Final Report (Vending Machine Project)

Giovanni Argueta

***Abstract*—The following paper details the process in creating a vending machine implementation on a FPGA using Vivado. It will cover the SPEC, state diagram, RTL code/synthesis, and usage of the vending machine on a Digilent Basys 3 Artix-7 FPGA trainer board. All RTL code is provided in the appendices. PmodOLEDCtrl module and sub module code is not provided as the code used was provided by the company Digilent, written by engineers Ryan Kim and Josh Sackos.**

I. Introduction

THIS report will cover the key steps in the creation of a personalized vending machine. The prompt given required the vending machine to consist of the ability for the user to input three coins: a nickel, dime, and a quarter in addition to providing four products for sale, all at set prices. On the FPGA the user must be able to see their selected cost on the left two digits, and their balance on the right-most two digits on the four-digit seven segment display. It must also have the ability of turning on LEDS once prod(s) are purchased and display the user's change. The final requirement included programming an attachable OLED to the FPGA that would display appropriate text and make the vending machine more user friendly. The rest was left up to the creator to decide on how the vending machine would operate. The first topic of discussion will be the derivation and explanation of the state diagram used to implement the vending machine controller.

II. From SPEC to Implementation

A. FSM

The first step in creating this vending machine was to create a finite state machine as we can represent each stage of the user's experience as its own state. For my state diagram I decided to have four states, init(initial), selection, dispense, and change but it is important to note that the only true states are selection and change as they require user input to proceed to the next state; the other two states automatically go into the next state as seen in Figure 1A. The initial state is used to reset all variables(cost, money, intermediate values, flags, etc) and displays a welcome screen for the user. This state will automatically go to the selection state. In the selection state the user can select product(s) and insert money. It is important to note that for my design I limited the money input to ninety-five cents. The OLED in this state displays a message that tells the user which product they selected by assigning it to either a 1 or 0; for example, if the user selects Chips, then on the screen the message will be Chips(SW3)=1, and Chips(SW3)=0 if chips is not selected. My line of products includes water priced at 25 cents, chips for 50 cents, a protein bar for 65 cents, and a soda priced at 80 cents. Once a user is done with their selections and insertion of coins, the user will have to turn on the confirmation switch(as indicated on the OLED) and will be tested for errors. These errors include not having enough money to purchase the product(s), not selecting a product, and exceeding the selection price of 95 cents. If one of or more of these errors are present, the user will be told via the OLED their mistake as well as see an error LED turning on and the FSM stays in the selection phase until that error is fixed. Once fixed, the user then enters the "dispense" phase where the corresponding product LEDS are turned on to mimic the act of dispensing virtual products. The dispense state message simply states that the vending machine is dispensing but will also include a special discount message which will be explained later. From there the user then enters the "change" state where the user's balance changes accordingly to display their change. For the user to purchase again, they will have to turn off all switches and then the cycle starts again with a cost and user balance reset to zero.

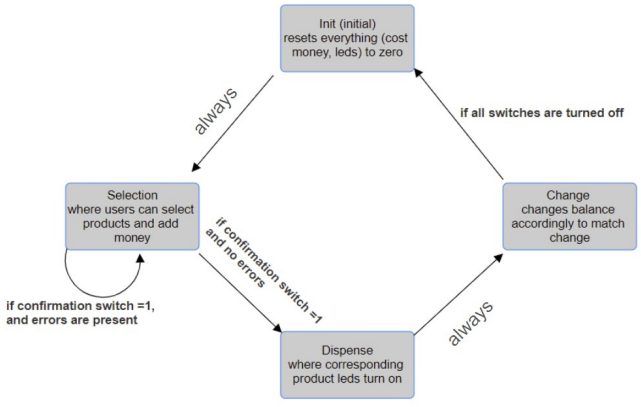

Figure 1A: State Diagram

B. Block Diagram

The finite state machine mentioned in the previous section is responsible for the logic for the vending machine controller but does not account for the whole machine implementation. The vending machine controller accounts for one block out of the four main logical blocks I used for this implementation. As seen in figure 1B, the other three main blocks include the clock enable, the display, and the PmodOLEDCtrl block. The vending machine controller has inputs of coins(represented by buttons on board) and products/confirmation (represented by switches on board) and directly outputs LEDS to the board. The other two outputs include pages 0-3, which represents each of the four lines the OLED is programmed in

and therefore must be connected directly to the PmodOLEDCtrl. For reference the OLED allows for 16 ASCII values on each line. The other output includes the digits that represent the two two-digit numbers for the cost and user balance that is connected to the display block. The display block does include other modules, but its main purpose is to translate four binary values into four digits that is shown in the seven-segment display on the FPGA board. Those digits are illuminated by using anodes and cathodes and that is why they are the outputs. The PmodOLEDCtrl block is responsible for translating the page values (that were initially given in hex values) to ASCII values that the user can see update in real time. The clock enable block is responsible for changing the frequency of each block. The provided prompt stated the vending machine must run at a 1hz refresh rate to make the state traversal have a one second delay. In order to keep the OLED on as long as the vending machine is on, I had to tie a clock enable signal to its EN(ENABLE) input as the original design had the OLED to be only on after a button is pressed, which would provide an inconvenience for users. To display almost in sync changes for the OLED in relation to the vending machine controller, I tied it to slightly slower clock enable signal (10hz) in order to make sure the OLED shows the appropriate changes. The global clock and reset were connected to all blocks and the reset button sets the FSM to the initial state. As seen in Figure 1B, this block describes the "top module", which represents the highest position in this hierarchical design. In figure 2B, the RTL schematic for the top module is shown and in figure 3B, the RTL schematic for the PmodOLEDCtrl is shown as that module involves a lot of unseen sub modules that make it function. They were both generated through Vivado's RTL Linter.

