Lab 4:

Finite State Machine Design:

Vending Machine

CS M152A Fall 2020, Majid Sarrafzadeh

Lab 6: Rajas Mhaskar

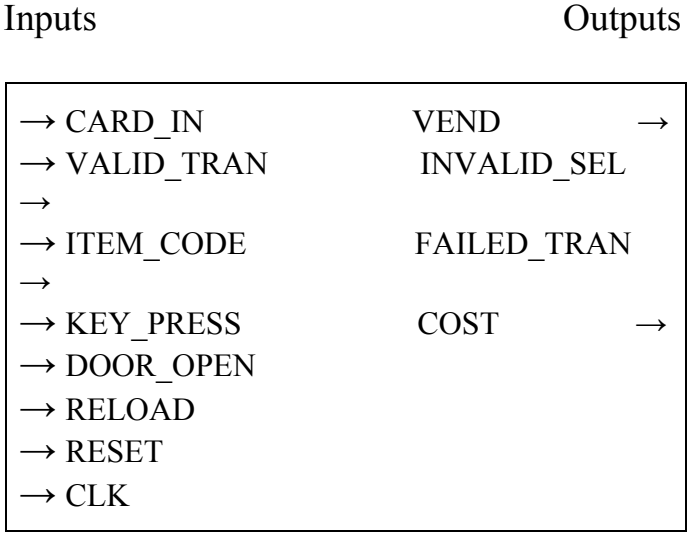
Kenta Hayakawa

12/11/2020

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.

The vending machine for this design has 20 different snacks for sale, each with a maximum of 10 items in stock. The customer can only buy one item at a time, and only accepts card payment. The machine has a total of 8 inputs and 4 outputs, as shown in the high-level of the machine below:



The inputs represent certain actions the user can do (except the CLK signal), and the outputs represent certain signals the machine will trigger depending on the input. For example, the VEND output signal will only be triggered high after the previous transaction steps have been deemed valid. Below are the general descriptions of each of the inputs and outputs:

Inputs

-**CLK**: 1 bit system clock, where the period is 10ns.

-**RESET**: 1 bit signal to set all item counts to 0 and outputs to 0 (empties the machine).

-**RELOAD**: 1 bit signal to set all item counts to the max value of 10 (restocks the machine).

-**CARD_IN**: 1 bit signal to represent the card being inserted.

-**ITEM_CODE**: 4 bit signal that represents one of the item codes from 00 to 19.

-**KEY_PRESS**: 1 bit signal that designates when ITEM_CODE is read.

-**VALID_TRAN**: 1 bit signal that goes high when the previous transactions are all deemed valid, and the item will be dispensed shortly.

-**DOOR_OPEN**: 1 bit signal that represents the door opening.

Outputs

-**VEND**: 1 bit signal that is triggered when transaction is deemed valid. Goes back to low when door opens or when door does not open for 5 clock cycles after validation.

-**INVALID_SEL**: 1 bit signal that is triggered when

1) Item code is invalid (ex: 26)
2) Second inputs is not entered for 5 cycles after the first.
3) There are no more of the selected item remaining in the machine

-**COST**: 3-bit signal that represents the cost of the item, from 0 to 6. Triggered once a valid item selection is entered.

