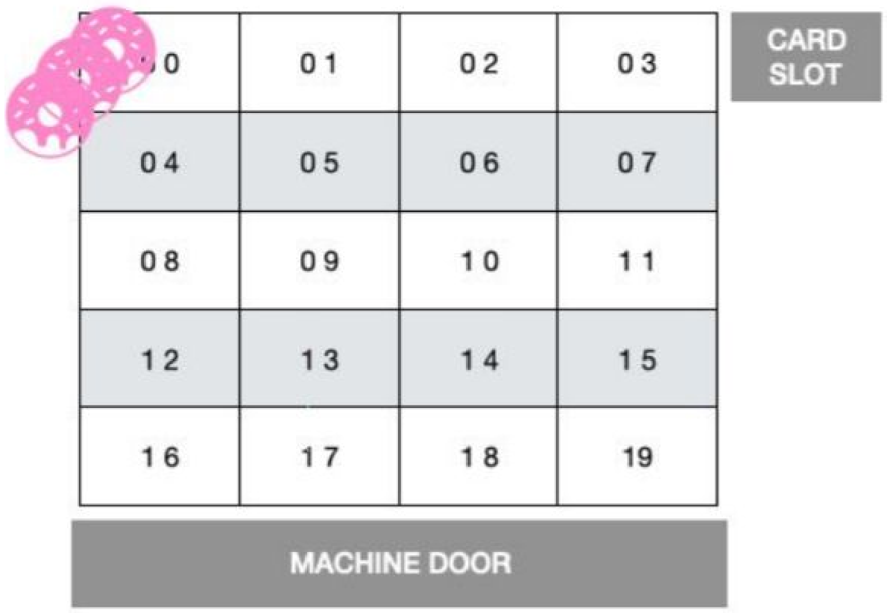
Adam Cole
UID: [redacted]

Project 4 Report

1. Introduction and Requirements

1.1 Introduction

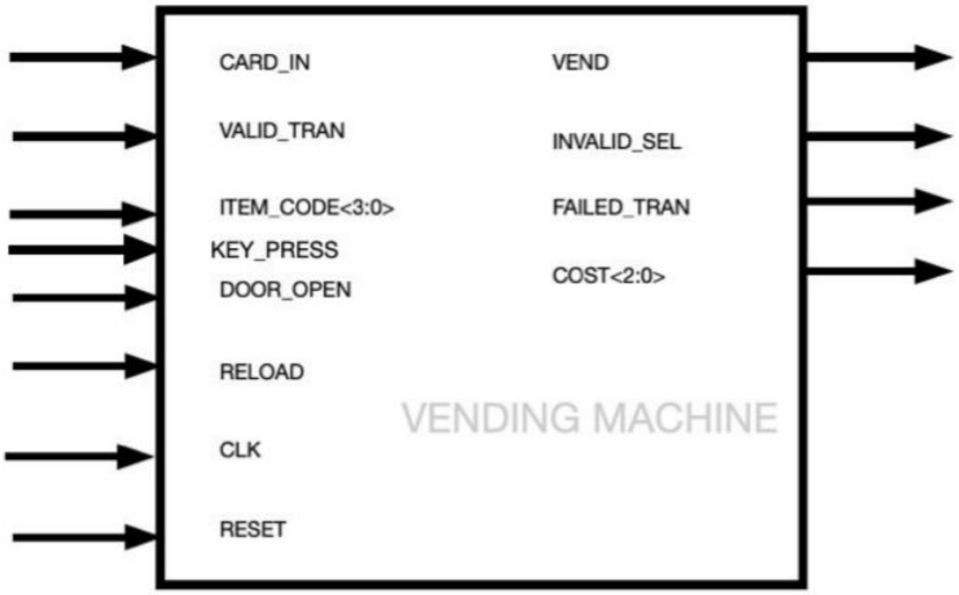
For Project 4, our goal was to design and implement a fully functional vending machine for around our college campus. The vending machine has twenty items for sale, each in a separate slot. Each slot can hold up to ten items. A high level diagram for the vending machine is shown below:



In this example, Donuts are the item in slot `00`. There is a card slot for payment, and no cash will be accepted at our vending machine. After a customer inserts their card, makes a selection, and payment is validated, the machine will vend the item through the machine door. The machine only allows buyers to purchase one item at time, but if the buyer leaves his/her card in the machine, another transaction will automatically start.