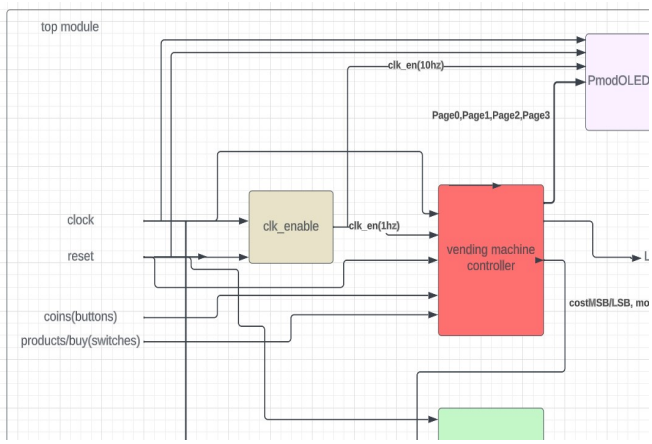

Figure 1B: Block Diagram

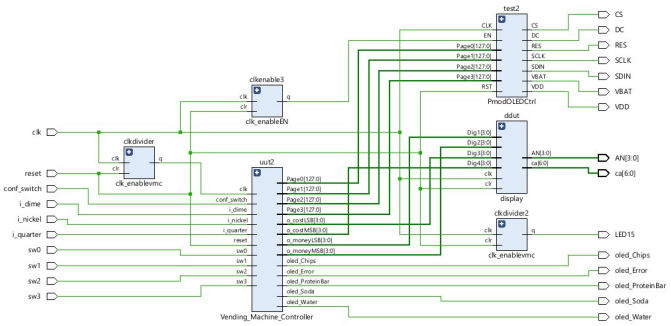

Figure 2B: RTL schematic for top module

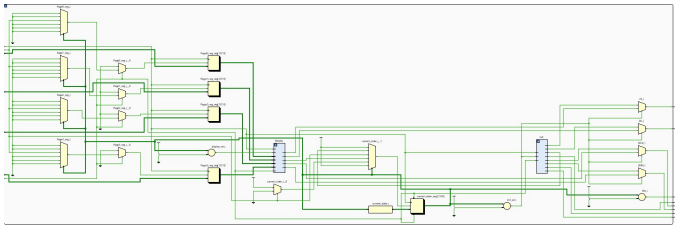

Figure 3B: RTL schematic for PmodOLEDCtrl module

C. User Manual

The user is first met with a welcome screen that then transitions to the selection screen. The user has four product switches that they can use: "SW3" on the board represents Water and costs 25 cents, "SW2" represents Chips and costs 50 cents. SW1 represents diet coke (DC) and costs 60 cents, and finally a protein bar (labeled by PB) costs 80 cents marked by "SW0". To insert coins, the user can press "W19" to insert a nickel, "U18" to represent a dime, and "T17" to insert a quarter. The "T18" button can be used as a "reset" button to bring the user back to the welcome screen and reset all selections/insertions. Once a user is done with their selections and money insertion, they can turn on "SW4" to confirm and the user will be met with four options. If the user does not have enough money to purchase selected item(s) then they will be prompted with the message "Insert more money". If they do not select a product to purchase, they will be prompted by "Select a prod". If their product selections exceed 95 cents, they will be prompted with the message "Deselect prod(s)" in order to indicate they have gone over the max. Any coins inserted that add up to over 95 cents will be ignored. All errors can be fixed without any other directions, just simply follow the order. Assuming the user does not perform any errors or fixes the error, they will transition to the dispense phase where the corresponding product LEDS turn on and then transition into the change state. In the change state their balance will change accordingly to match the change. The user is told that to buy again they must turn off switches(SW4-SW0). A hidden discount of 20 cents can be attained by selecting water and a protein bar. In Figure 1C, one can see the board setup.

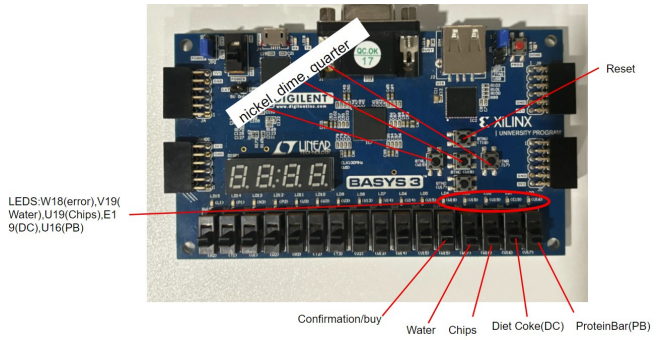

Fig 1C: Board setup

D. Simulations

In the process of coding the vending machine controller in Verilog, it was essential to test different iterations of a user's experience. This section will cover three key scenarios and show their corresponding simulation waveforms that indicate working outputs. For simplicity, these waveforms only include the key inputs and outputs as something like Page0-Page4 outputs can only be read in hex and therefore need more ample time to confirm their validity.