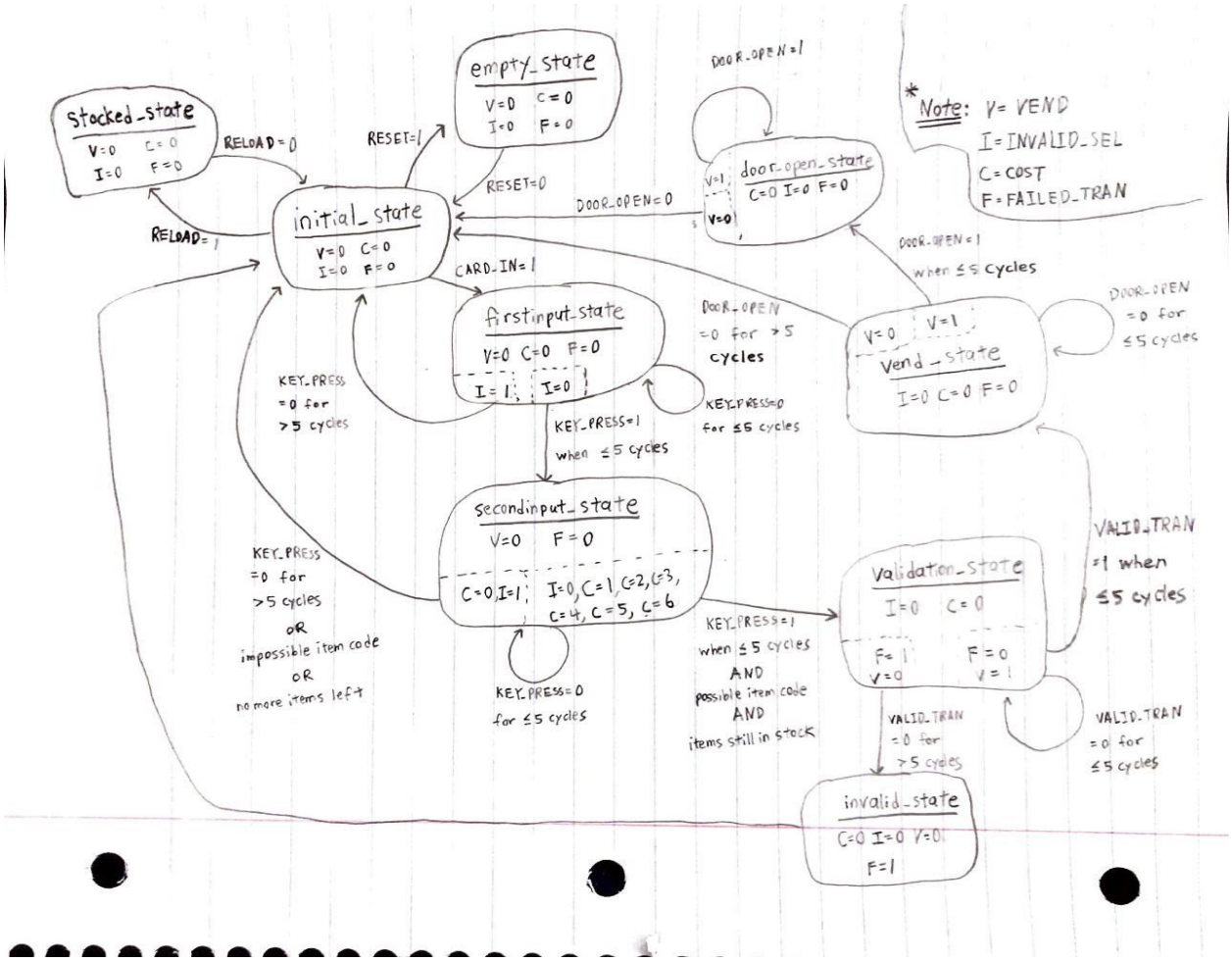
-**FAILED_TRAN**: 1 bit signal that goes high if the VALID_TRAN signal does not go high after 5 clock cycles after entering item code.

Each of the 20 items have a specific item code associated with it (from 00 to 19), and each item has a specific cost to buy them, given in the table below:

ITEM_CODE	COST
00, 01, 02, 03	1
04, 05, 06, 07	2
08, 09, 10, 11	3
12, 13, 14, 15	4
16, 17	5
18, 19	6

Design Description

The vending machine has a total of 9 states, each determined by the previous inputs and current states. Below is the FSM diagram of this machine:



The description for each of the states is as follows:

1) **initial_state**: Also known as the idle state. The machine waits for a new transaction to begin with the card inserting. All outputs are set to 0 at this point, since nothing is happening.

2) **empty_state**: Sets all item counters to 0, effectively emptying the machine of any items. No new transaction can be started at this state.

3) **stocked_state**: Sets all item counters to 10, effectively restocking the machine. No new transaction can be started at this state.

4) **firstinput_state**: Card is inserted, and the machine waits for an input. The input can only be read while the KEY_PRESS signal is high.

5) **secondinput_state**: The machine takes in the second input. If it is inputted more than 5 clock cycles after the first input, or if the item code inputted is none of the available item codes, it is deemed invalid, and goes back to the initial state.