1.2 Design Requirements

We implement the logic of the vending machine through a Verilog source code module in Xilinx ISE. The inputs and outputs of our module match the high level schematic shown below:



These inputs and outputs follow the following specification:

INPUT	BITS	BEHAVIOR
CLK	1	System Clock (T = 10ns).
RESET	1	Set all item counters and outputs to 0, go to idle state.
RELOAD	1	Set all item counters to 10.
CARD_IN	1	Stays High when Card remains inserted
ITEM_CODE <3:0>	4	2 digit item code, input one digit at a time. Must be input while the Card is inserted.
KEY_PRESS	1	Transitions to High when the user presses a button and updates ITEM_CODE to be read.
VALID_TRAN	1	Transaction using Card is valid (from Bank)
DOOR_OPEN	1	User opened the machine door (once Vend=1)

OUTPUT	BITS	BEHAVIOR
VEND	1	Set to High once VALID_TRAN is also High. Set to Low once DOOR_OPEN toggles or if the machine door is not opened in 5 clock cycles.
INVALID_SEL	1	Set to High if only 1 digit of ITEM_CODE is entered and there is no second digit after 5 clock cycles, OR the second digit is invalid, OR the item counter is 0.
COST <2:0>	3	Set to 000 by default. Set to the cost of an item once ITEM_CODE is input, and remains at this value until a new transaction begins.
FAILED_TRAN	1	Set to High if VALID_TRAN does not go High after 5 clock cycles.

The machine must start in the idle state. When idle, the machine waits for a new transaction to begin. From the idle state, the machine can be reloaded or a new transaction can begin. When reloaded, all item counters are refilled to 10 and the machine moves back into idle. A new transaction begins when a card is inserted into the machine.

Once a transaction has started, the machine must get the item code from the user. The item code is input one digit at a time. The user has five system clock cycles to input each digit. If either digit exceeds five cycles or the item code is invalid, the vending machine transitions back to idle. An item code can be invalid if the number exceeds 19 or the slot for the item is empty. If the item code is valid, the machine executes the transaction. The cost of the selected item will be displayed before the transaction happens. The cost of items will follow the following chart:

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

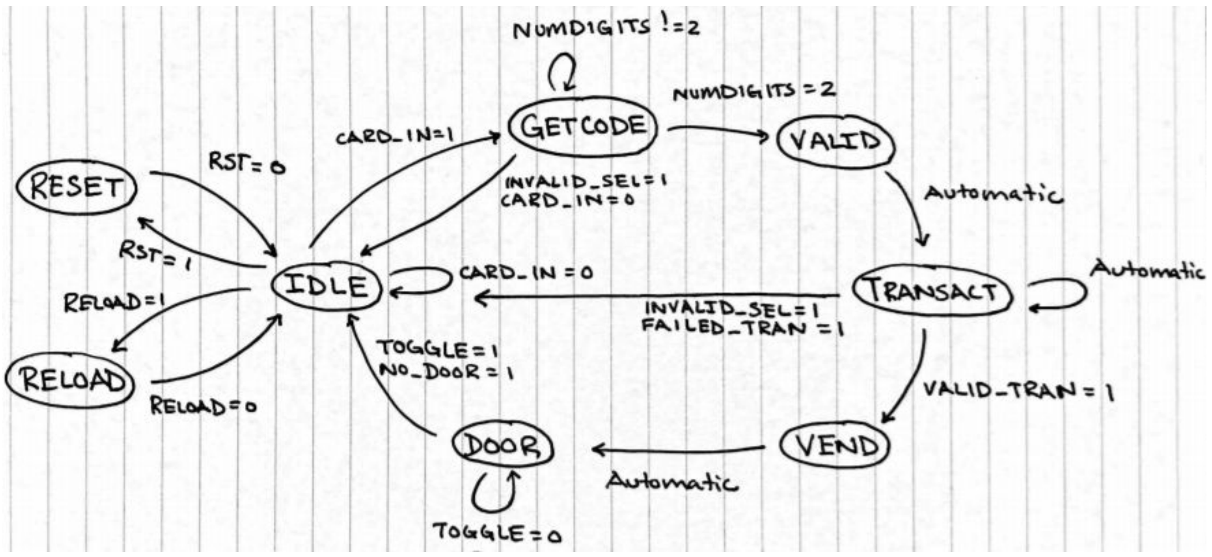
Table 1: ITEM CODE vs COST

Once the transaction has begun, the vending machine waits 5 clock cycles for a valid transaction notification from the bank. If the machine receives the notification, the machine begins vending the item. If no message is received in 5 clock cycles, the transaction fails and the machine moves back into idle. When vending the item, the machine will decrement the item counter and wait 5 clock cycles for the user to open the machine door. Once open, the machine will wait indefinitely until the door is closed and then move into idle. If the door is not opened within 5 clock cycles, the machine will return to idle.

2. Design Description

2.1 Finite State Machine Diagram

In order to implement these requirements in our vending machine design, we created 8 states for our machine. Below is our Finite State Machine Diagram:



We utilized four always blocks to implement our design. The first always block updates the current state of the machine to the next state at each clock cycle. The second always block updates counters at each clock cycle for each of the states. This helps to implement the timing requirements in our machine. The third always block investigates the logic flow behind what to update `NEXT_STATE` to. Finally, we use a fourth always block in order to execute combinatorial logic of the `CURR_STATE`.

