_62_o_Mux_231_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_38_o_Mux_183_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_46_o_Mux_199_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_42_o_Mux_191_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_70_o_Mux_247_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
```

```
  current_state[3]_GND_26_o_Mux_159_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_34_o_Mux_175_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_30_o_Mux_167_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_22_o_Mux_151_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_82_o_Mux_271_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.
WARNING:PhysDesignRules:372 - Gated clock. Clock net
  current_state[3]_GND_14_o_Mux_135_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:Security:54 - 'xc6slx16' is a WebPack part.
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
---------------------------------

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

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

+----------------------------------------------------------------------------------------------------------------------------
--------------------------+
| IOB Name                         | Type             | Direction | IO Standard        | Diff  | Drive    | Slew |
Reg (s)          | Resistor | IOB          |
|                           |                  |          |                    | Term  | Strength | Rate |
|          | Delay |
+----------------------------------------------------------------------------------------------------------------------------
--------------------------+
| CARD_IN                          | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| CLK                              | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| COST<0>                          | IOB              | OUTPUT       | LVCMOS25            |          | 12
| SLOW | OLATCH     |          |          |
| COST<1>                          | IOB              | OUTPUT       | LVCMOS25            |          | 12
| SLOW | OLATCH     |          |          |
| COST<2>                          | IOB              | OUTPUT       | LVCMOS25            |          | 12
| SLOW | OLATCH     |          |          |
| DOOR_OPEN                        | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| FAILED_TRAN                      | IOB              | OUTPUT       | LVCMOS25            |          | 12
| SLOW | OLATCH     |          |          |
| INVALID_SEL                      | IOB              | OUTPUT       | LVCMOS25            |          | 12
| SLOW | OLATCH     |          |          |
| ITEM_CODE<0>                               | IOB              | INPUT          | LVCMOS25            |
|          |          |          |          |          |
| ITEM_CODE<1>                               | IOB              | INPUT          | LVCMOS25            |
|          |          |          |          |          |
| ITEM_CODE<2>                               | IOB              | INPUT          | LVCMOS25            |
|          |          |          |          |          |
| ITEM_CODE<3>                               | IOB              | INPUT          | LVCMOS25            |
|          |          |          |          |          |
| KEY_PRESS                        | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| RELOAD                           | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| RESET                            | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| VALID_TRAN                       | IOB              | INPUT          | LVCMOS25            |          |
|          |          |          |          |
| VEND                               | IOB               | OUTPUT       | LVCMOS25             |           | 12
| SLOW | OLATCH     |           |           |
+--------------------------------------------------------------------------------------------------------------------------
--------------------------+

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.
```