6) **validation_state**: The machine waits for the VALID_TRAN signal to go high, representing that the inputted code and the card is valid for transaction. If the signal does not go high within 5 clock cycles, it is considered a failed transaction, and goes to the invalid state.

7) **vend_state**: Decrements the selected item's counter, and waits for the door to open for pick-up. If the DOOR_OPEN signal does not come within 5 clock cycles, the transaction is deemed invalid, and goes back to initial state.

8) **door_open_state**: The door opens, and is in this state indefinitely until the DOOR_OPEN signal goes back to 0 (meaning the item was picked up and

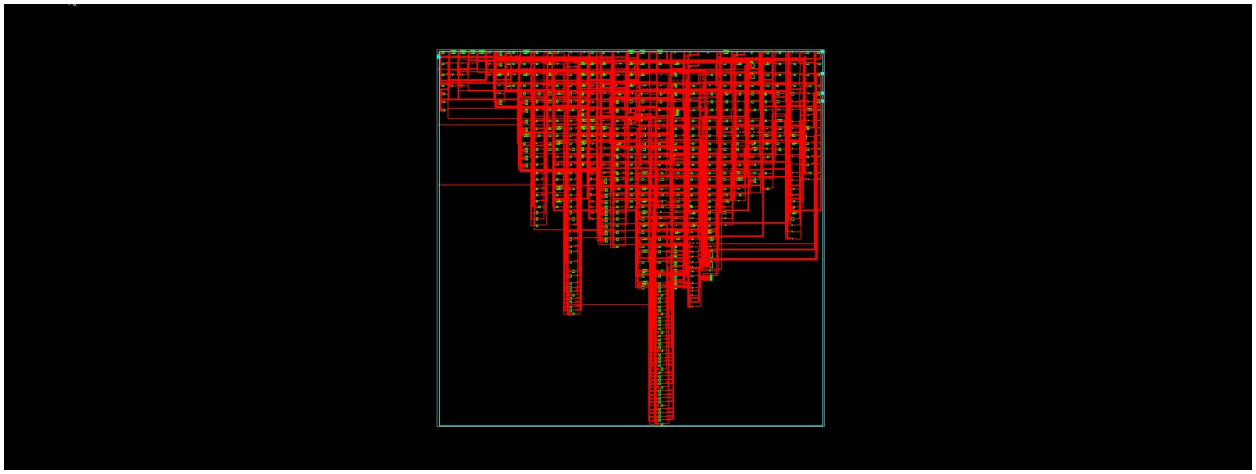
the door has been closed). The machine goes back to the initial state if the item has been successfully picked up, and waits for a new transaction.

9) **invalid_state**: The machine has deemed the card payment invalid, and the invalid bit is set to high for one clock cycle. Then, it goes back to initial state to wait for a new transaction.

For this project, no submodules were used, and all the logic was incorporated into the top module. A total of 4 always blocks were used in this top module:

- always@(posedge CLK): current state transitions to the next state
- always@(negedge CLK): counts the clock cycles on every negative edge. Necessary because many of the states go back to the initial state, or are deemed invalid after a wait of 5 clock cycles after the previous state or input.
- always@(*): There are 2 always@(*) blocks used. The first one is used to determine under what conditions the states will change, and also the 5 clock cycle condition that sends a state back to the initial state (invalid transactions). The second one is used to design the outputs for each state.

The schematic for the machine is given below. Since it is very complex, it shows that the implementation of the machine was not very modular, since use of more submodules could have made it look more organized. This observation is true, since the machine really only used one top module, and no submodules.