My states were encoded in the following order:

STATE	**ENCODING**
RESET_STATE	000
RELOAD_STATE	001
IDLE_STATE	010
TRANSACT_STATE	011
VEND_STATE	100
GETCODE_STATE	101
VALID_STATE	110
DOOR_STATE	111

2.2 Idle State, Reset State, and Reload State

To implement our design, we started with three basic states - idle, reset, and reload. When our module enters the idle state, all outputs are set to 0. If input `CARD_IN = 1`, our machines' `NEXT_STATE` variable is updated to the get-code state. If input `RESET = 1`, our machines' `NEXT_STATE` variable is updated to the reset state. In the reset state, all outputs and variables used in our module are initialized to 0, and the machine moves back into the idle state. If input `RELOAD = 1`, our machines' `NEXT_STATE` variable is updated to the reload state. In the reload state, all of the item counters are refilled to 10 and the machine moves back into the idle state.

2.3 Get-code State and Valid State

Our get-code state is responsible for receiving the customer input while verifying that the input item code is a valid choice. I implement this functionality by creating a `NUMDIGITS` variable to keep track of how many digits have been input. If `NUMDIGITS` is 1 and the input value is 1, then the item value is updated to 10. If `NUMDIGITS` is 1 but the input value is 0, then the item value is updated to 0. Finally, if the item value is neither of those two, the item value is set to 20, because after the second digit we want the value to be invalid either way. When `NUMDIGITS` is 2, the item value is added to the existing item value. This logic only runs when `KEY_PRESS` is 1, and the counter is also reset here in the other always block. If the counter for this state ever reaches 6 - indicating 5 clock cycles have completed, `INVALID_SEL` is set to 1 and the machine moves back to idle state. Once two digits have been entered, `NEXT_STATE` is set to the valid state, which checks the validity of the item code.

Our valid state is responsible for checking the validity of the item value and updating the cost. This happens all in one clock cycle, so the valid state automatically sets `NEXT_STATE` to the transact state. In the combinatorial logic, the valid state checks that the item value is within 00 - 19. If not, `INVALID_SEL` is set to 1. Furthermore, the valid state checks the counter of the correct item value. If it equals 0, `INVALID_SEL` is set to 1. In the same case statement, the correct cost is updated for each item.

2.4 Transact State, Vend State, and Door State

The transact state is one of the simplest states in our design. The state waits for a `VALID_TRAN` input, and when it receives this, `NEXT_STATE` is set to the vend state. The combinatorial logic simply checks if the counter ever reaches 6 - indicating 5 clock cycles have passed - and if it does, sets `FAILED_TRAN` to 1. If `FAILED_TRAN` is 1, the machine did not receive a `VALID_TRAN` input before 5 clock cycles passed. Here, the machine moves back to the idle state.

The vend state is much like the valid state in that the entire state executes in one clock cycle. In this way, `NEXT_STATE` automatically updates to the door state. The vend state sets `VEND` to 1 and decrements the count of the item's counter.

Finally, we reach the door state. The door state is responsible for correctly transitioning the machine back to the idle state within the correct timing. The combinatorial logic for the door state checks if the counter reaches 6 - indicating 5 clock cycles have elapsed. If this happens, flag `NO_DOOR` is set, and the machine returns to the idle state. If

`DOOR_OPEN` is 1, however, the door has been opened. In this case, the counter stops incrementing, and the machine hangs until `DOOR_OPEN` returns to 0, then returns to the idle state.