The first scenario cover is the most basic purchase cycle of a user selecting a water and entering a quarter. For reference in these waves forms the states are represented from 0-3, all in ascending order. Therefore state 0 represents the "init" state, state 1 represents the selection state, state 2 represents the dispense state, and state 3 represents the change state. As seen in figure 1D, the user turns on "SW0" to select water(the switch numbers were changed during selection of IO ports) and therefore the cost changes to "25", again represented by two digits instead of one together. Once a quarter is inserted, the cost turns into "25". It is only when the "conf_switch" is turned on when the user is transition from state 1(selection) to state 2(dispense) and as planned the water led turns on. It automatically goes to state 3(change) and the user's money is then changed to 0, corresponding to the leftover change. The user only is brought back to the state 0/1 again after all switches all are turned off (from 1 to 0) and then the cycle continues.

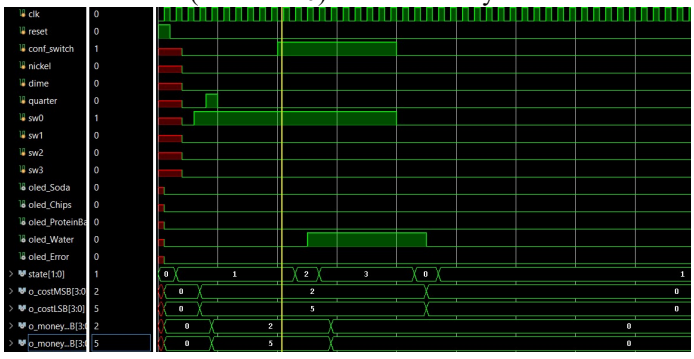

Fig 1D: Simulation 1

In the next simulation, I simulated the act of purchasing chips and deselecting water. The reason why this simulation holds great significance is in the testing stage, due to using a flag system when selecting products, my vending machine initially had a problem registering a given product was deselected and adjusting the cost accordingly. Through understand the scope of my flag if else statements, I correctly adjusted the code to make sure if a product was deselected then not only would the cost adjust accordingly but also when it came time to calculate the change, the system would only account for product switches that were still on, and not those that were turn off. As seen in Fig 2D, this simulation shows the insertion of two quarters and a dime, making the money increment accordingly up to 60 cents. The water and chips switch were turned on to account for an initial cost of 75 cents, but once the water switch is turned off the cost adjusts back to 50 cents, as the water costed 25 cents. Once the confirmation switch is turned on there are no errors so therefore it transitions into the dispense state where the chips LED turns on and again the user's money changes accordingly (10 cents) once in the change state.

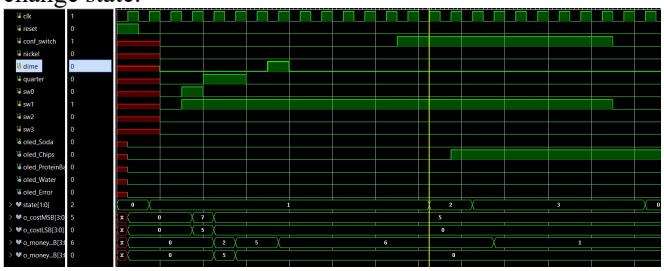

Fig 2D: Simulation 2

In this third simulation, I tested the act of receiving an error and the user fixing that error to proceed with purchasing. The error that will be tested is the user not having enough money to cover the cost of a product. As seen in figure 3D, the user is purchasing chips and adding one quarter, which accounts for the user needing 25 cents more in order to account for the cost. Once the user turns on the confirmation switch the error LED turns on and as expected, and the user stays in the selection state(1). The user is only taken out of this stage once the error is fixed and in this case another quarter is inserted to finally cover the cost. One clock cycle after the quarter is inserted, the user is brought into the dispense stage and the chips LED turns on and the user's money is adjusted accordingly to zero cents and the cycle continues once all switches are off.

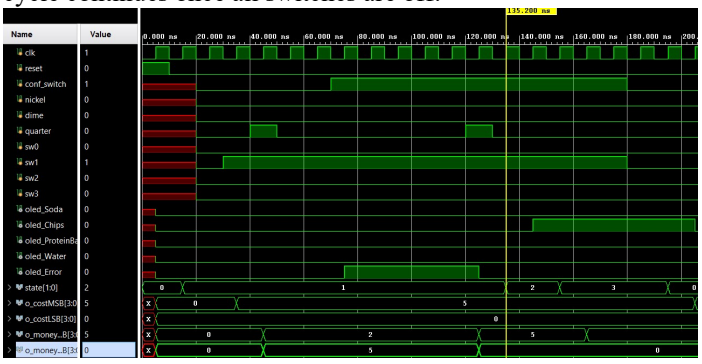

Fig 3D: Simulation 3

E. Synthesis and Implementation Reports

The next step in implementing this vending machine project was to undergo Vivado's synthesis and implementation of the top module. These reports are crucial in understanding what mistakes were left and how optimized the program was regarding utilization. As seen in the first two tables, the amount of look-up tables (LUT) used were around 24 percent usage, making up a large percentage compared to peers that had an average of around ten percent usage. This can be attributed to the large amount of intermediate integer variables I used in the vending machine controller while my peers chose to use hard-coded values for logic such as when a product is selected and deselected. The third table displays the synthesis optimization runtime and complete. It is important to note that I did receive some errors that were investigated further to make sure it did not alter the design. The synthesis finished with 0 errors, 0 critical warnings and 6 warnings.