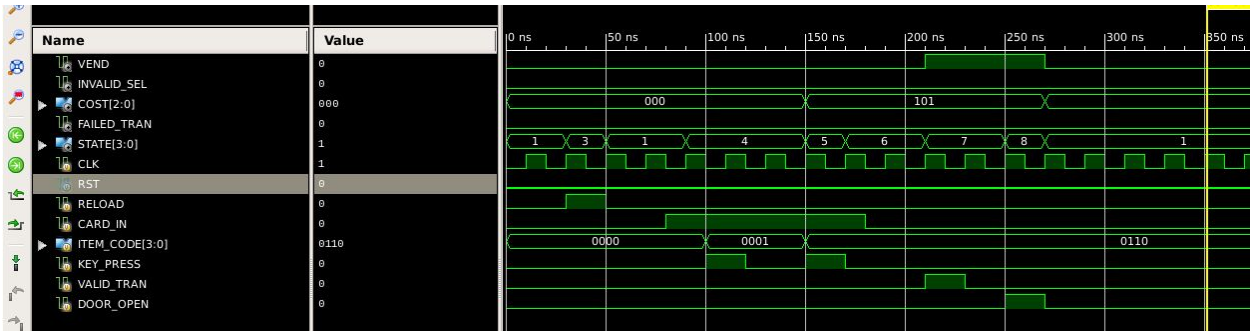
Simulation Documentation

The testbench for this project included all 8 possible inputs: CLK, RELOAD, RESET, CARD_IN, ITEM_CODE, KEY_PRESS, VALID_TRAN, and DOOR_OPEN. A 100hz clock was generated by giving a 10 unit time delay every clock pulse. Note that the simulations show another output, STATE. This is only used for simulation purposes and to show the transitions of the states more easily. The actual source code will not contain this output. The STATE outputs correspond to the actual machine states as follows:

1 = initial_state
2 = reset_state
3 = reload_state
4 = firstinput_state
5 = secondinput_state
6 = validation_state
7 = vend_state
8 = door_open_state
9 = invalid_state

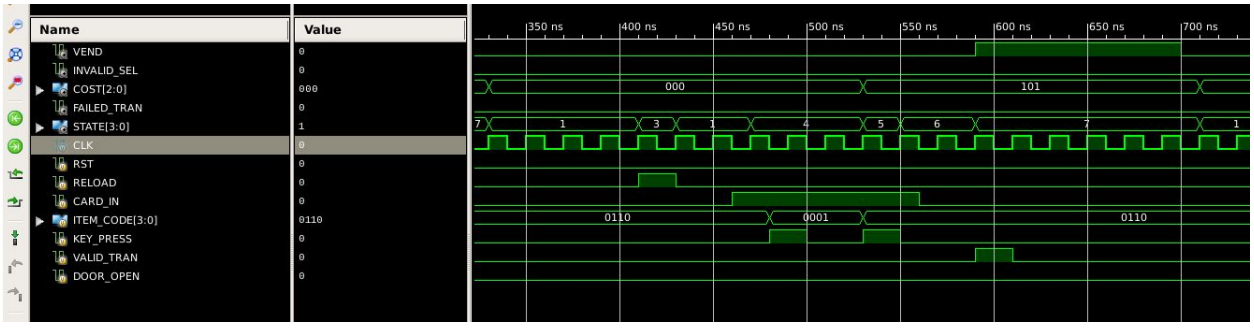
The first simulation below shows the case for when a transaction was a success. Here, a reload signal is triggered, which sets the item counters to the max value of 10. Then, a card is inserted, and a transaction begins, as modeled with the card_in waveform. Then, when the first key_press signal goes high, the item code changes to 0001. This is equivalent to the user pressing the button 1 as his/her first

input. The key_press goes low again, because the user is not hitting the second input yet. Before reaching 5 clock cycles since the first input was made, a second input 0110 was entered, meaning a 6 was selected. This means that the user wants item 16, which has a cost of 5. We can tell this selection was valid, since no invalid_sel signal is triggered. In addition, the cost output now displays 101, meaning 5, as the second input is entered. Less than 5 cycles later, a valid_tran signal is successfully triggered, which causes the vend output signal to be triggered successfully. Less than 5 cycles after the validation, the door_open signal is triggered, meaning the door for the machine has opened. Around 2 cycles later, the door_open goes to 0, making the vend signal go to 0, meaning the item has been picked and the door has closed, and goes back to the initial state shortly after to take in another transaction. The state outputs go in the order of 1,3,1,4,5,6,8,1, which is exactly what the order of this successful case is.