3. Simulation Documentation

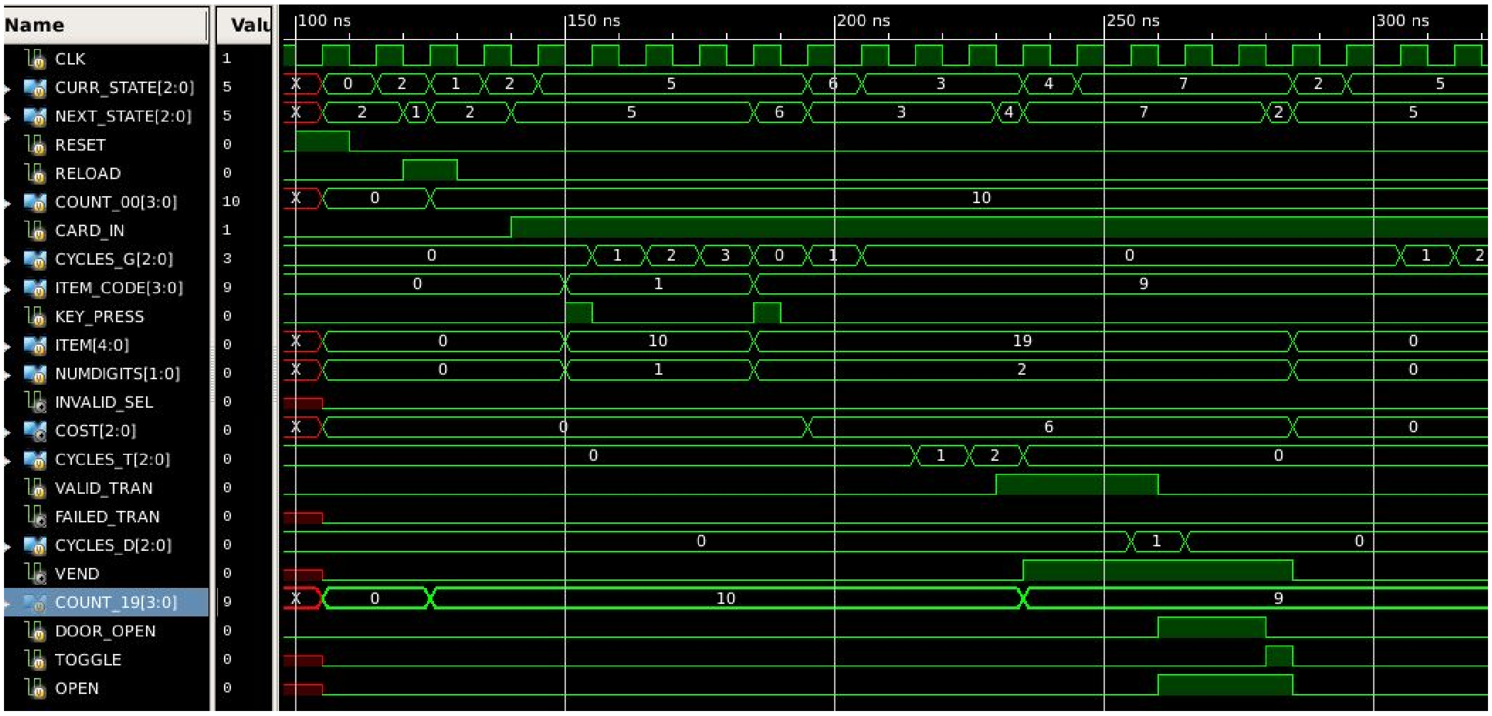
3.1 Testbench

Unable to test every possible input sequence for our vending machine, our testbench verilog file verifies a successful transaction and then investigates four special cases.

First is the special case that a customer leaves his card in the machine after the transaction has completed. In this special case, the machine should go immediately into a new transaction from the idle state. Our next special case is when a user inputs an invalid item code. Here, the user inputs 55, which exceeds the number of items in the machine. In this case, the `INVALID_SEL` bit should be set to 1 and the machine should return to idle. In our third special case, the user never opens the door after purchasing an item. In this case, the machine should wait 5 clock cycles and then return to idle, still decrementing the item counter since the snack was vended. In our last special case, the user removes his credit card while inputting the item code. In this case, our machine should return to the idle state and wait for the card to be re-inserted.

3.2 Successful Transaction and Customer leaves card in Machine

Our first test validates the vending machine process through a successful transaction and combines this with our first special case test. Here, `CARD_IN` remains 1 throughout the transaction, and after the machine returns to idle. In this case, our machine should immediately begin a second transaction. The waveform associated with our first test is shown below:

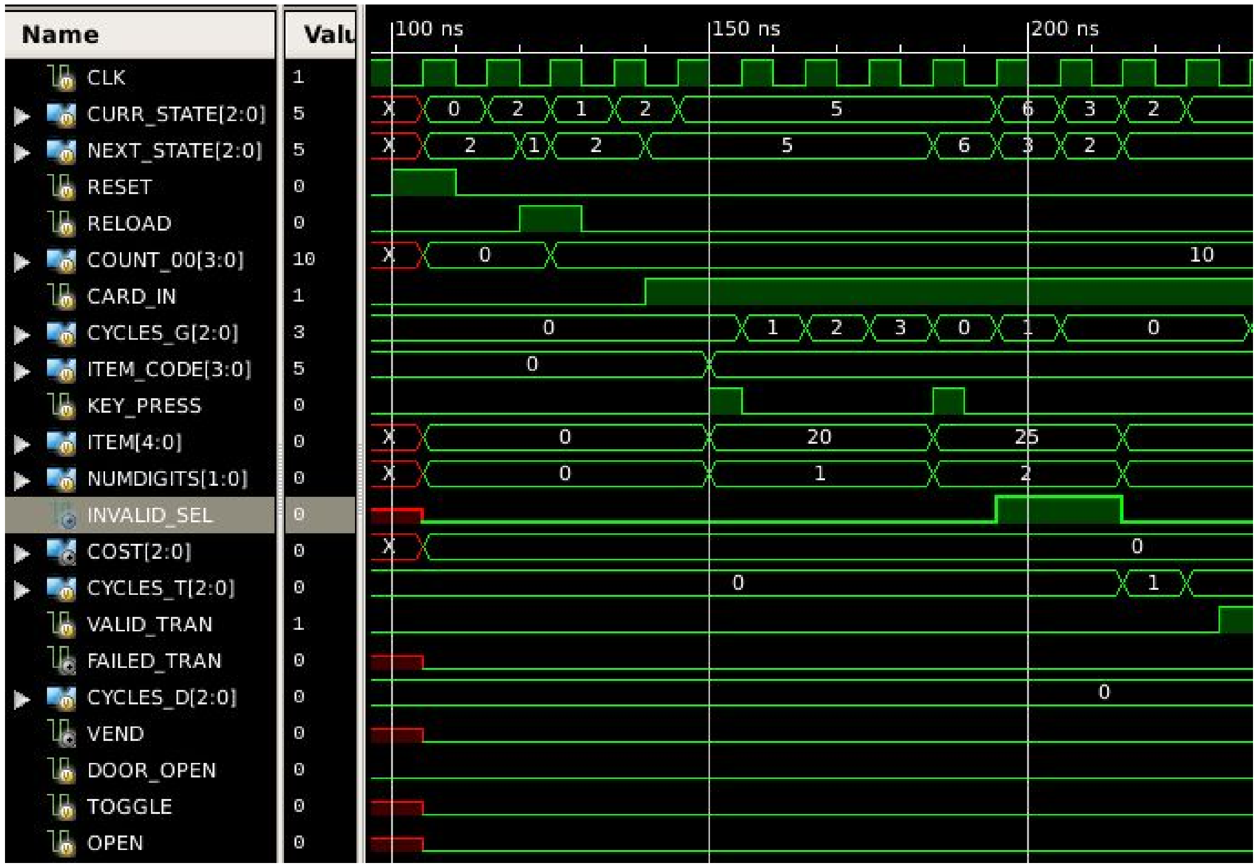


Here, you can see the successful first test. I aligned the waves sequentially top down, so we can easily step through the successful vending process. First, we can see the clock toggling with a period of 10ns as per the spec. Next, we see the reset state correctly sets all fields to 0. At reload, both counters for items 00 and 19 reset to 10. The machine hangs out in the idle state until `CARD_IN = 1` and then moves onto getcode state. Once this happens, we can see `CYCLES_G` increment and begin to count, resetting to 0 at `KEY_PRESS = 1`. Below those waves, we see the item value correctly updating to 10, and then 19 - matching `NUMDIGITS` as well. Once both digits are accepted, the machine moves into the valid state. From the valid state, the machine spends 1 clock cycle validating the item and outputting the correct cost, 6. Once this completes, the machine transitions to the transact state. Once this happens, we can see `CYCLES_T` increment and begin to count, resetting to 0 at `VALID_TRAN = 1`. Once this happens, the machine moves into the vend state. From here we see `VEND = 1`, and immediately `COUNT_19` decrements from 10 to 9 correctly. This occurs in 1 clock cycle, and the machine moves into the final door state. With the last 3 waveforms, we can see that once `DOOR_OPEN` toggles, the `CURR_STATE = IDLE_STATE`.

Since `CARD_IN = 1` remains the entire time, the machine spends only 1 clock cycle in idle, and then moves onto a new transaction as expected - validating our design for this special case.

3.3 User inputs incorrect Item Code

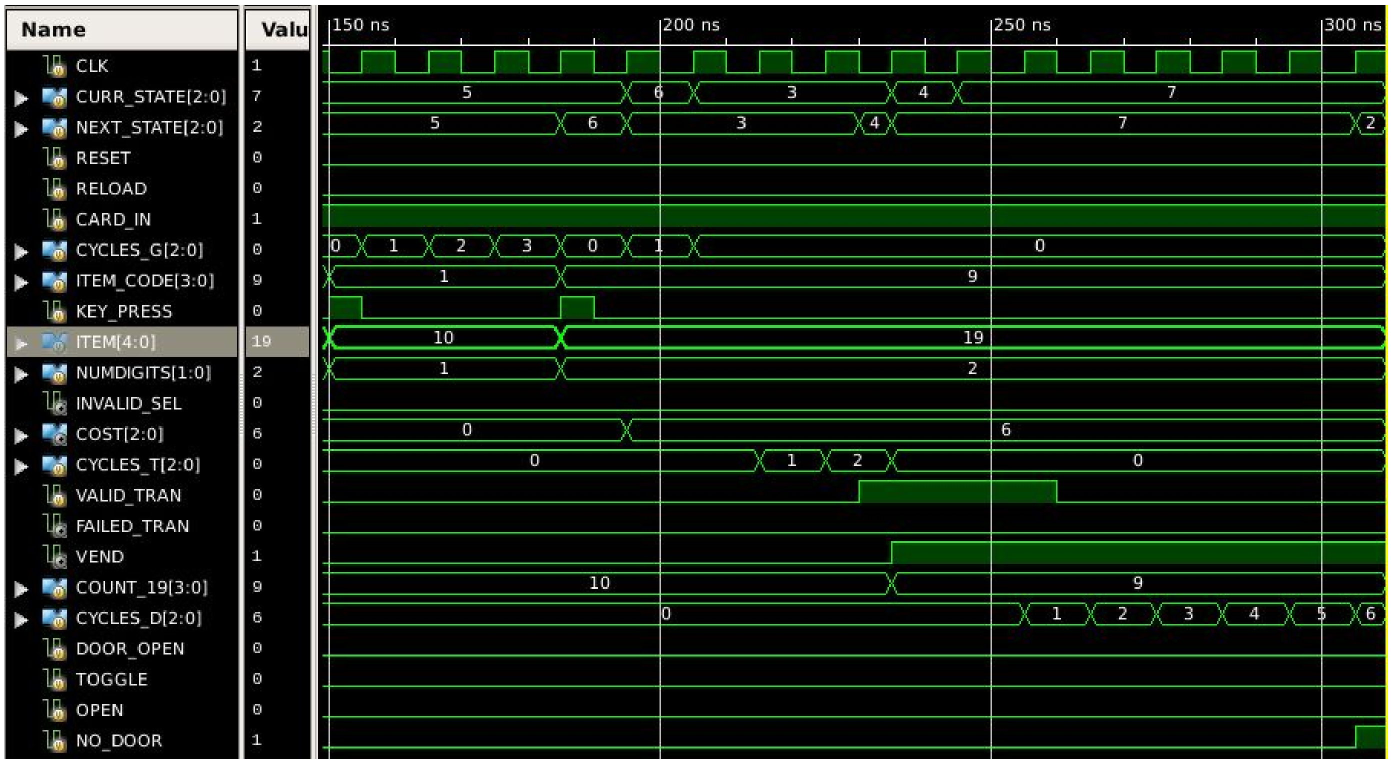
Our second test introduces the possibility of the user inputting an incorrect item code, one that exceeds the 19 possible items. In this case, out user inputs value 55 for item code. When this happens, our vending machine should set `INVALID_SEL = 1` and return the machine to idle, ready for another transaction. The waveform associated with our second test is shown below:



I used the same example as in **3.2** except switching the values of `ITEM_CODE`. Here, we see that once the machine enters `CURR_STATE = 101` (get-code state), the vending machine reads in 5 at each `KEY_PRESS = 1`. Since the first digit is not a 1 or a 0, we know the value should not be valid. Therefore, the item value jumps to 20 instead of 10, ensuring invalidity. After the second `KEY_PRESS`, the item value becomes 25. While this differs from the user's input 55, functionality remains the same. Once the item value becomes 25, `INVALID_SEL = 1.` In the next state, transact state, the machine reads `INVALID_SEL` and updates `CURR_STATE = IDLE_STATE.` This validates our design in this special case.

3.4 Door Unopened after 5 Clock Cycles

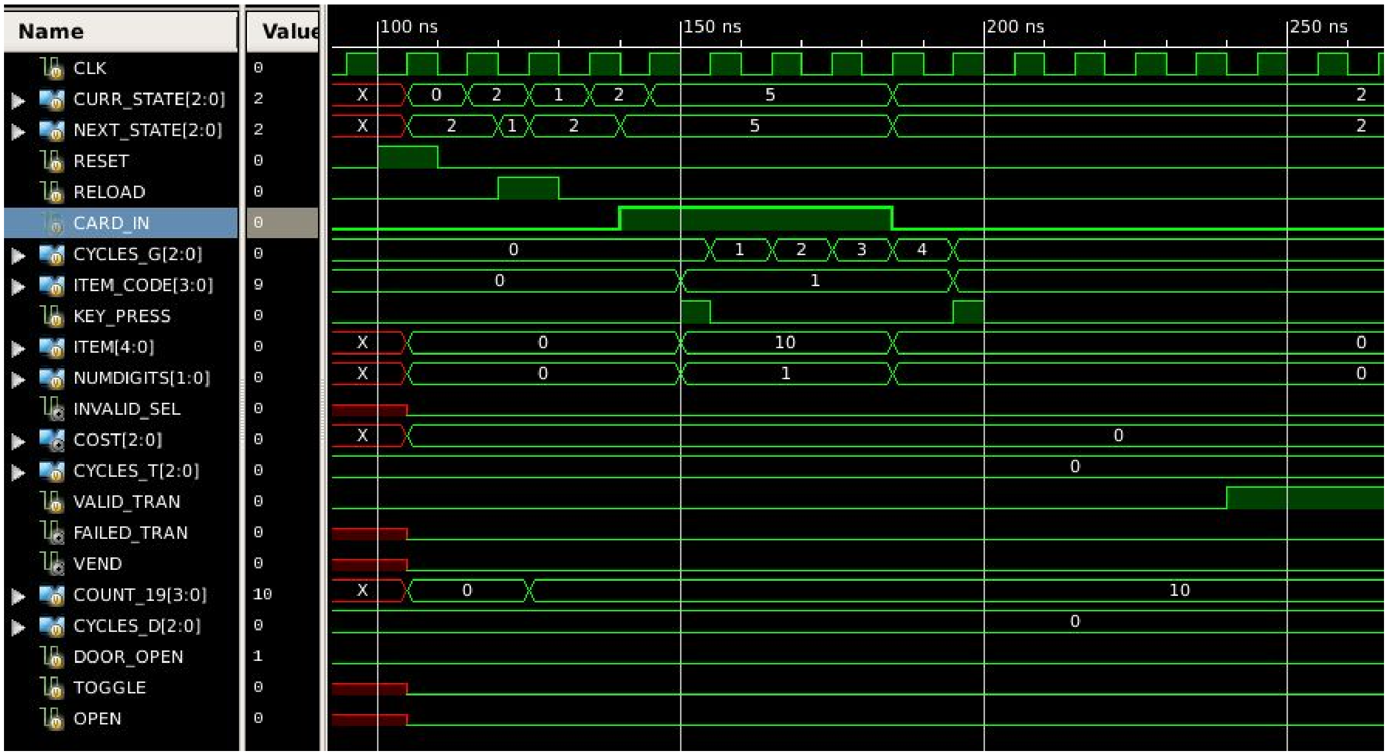
Our third test investigates the situation where the transaction is successful, but the `DOOR_OPEN` bit is never set to 1 within 5 clock cycles of the machine vending the item. In this case, the machine should return to the idle state. The waveform associated with our third test is shown below:



Again, I used the same example as in **3.2** except switching the values of `DOOR_OPEN`. Here, our waveforms start at the first `KEY_PRESS` which matches the above figures. Our waveform differs, however, once `CURR_STATE = DOOR_STATE`. We can see once this happens, `CYCLES_D` begins to increment and once 5 clock cycles have passed without `DOOR_OPEN = 1`, flag `NO_DOOR` is set, causing the machine to return to `CURR_STATE = IDLE_STATE`. While the simulation ends, we can see at the top right corner that `NEXT_STATE = IDLE_STATE` once `NO_DOOR` is set, validating our design in this special case.

3.5 Card Removed while inputting Item Code

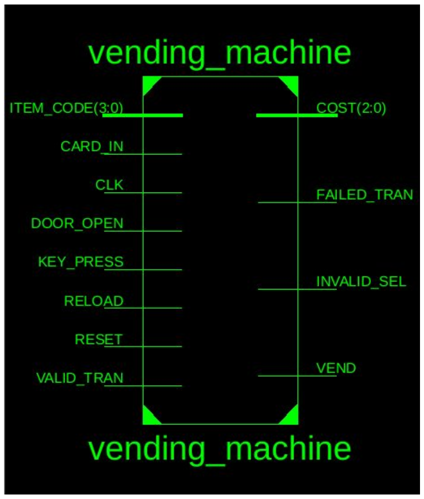
Our last test investigates the case where the user removes his credit card from the vending machine mid transaction, while he is inputting the item code. In this case, the machine should read `CARD_IN = 0` during the get-code state and return to idle, ready for the card to be re-inserted and a new transaction to begin. The waveform associated with our last test is shown below:



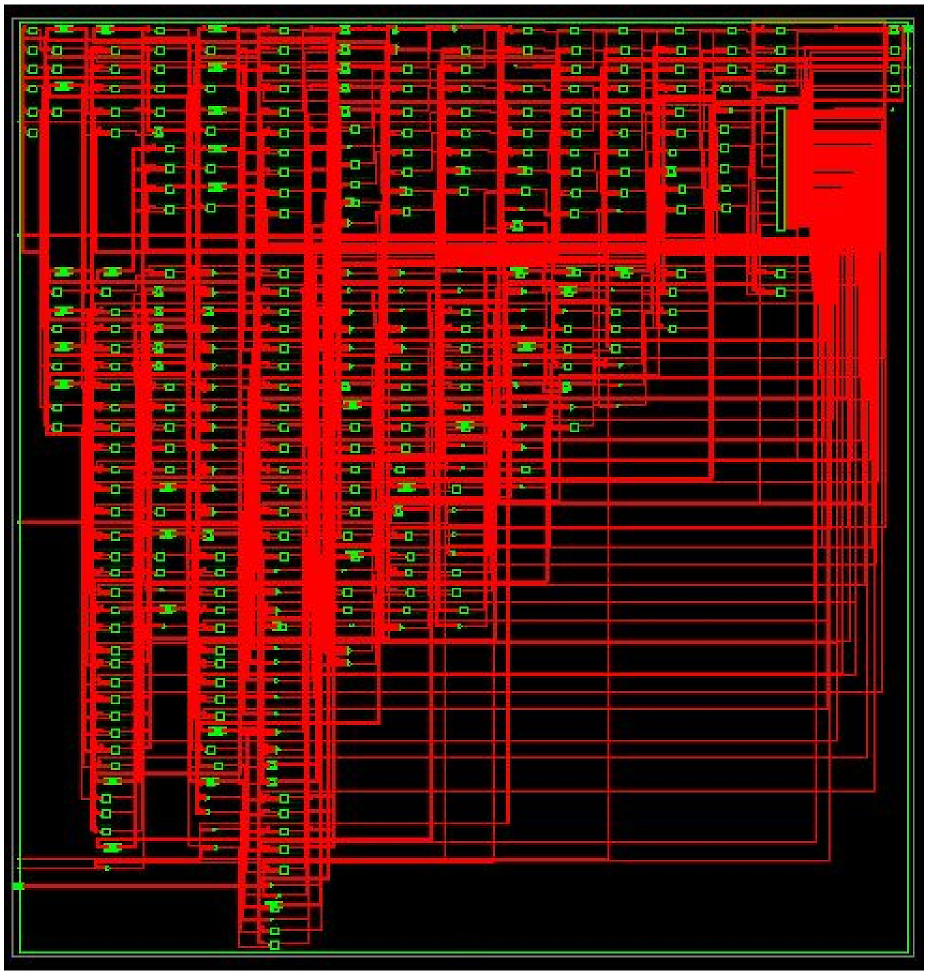
Again we start with the same example as before, except we change `CARD_IN = 0` during the get-code state just before the second `KEY_PRESS`. As we can see, the user inputs the first `KEY_PRESS` into our machine, and removes his credit card before inputting the second value. As a result, `CURR_STATE = IDLE_STATE` immediately, and remains idle waiting for the card to be re-inserted. This is visible in the first two waveforms below the clock, and validates our design in this special case.