TABLE I (POST SYNTHESIS)

Resource	Estimation	Available	Utilization
LUT	5071	20800	24.38
FF	759	41600	1.82
IO	34	106	32.08
BUFG	2	32	6.25

TABLE II (POST IMPLEMENTATION)

Resource	Utilization	Available	Utilization
LUT	5068	20800	24.37
FF	760	41600	1.83
BRAM	0.5	50	1.00
IO	34	106	32.08
BUFG	2	32	6.25

TABLE III (SYNTHESIS OPTIMIZATION RUNTIME/COMPLETE)

	Time(s):cpu	elapsed	Memory(MB):peak	gain
Runtime	00:01:26	00:04:09	153.402	574.246
Complete	00:01:41	00:04:19	1532.402	669.145

F. Demonstration results

This section will showcase the results that a user experiences while using the vending machine controller. For reference these pictures do not follow directly from another and are only for demonstration purposes. In figure 1F, one can see what the welcome screen looks like.

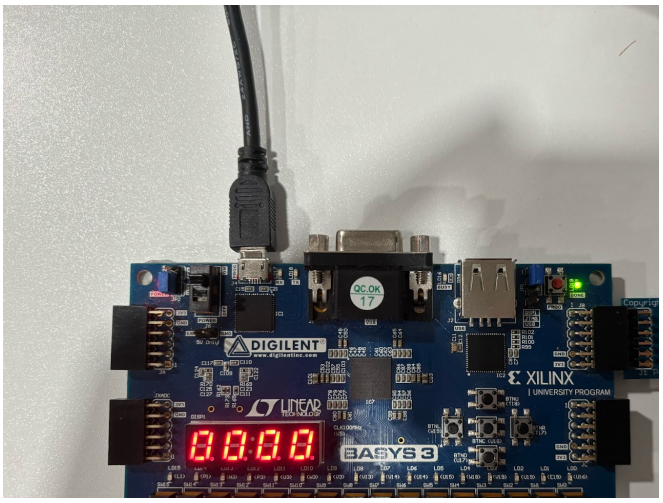

Fig 1F: Welcome screen

As explained before, the welcome screen is automatically transitioned into the selection screen. All directions are given in the OLED during this stage(Fig 2F).

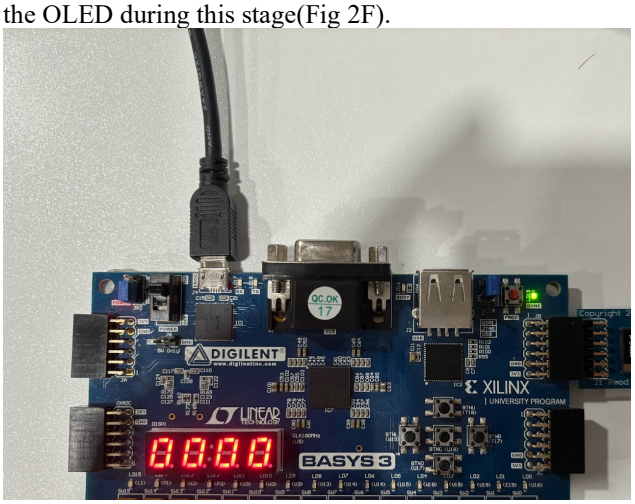

Fig 2F: Selection screen

The following three figures will show what occurs when a user triggers three errors. The first error shown in Figure 3F displays how when no products are selected, the fourth line on the OLED changes to "Select a Prod" in order to indicate they need to purchase a product in order to continue. The error LED above the confirmation switch is also turned. For this error and the following ones, the user can fix it and the error message will go away and the vending machine will progress as usual.

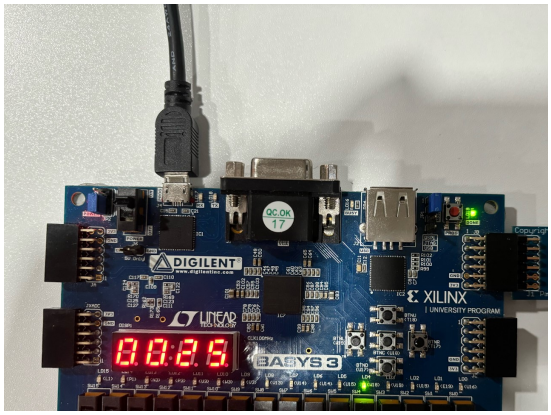

Fig 3F: Error #1 message

In Figure 4F, the user has selected water and chips to make the total cost 75 cents, but also selects the protein bar which while the display does not show, does account for a cost balance of over 95 cents. Once that confirmation switch is turned on the fourth line changes to “Deselect Prod(s)” to indicate the user has gone over the cost limit. The error LED is also turned on as well.

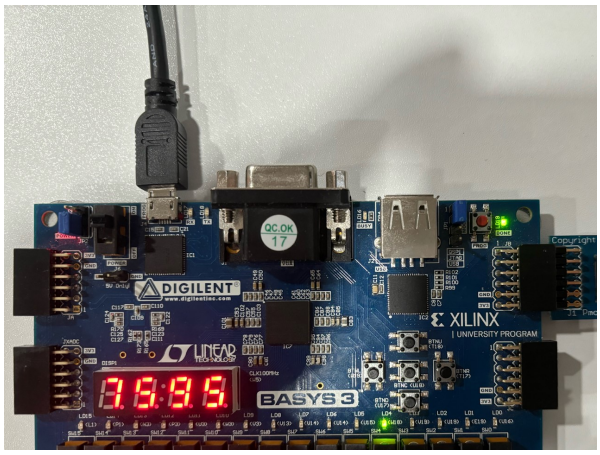

Fig 4F: Error #2 message

The third error displays a user selecting chips that cost 50 cents and confirming this purchase while only entering 25 cents. The user is met with the error message “Enter more money” and again the error LED is turned on(Figure 5F).