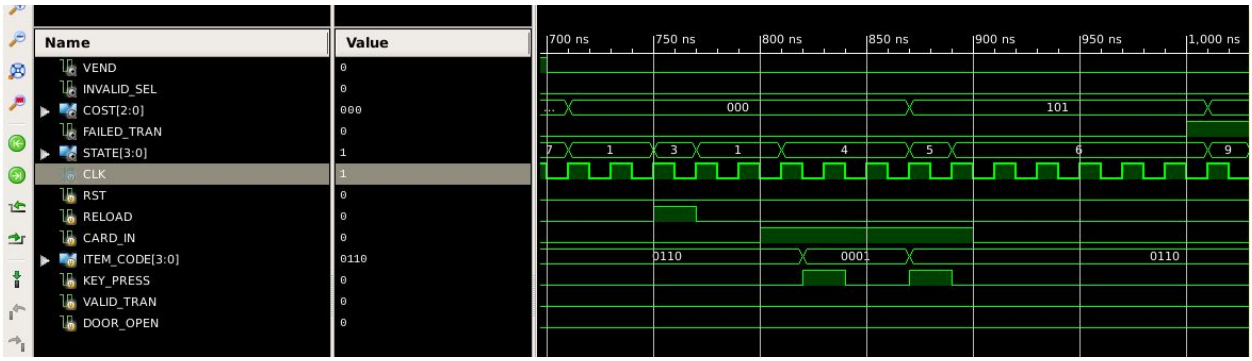
-Case 1: Success case

The second simulation below, which continues from the previous simulation, shows an edge case, where the door never opens for the user after successful validation. Here, we can see that up until the validation state, the transaction is successful, as it shows the same waveforms as the success case. The input is the same 16. However, after the valid_tran signal is thrown, the door_open signal never comes, so the machine reverts to its initial state. This can be seen with the vend signal automatically going back to 0 after 5 clock cycles, as well as the cost display going back to 0 at that same time, indicating that the machine has gone back to its initial state. The state outputs go in the order of 1,3,1,4,5,6,7,1, which is exactly what the order of this successful case is (no state 8 like the previous case, because the machine never goes to door_open_state).



-Door never opens case

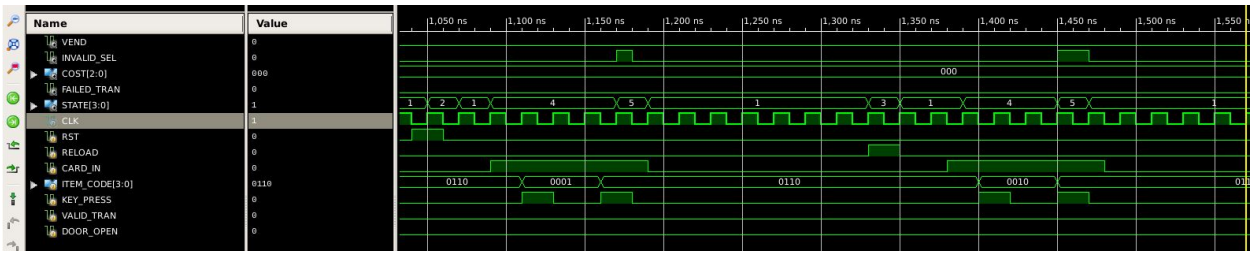
The third simulation that continues from the previous simulation shows another edge case, where the valid_tran signal never comes. The user does select the item properly, since no invalid_sel signal is triggered. However, the valid_tran signal is never triggered, so 5 cycles after the second input, a failed_tran signal is triggered, and takes the machine back to the initial state. In the code, an invalid state is set up specifically for this case, and this state triggers the failed_tran signal. The state outputs go in the order of 1,3,1,4,5,6,9 which is exactly what the order of this successful case is (a state 9 is introduced here because the machine goes to the invalid_state).



-No valid signal case

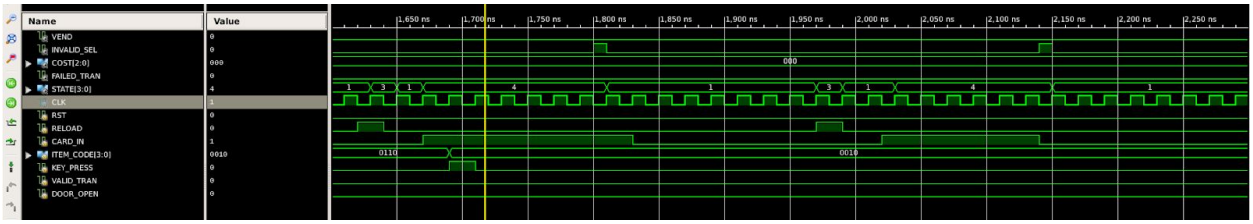
The fourth simulation below shows the next two edge cases for convenience. Since this is continuing from the last simulation, the cost display is 0. The first half of the simulation shows the case when the user selects an item with no more snacks left. A reset signal is initially triggered, which brings all the item counters to 0. The