4. Conclusion and Synthesis

After running our testbench and viewing the output waveforms, it is clear that our design and implementation of the vending machine proposal was a success. When synthesizing our verilog module, the high level RTL matches the one given to us in the spec. This is shown below:



When viewing the detailed RTL Schematic, the complexity of this abstract module becomes increasingly apparent. Shown below is a zoomed-out representation of our module being implemented by Xilinx ISE:



Most of these boxes shown are multiplexers (MUXs) and combinatorial logic. The RTL schematic shown above can be validated by the HDL Synthesis report output by the Xilinx ISE during Synthesis. The report is displayed below:

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <vending_machine>.
    Related source file is "/home/ise/Xilinx_VM/FSM/vending_machine.v".
        reset_state = 3'b000
        reload_state = 3'b001
        idle_state = 3'b010
        transact_state = 3'b011
        vend_state = 3'b100
        getcode_state = 3'b101
        valid_state = 3'b110
        door_state = 3'b111
    Found 3-bit register for signal <CYCLES_G>.
    Found 3-bit register for signal <CYCLES_T>.
    Found 3-bit register for signal <CYCLES_D>.
    Found 3-bit register for signal <CURR_STATE>.
    Found finite state machine <FSM_0> for signal <CURR_STATE>.
    -----------------------------------------------------------------------
    | States             | 8                                              |
    | Transitions        | 18                                             |
    | Inputs             | 9                                              |
    | Outputs            | 68                                             |
    | Clock              | CLK (rising_edge)                              |
    | Reset              | RESET (positive)                               |
    | Reset type         | synchronous                                    |
    | Reset State        | 000                                            |
    | Encoding           | auto                                           |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------

                                    ...

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

Macro Statistics
# RAMs                                                 : 1
 32x16-bit single-port Read Only RAM                   : 1
# Adders/Subtractors                                   : 20
```

```
 3-bit adder                                           : 3
 4-bit subtractor                                      : 16
 5-bit adder                                           : 1
# Registers                                            : 3
 3-bit register                                        : 3
# Latches                                              : 81
 1-bit latch                                           : 81
# Comparators                                          : 6
 5-bit comparator greater                              : 6
# Multiplexers                                         : 225
 1-bit 2-to-1 multiplexer                              : 217
 3-bit 2-to-1 multiplexer                              : 8
# FSMs                                                 : 1

=========================================================================
```

When synthesizing my verilog files, the synthesis report passed without errors - but the report did contain some warnings that detailed inconsistencies with some sensitivity lists. While the simulation proved successful, without a FPGA Board to implement my design on, my solutions cannot be truly verified as accurate in regards to timing issues.

After synthesizing, we implement this design virtually as if we would pin it to an FPGA Board. The Map Report Summary is shown below:

```
Design Summary
--------------

Design Summary:
Number of errors:      0
Number of warnings:   26
Slice Logic Utilization:
  Number of Slice Registers:                   102 out of  18,224    1%
    Number used as Flip Flops:                  17
    Number used as Latches:                     85
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                        150 out of   9,112    1%
    Number used as logic:                      150 out of   9,112    1%
      Number using O6 output only:             102
      Number using O5 output only:               0
      Number using O5 and O6:                   48
      Number used as ROM:                        0
    Number used as Memory:                       0 out of   2,176    0%
```

```
Slice Logic Distribution:
  Number of occupied Slices:                    56 out of   2,278    2%
  Number of MUXCYs used:                         0 out of   4,556    0%
  Number of LUT Flip Flop pairs used:          168
    Number with an unused Flip Flop:            67 out of     168   39%
    Number with an unused LUT:                  18 out of     168   10%
    Number of fully used LUT-FF pairs:          83 out of     168   49%
    Number of unique control sets:              29
    Number of slice register sites lost
      to control set restrictions:             138 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:                                 3

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:                   3 out of     248    1%
    Number used as OLOGIC2s:                     3
    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:                3.66

Peak Memory Usage:  672 MB
Total REAL time to MAP completion:  20 secs
Total CPU time to MAP completion:   18 secs

Mapping completed.
```

The map report details the specific design of my module implementation in multiplexers, latches, and combinatorial gates. Although not shown in the design summary, the map report also displays the I/O pinnings to be used on an FPGA board, which would help me physically implement my design if I wanted to truly create the vending machine.