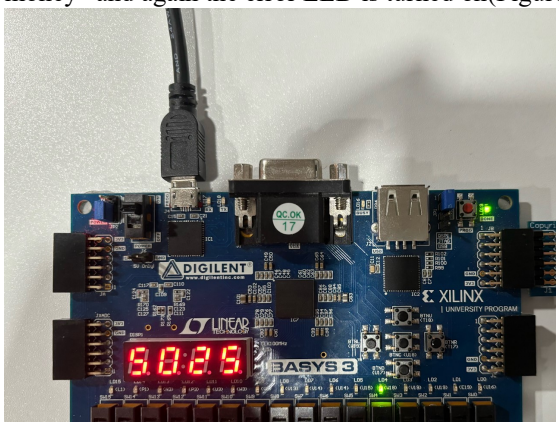

Fig 5F: Error #3 message

The following two figures display the two possible outcomes for the “dispense” state OLED messages. The normal one simply states “Dispensing” (Figure 6F) while in Figure 7F the message also indicates that a discount will be applied when purchasing a water and a protein bar. While in the figure it states the cost is 80 cents, it does decrement shortly after to 70 to apply the 20-cent discount (original 90 cents). In both iterations the corresponding product LEDs turn on.

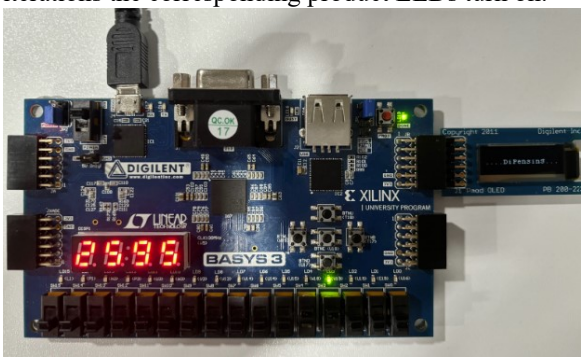

Figure 6F: Dispense state

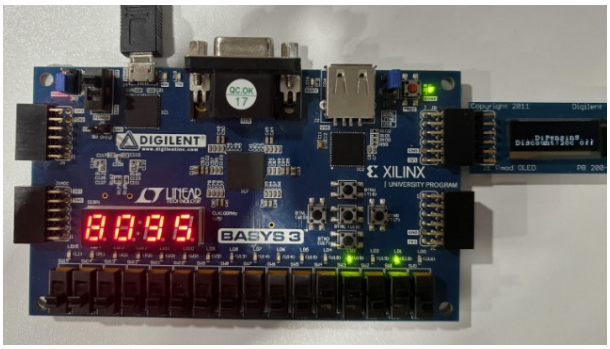

Figure 7F: Dispense state(with discount)

The last demonstration result shows the message given once in the change state, as stated before it tells the user to turn off all switches to start the cycle again. User money (right two digits) changes accordingly in this stage and the dispense LEDS are to stay on until the cycle starts again (Figure 8F).

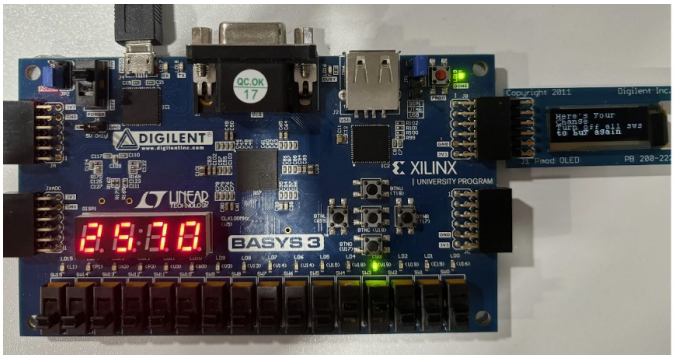

Fig 8F: Change state

G. Enhancements

This project brought the opportunity to enhance the vending machine to bring a more user-friendly experience. After trying to implement different ideas, I was left with the error enhancements and the discount. My original idea with a user performing an error was to reset the cycle and start from zero. However, after relating the project to realistic vending machines, I concluded that it was unfair to the user to essentially get money stolen from them due to causing an error. For this reason, I decided that I wanted to give the user a chance to fix their error and at the same time make it clear what exact error they made. As a mini-enhancement I decided to add a 20-cent discount for users that selected a water and a protein bar as a "health incentive".

III. Conclusion & Future Works

This conclusion section will begin with answering post-questions that were asked regarding the project. Question number one asks about the additional features added to this project, and as explained in the enhancements sub section, I added the error message and correction to make the user experience less complex and added a discount to incentivized good health choices. Question number two asks about what I used as my states and my why. As explained in the FSM sub section, I chose my four states as they represented to me the four main stages of a vending machine experience, the welcome screen, the selection stage, the dispense stage and then the change stage. I made the user only have to perform two actions in order to go from stage to stage in order to keep it simple. The third question asks about BCD and why it is useful. BCD refers to the use of writing numbers as a four-digit binary code. It was particularly useful in this project as we need four four-digit binary values to pass to each of the digit slots in the seven-segment display.

This project was a great introduction to understanding Verilog and how we can implement our logic on a FPGA board. The only real missing part from this implementation is the debouncing of buttons. To explained further as per the code, the FSM works for every positive edge clock, therefore when a user presses a button, the vending machine will only account for that button being pressed only if it is caught on the positive edge. Therefore, to combat this a user can simply hold down the button for about a second or two in order to catch that positive edge. Even regarding this feature however, this project still particularly strengthened my skills in creating an FSM and the use of a flag/protection system in Verilog versus how it is used in non-hardware programming. After successfully implementing this project, I have full confidence in applying these skills to more advanced Verilog programming and projects. Some ideas I have include building a traffic light controller, a digital oscilloscope, and designing a CPU. All of these projects will include more knowledge on the applications of FPGAs however as this project simply displayed direct implementation onto the board.

