Lab 3 : Vending Machine

EECE5117C

Team 3

**Purpose**

The purpose of this lab was to understand a mealy state machine. A mealy state machine takes in account for what state the machine is currently in, as well as inputs. In the case for this lab, the state would be selecting what kind of payment, receiving money, selection state, and dispensing state. The inputs would be the buttons used for picking the payment type, and the product selection buttons.

**Procedure**

We wrote each function separately and tested it on the board, recursively editing them until our tests passed. Then once each function worked as expected we moved on to the next function. The following procedure was used for this lab. We used this method for all functions.

**Steps**

1. Design pseudo code for how an elevator should work
2. Draw State Diagrams
3. Write test cases
4. Label the inputs and wires with the corresponding variables
5. Write the code from the diagram
6. Simulate in Vivado
7. Test on board

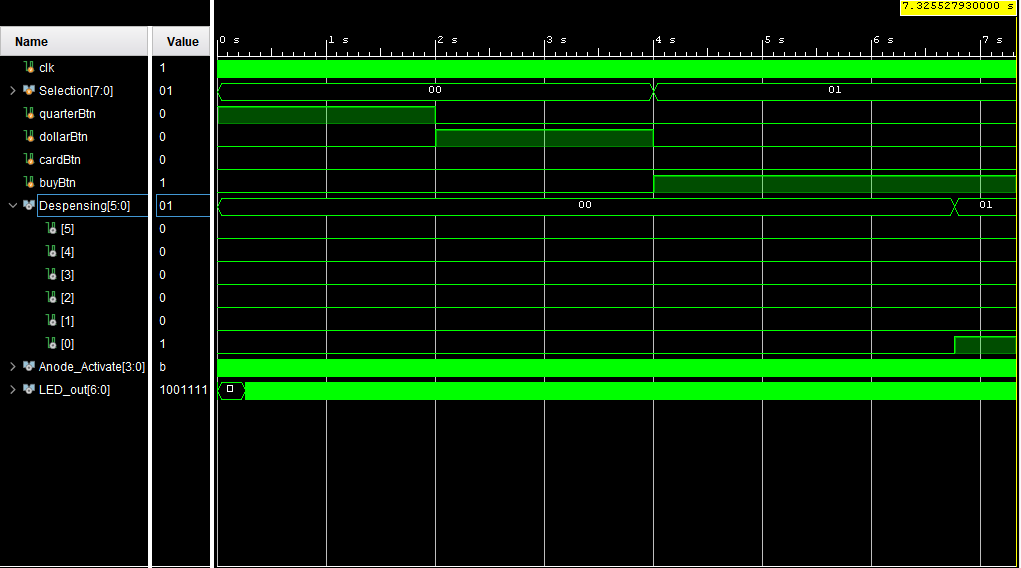


Figure 1: Simulation

The figure above shows the simulation of our test cases. The test Select product number one which is worth 50 cents. Then, the test adds a quarter then adds one dollar to the balance. After the product is selected and the money is added, buy button (buyBtn) is pressed to dispense the product. The Dispensing output are five LEDs which upon successful transaction it lights up the first LED and then the sec with the first and then the third with the previous ones till all five light up to simulate a dispensing (loading) bar. The figure above shows what was just descried till the first dispensing LED light up which confirms the purchase was successful.

**Test Bench Code**

`timescale 1ns / 1ps

module testbench;

reg clk;

reg [7:0] Selection;

reg quarterBtn;

reg dollarBtn;

reg cardBtn;

wire [5:0] Despensing;

wire [3:0] Anode\_Activate; // anode signals of the 7-segment LED display

wire [6:0] LED\_out;// cathode patterns of the 7-segment LED display

VendingMachine uut (

.clk(clk),

.Selection(Selection),

.quarterBtn(quarterBtn),

.dollarBtn(dollarBtn),

.cardBtn(cardBtn),

.Despensing(Despensing),

.Anode\_Activate(Anode\_Activate),

.LED\_out(LED\_out)

);

always begin

//Simulate the clock

#10 clk = 1;

#10 clk = 0;

end

initial begin

//Initialize variables

Selection = 0;

quarterBtn = 0;

dollarBtn = 0;

cardBtn = 0;

quarterBtn = 1; //add a quarter to the balance

#2000000000

quarterBtn = 0;

dollarBtn = 1; //add a dollar to the balance

#2000000000

dollarBtn = 0;

#20 Selection = 1; //attempt to buy product (should remove 50 cents from balance)

buyBtn = 1;

end

endmodule

**Test Vectors**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Vector** | **Money Input** | **Type of Payment** | **Product to buy** | **Output** |
| 1 | 0.25 | cash | 1 | Not enough money – display error |
| 2 | 0.50 | cash | 1 | Dispense animation – balance removes 0.50 |
| 3 | x | card | 1 | Dispense animation |
| 4 | 1.25 | cash | 5 | Dispense animation – balance removes 1.25 |
| 5 | 1.25 | cash | 4 | Error displayed, not enough product (in our code this product is preset to 0 in stock) |

**Figure 2: Test vectors for vending machine scenarios**

**Problems**

There were a few problems we had with this lab. One thing we had difficulty with was getting the seven-segment display to show different characters on each segment. For a while we could only get one character to show on the whole display. After doing some research we found out there is a combination of pins we can use to turn on/off the displays in a specific timing in order to display multiple characters.

We also had difficulty showing the “.” symbol to display monetary values. The program did not necessarily know whether a digit needed to have a period or not, since all we sent it was the number, not which display it was going to. To fix this we added a second parameter which told the program to add the period symbol to the display if the flag was on.

**Takeaways**

One of the takeaways from this lab is to let others test the product as the people building it often times tend to miss some test cases, as the test cases they thought about are what they built the code to work around. When others get to test a product, they often times come up with new test cases which might not have been considered. That helps the developers make a better more reliable product.

Another one would be, mealy machines are good when the system’s output needs to be depended on the user input along with the current state. The vending machine required that as it is needed to save the current input balance, selected product which both are user inputs, then depending on the vending machine dispensing state the program determines what state to move to.

**Team Breakdown**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Cases/Program Design | Behavioral Code | Diagrams/Reports |
| Ben Cohen | x | x | x |
| Moaz Abougabal | x | X | x |
| Quinn Nye | x | x | x |