user then inputs the first and second inputs successfully. However, since the reset signal was triggered earlier, there are no more of that selected item left in the machine, so an invalid_sel signal is triggered. The cost display also doesn't change from the initial value of 0, because the selection was invalid. The second half of the simulation shows the case when the user inputs an invalid item code, in this case 26. We first reload from the last simulation, so that the case of 0 items doesn't interfere with this particular case. We can see that the first key_press signal of the second half has the item code value of 0010, and the second input has the value of 0110, indicating a selection of item 26. This does not exist, so an invalid_sel signal is triggered, and the cost remains 0, similar to the first half of the simulation.
The state outputs go in the order of 1,2,1,4,5 for the first half, then 1,3,1,4,5,1 for the second half, which is exactly what the order of these successful cases are (both of the cases end at state 5 and go back to initial state, because the inputs were both invalid).



-0 items case (first half) and invalid item code case (second half)

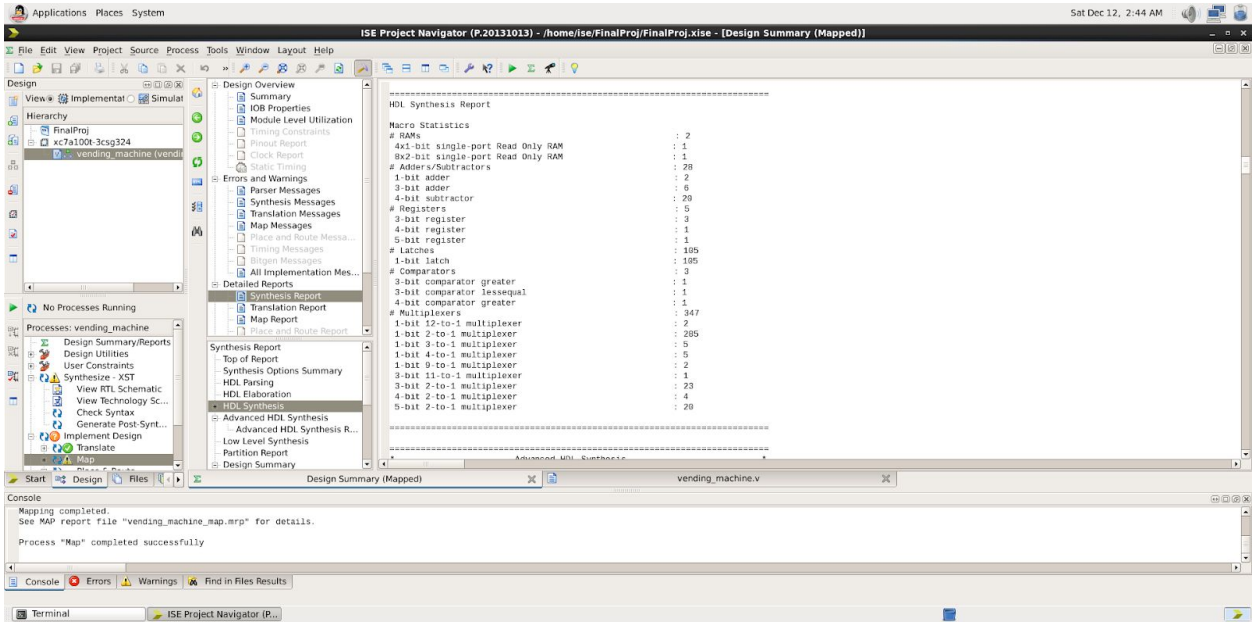
The below simulation illustrates 2 extra edge cases, beyond the minimum requirement of 1 success case and 4 edge cases. The first half of the simulation shows the case when the second input never comes. The first input can be seen to be successfully entered due to the key_press signal going high, with a value of 0010, and no invalid_sel triggers. However, the second key_press signal never comes, indicating that the user did not enter the second digit, so an invalid_sel signal is triggered, and takes the machine back to initial state. The second half of the simulation shows the case when the user enters a card but never selects an item. Here, a card_in signal is triggered, but the no key_press signal ever comes, so after more than 5 cycles, an invalid_sel signal is triggered, indicating end of transaction and revert to idle state. For both cases, the machine ends at state 4 before going back to idle state. Both cases do make it to the firstinput_state, because the card is inserted, but neither make it do the secondinput_state, or state 5. For the first case, this is because the first input is never made, and for the second case, it is because the second input is never made.