Appendix

Appendix A Vending machine controller code

```
module Vending_Machine_Controller(
input clk, input reset, input conf_switch,input i_nickel, input i_dime, input i_quarter,
input sw0, input sw1, input sw2, input sw3,
output reg oled_Soda, output reg oled_Chips, output reg oled_ProteinBar,
output reg oled_Water,output reg oled_Error,
//output reg [1:0] state,
output reg [3:0] o_costMSB, output reg [3:0] o_costLSB,
output reg [3:0] o_moneyMSB, output reg [3:0] o_moneyLSB,
output reg [127:0] Page0, output reg [127:0] Page1, output reg [127:0] Page2, output reg [127:0] Page3
);
//sw0 = Water, sw1= chips, sw2= ProteinBar, sw3 = Soda
parameter init = 2'b00;
parameter selection = 2'b01;
parameter dispense = 2'b10;
parameter change = 2'b11;

//wire [1:0] state_int;

reg [7:0] bit1;
reg [7:0]bit2;
reg [7:0]bit3;
reg [7:0]bit4;

reg [1:0] state;
reg [1:0] next_state;
integer money;
integer cost; // product cost
integer Water = 25;
integer Chips = 50;
integer ProteinBar = 65;
integer Soda= 80;
integer nickel= 5;
integer dime = 10;
integer quarter = 25;
integer flag1 = 0;
integer flag2 = 0;
integer flag3 =0;
integer flag4 =0;
integer moneyCalculated = 0;

//95 % 10 = 5, gives ls digit,
//95 /10 = 9 , integer division (rounding down) gives MS digit
// change demical(digits) to 4 bit, decoder


always @(posedge clk , posedge reset)
```

```
// assign state_int = state;
 //    assign o_state = state_int;
 if (reset == 1) begin
 state <= init;
 end
 else begin state <= next_state;
 end

 always @(posedge clk)
    begin
    case (state)

        init: begin

// "ENEB344"
 Page0 <= {8'h45, 8'h4E, 8'h45, 8'h42, 8'h33, 8'h34,
8'h34, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};
// "Vending Machine"
 Page1 <= {8'h56, 8'h65, 8'h6E, 8'h64, 8'h69, 8'h6E,
8'h67, 8'h20, 8'h4D, 8'h61, 8'h63, 8'h68, 8'h69, 8'h6E, 8'h65,
8'h20};
// "Giovanni Argueta"
 Page2 <= {8'h47, 8'h69, 8'h6F, 8'h76, 8'h61, 8'h6E,
8'h6E, 8'h69, 8'h20, 8'h41, 8'h72, 8'h67, 8'h75, 8'h65, 8'h74,
8'h61};
// "blank"
 Page3 <= {8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};

            oled_Soda =0;
            oled_Chips =0;
            oled_ProteinBar =0;
            oled_Water =0;
            oled_Error=0;
            cost = 0;
            o_costMSB = 4'b0000;
            o_costLSB= 4'b0000;
            o_moneyMSB= 4'b0000;
            o_moneyLSB= 4'b0000;
            money =0;
            flag1=0;
            flag2=0;
            flag3=0;
            flag4=0;
            moneyCalculated = 0;

            bit1=8'h30;
            bit2=8'h30;
            bit3=8'h30;
            bit4=8'h30;

  next_state= selection;
 end
    selection: begin
    oled_Error=0;

if (sw0==1) begin
  if (flag1 ==0) begin
  cost = cost + Water;
  flag1 =1;
  bit1= 8'h31;
 end
 end
 else if (sw0 == 0) begin
 if (flag1 ==1) begin
 cost = cost - Water;
 flag1 = 0;
  bit1= 8'h30;
 end
 end

if (sw1==1) begin
  if(flag2 ==0) begin
cost = cost + Chips;
flag2 = 1;
 bit2= 8'h31;
end
end
else if(sw1 == 0) begin
if (flag2 == 1) begin
cost = cost- Chips;
flag2 = 0;
 bit2= 8'h30;
end
end

if (sw2==1) begin
if(flag3 ==0) begin
  cost = cost + ProteinBar;
  flag3= 1;
   bit3= 8'h31;
  end
  end
  else if (sw2 ==0) begin
  if(flag3 ==1) begin
  cost = cost - ProteinBar;
  flag3 =0;
   bit3= 8'h30;
  end
  end

  if (sw3==1) begin
  if(flag4==0) begin
  cost = cost + Soda;
  flag4 =1;
   bit4= 8'h31;
  end
  end
  else if (sw3==0) begin
  if (flag4 ==1) begin
  cost = cost - Soda;
  flag4 =0;
```

```
            bit4= 8'h30;
            end
            end


      // "Water:SW3=(1/0)"
       Page0 <= {8'h57, 8'h61, 8'h74, 8'h65, 8'h72, 8'h3A, 8'h53, 8'h57, 8'h33, 8'h3D, bit1, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20};
      // "Chips:SW2=(1/0)"
      Page1 <= {8'h43, 8'h68, 8'h69, 8'h70, 8'h73, 8'h3A, 8'h53, 8'h57, 8'h32, 8'h3D, bit2, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20};
      // "PB:SW1=1DC:SW0=1"
       Page2 <= {8'h50, 8'h42, 8'h3A, 8'h53, 8'h57, 8'h31, 8'h3D, bit3, 8'h44, 8'h43, 8'h3A, 8'h53, 8'h57, 8'h30, 8'h3D, bit4};
      // "Buy: SW4=1"
       Page3 <= {8'h42, 8'h75, 8'h79, 8'h3A, 8'h20, 8'h53, 8'h57, 8'h34, 8'h3D, 8'h31, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20};


      if( cost <= 96) begin

       case (cost / 10)
                0: o_costMSB = 4'b0000;
                1: o_costMSB = 4'b0001;
                2: o_costMSB = 4'b0010;
                3: o_costMSB = 4'b0011;
                4: o_costMSB = 4'b0100;
                5: o_costMSB = 4'b0101;
                6: o_costMSB = 4'b0110;
                7: o_costMSB = 4'b0111;
                8: o_costMSB = 4'b1000;
                9: o_costMSB = 4'b1001;
                default: o_costMSB = 4'b0000; // Default case, display 0
            endcase

          case (cost % 10)
                0: o_costLSB = 4'b0000;
                1: o_costLSB = 4'b0001;
                2: o_costLSB = 4'b0010;
                3: o_costLSB = 4'b0011;
                4: o_costLSB = 4'b0100;
                5: o_costLSB = 4'b0101;
                6: o_costLSB = 4'b0110;
                7: o_costLSB = 4'b0111;
                8: o_costLSB = 4'b1000;
                9: o_costLSB = 4'b1001;
                default: o_costLSB = 4'b0000; // Default case, display 0
            endcase
        end


      // bonus feature
     /*  if( sw0 && sw2) begin
       cost = cost - Water;
       cost = cost - ProteinBar;
       cost = cost + 70;
       end */
        // bonus feature

       // add msb and lsb for cost
       if (money <= 95) begin
       if (i_nickel ==1) begin money = money + nickel;
       end
       if (i_dime==1)begin money = money + dime;
      end
       if (i_quarter==1) begin money = money + quarter;
      end
      end
      if (money <=95) begin
       case (money / 10)
                0: o_moneyMSB = 4'b0000;
                1: o_moneyMSB = 4'b0001;
                2: o_moneyMSB = 4'b0010;
                3: o_moneyMSB = 4'b0011;
                4: o_moneyMSB = 4'b0100;
                5: o_moneyMSB = 4'b0101;
                6: o_moneyMSB = 4'b0110;
                7: o_moneyMSB = 4'b0111;
                8: o_moneyMSB = 4'b1000;
                9: o_moneyMSB = 4'b1001;
                default: o_moneyMSB = 4'b0000; // Default case, display 0
            endcase
         case (money % 10)
                0: o_moneyLSB = 4'b0000;
                1: o_moneyLSB = 4'b0001;
                2: o_moneyLSB = 4'b0010;
                3: o_moneyLSB = 4'b0011;
                4: o_moneyLSB = 4'b0100;
                5: o_moneyLSB = 4'b0101;
                6: o_moneyLSB = 4'b0110;
                7: o_moneyLSB = 4'b0111;
                8: o_moneyLSB = 4'b1000;
                9: o_moneyLSB = 4'b1001;
                default: o_moneyLSB = 4'b0000;
            endcase

       end

      if (money >=95)begin
       money = 95;
       end
      if (conf_switch == 1) begin
          if((cost==0)) begin
              next_state= selection;
                oled_Error =1;
                 //Select a prod
                 Page3 <= {8'h53, 8'h65, 8'h6c, 8'h65, 8'h63,
```

```
8'h74, 8'h20, 8'h61, 8'h20, 8'h70, 8'h72, 8'h6F, 8'h64, 8'h20,
8'h20, 8'h20};
                    end
                if ( (money < cost) || (money==0)) begin
                    next_state = selection;
                     oled_Error = 1;
                    //Enter more money
                    Page3 <= {8'h45, 8'h6E, 8'h74, 8'h65, 8'h72,
8'h20, 8'h6D, 8'h6F, 8'h72, 8'h65, 8'h20, 8'h6D, 8'h6F, 8'h6E,
8'h65, 8'h79};
                     end
                   if( (money <cost) && (cost >95)) begin
                    next_state= selection;
                    oled_Error=1;
                    //Deselect prod(s)
                    Page3 <= {8'h44, 8'h65, 8'h73, 8'h65, 8'h6C,
8'h65, 8'h63, 8'h74, 8'h20, 8'h70, 8'h72, 8'h6F, 8'h64, 8'h28,
8'h73, 8'h29};
                     end
                      else if((money >=cost) && (money>0) &&
(cost >0) && (cost<=95))begin
                       next_state=dispense;
                     end
                    end
                     else next_state= selection;
                      end



          dispense: begin

       // "blank"
        Page0 <= {8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};
       // "blank"
        Page1 <= {8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};
       // " ....Dispensing... "
        Page2 <= {8'h2E, 8'h2E, 8'h2E, 8'h2E, 8'h44, 8'h69,
8'h70, 8'h65, 8'h6E, 8'h73, 8'h69, 8'h6E, 8'h67, 8'h2E, 8'h2E,
8'h2E};
       // "blank"
        Page3 <= {8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};

          oled_Error =0;
          if( sw0 && sw2) begin
          cost = cost - Water;
          cost = cost - ProteinBar;
          cost = cost + 80;
          //Discount!20c off
          Page3 <= {8'h44, 8'h69, 8'h73, 8'h63, 8'h6F, 8'h75,
8'h6E, 8'h74, 8'h21, 8'h32, 8'h30, 8'h63, 8'h20, 8'h6F, 8'h66,
8'h66};
          end


            if( cost<=96) begin
          case (cost / 10)
                    0: o_costMSB = 4'b0000;