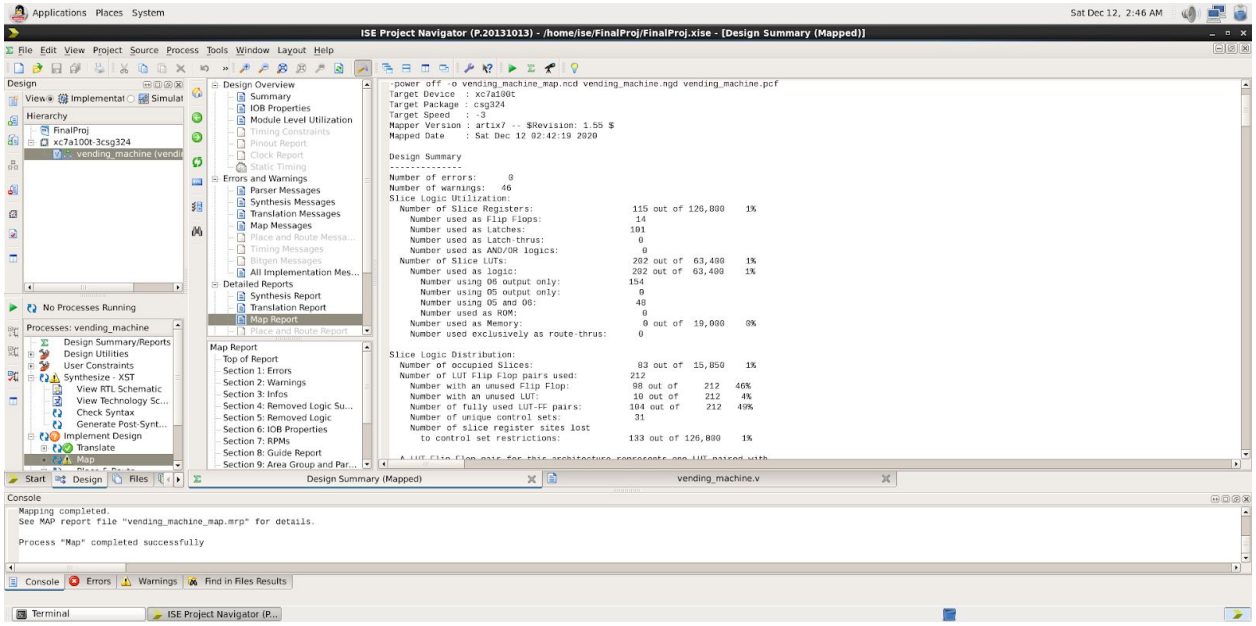
-No second input case (first half) and no first input case (second half)

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. Most of the components used were multiplexers. From the map report, it can be seen that 0 errors were made, which is important, and a lot of slice LUTs and slice registers were used.

Design Summary:



Map Report:



Conclusion

This lab gave me a good experience with coding finite state machines using verilog and xilinx. We designed a vending machine that dispenses an item to the customer, or does not due to an invalid transaction. The machine had 9 possible inputs and 4 possible outputs. Since finite state machines are used everywhere in the real world, I believe having good practice with it is important, and I think I got a solid experience with it with this project and the last project. The one major obstacle I had during this project was the 5 cycle counter. Although it works for most of the cases, there still seems to be a slight discrepancy in timing, due to the

state updating on the positive edge but the clock cycle counter updating on the negative edge. This was necessary, however, because updating both of these factors on the same clock edge will cause a race condition, which will cause much more errors.

Overall, I believe this was a great project to practice coding a FSM, and I hope I can put these skills to good use in potential future projects.