                    1: o_costMSB = 4'b0001;
                    2: o_costMSB = 4'b0010;
                    3: o_costMSB = 4'b0011;
                    4: o_costMSB = 4'b0100;
                    5: o_costMSB = 4'b0101;
                    6: o_costMSB = 4'b0110;
                    7: o_costMSB = 4'b0111;
                    8: o_costMSB = 4'b1000;
                    9: o_costMSB = 4'b1001;
                    default: o_costMSB = 4'b0000; // Default
case, display 0
                endcase

              case (cost % 10)
                    0: o_costLSB = 4'b0000;
                    1: o_costLSB = 4'b0001;
                    2: o_costLSB = 4'b0010;
                    3: o_costLSB = 4'b0011;
                    4: o_costLSB = 4'b0100;
                    5: o_costLSB = 4'b0101;
                    6: o_costLSB = 4'b0110;
                    7: o_costLSB = 4'b0111;
                    8: o_costLSB = 4'b1000;
                    9: o_costLSB = 4'b1001;
                    default: o_costLSB = 4'b0000; // Default
case, display 0
                endcase
                end
        /* if( sw0 && sw2) begin
            cost = cost - Water;
            cost = cost - ProteinBar;
            cost = cost + 80;
            end */

             if(sw0) begin
             oled_Water = 1;
             end
             if (sw1) begin
             oled_Chips =1;
             end
             if (sw2) begin
             oled_ProteinBar =1;
             end
             if (sw3) begin
             oled_Soda =1;
             end

             if(money >= cost) begin
            next_state = change;
            end



            end
```

```
          change: begin

      // "Here's your"
      Page0 <= {8'h48, 8'h65, 8'h72, 8'h65, 8'h27, 8'h73,
8'h20, 8'h79, 8'h6F, 8'h75, 8'h72, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};
      // "Change"
       Page1 <= {8'h43, 8'h68, 8'h61, 8'h6E, 8'h67, 8'h65,
8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20, 8'h20,
8'h20};
      // "Turn off all sws"
       Page2 <= {8'h54, 8'h75, 8'h72, 8'h6E, 8'h20, 8'h6F,
8'h66, 8'h66, 8'h20, 8'h61, 8'h6C, 8'h6C, 8'h20, 8'h73, 8'h77,
8'h73};
      // "to buy again"
       Page3 <= {8'h74, 8'h6F, 8'h20, 8'h62, 8'h75, 8'h79,
8'h20, 8'h61, 8'h67, 8'h61, 8'h69, 8'h6E, 8'h20, 8'h20, 8'h20,
8'h20};

            if (!moneyCalculated && (money >= cost)) begin
           money = money - cost;
           moneyCalculated =1 ;
        end
        if(money <=95)begin
             case (money / 10)
                0: o_moneyMSB = 4'b0000;
                1: o_moneyMSB = 4'b0001;
                2: o_moneyMSB = 4'b0010;
                3: o_moneyMSB = 4'b0011;
                4: o_moneyMSB = 4'b0100;
                5: o_moneyMSB = 4'b0101;
                6: o_moneyMSB = 4'b0110;
                7: o_moneyMSB = 4'b0111;
                8: o_moneyMSB = 4'b1000;
                9: o_moneyMSB = 4'b1001;
                default: o_moneyMSB = 4'b0000;
             endcase
        case (money % 10)
                0: o_moneyLSB = 4'b0000;
                1: o_moneyLSB = 4'b0001;
                2: o_moneyLSB = 4'b0010;
                3: o_moneyLSB = 4'b0011;
                4: o_moneyLSB = 4'b0100;
                5: o_moneyLSB = 4'b0101;
                6: o_moneyLSB = 4'b0110;
                7: o_moneyLSB = 4'b0111;
                8: o_moneyLSB = 4'b1000;
                9: o_moneyLSB = 4'b1001;
                default: o_moneyLSB = 4'b0000;
             endcase
             end
            if ((sw0 == 0) && (sw1 == 0) && (sw2 ==0)
&& (sw3 ==0))
             next_state = init;
        end
          endcase
```

Appendix B clock enable code(1 hz)

```
    module clk_enablevmc (
    input clk,
    input clr,
    output reg q );

    reg [27:0] S;
    always @(posedge clk or posedge clr)begin
    if (clr == 1) begin
    S = 2'b00;
   end
   else if (S==99999999) begin
    S=0;
   q=1;
    end
    else begin
       S= S + 1;
       q=0;
    end

    end


  endmodule
```

Appendix C Display module

```
    module display(
    input clr,
    input clk,
    input [3:0] Dig1,
    input [3:0] Dig2,
    input [3:0] Dig3,
    input [3:0]Dig4,
    output [3:0]AN,
    output [6:0] ca
    );
    wire q;
    wire [1:0] S;
    wire [3:0] x;

    clk_enable c1(
    .clk(clk), .clr(clr), .q(q));

    anode_driver
an(.clk(clk),.clr(clr),.clk_en(q),.S(S),.AN(AN));

    mux4to1
m1(.Dig1(Dig1),.Dig2(Dig2),.Dig3(Dig3),.Dig4(Dig4),.S(S
),.x(x));

    hex2sevseg hex(.x(x),.ca(ca));

  endmodule
```

Appendix C.1 anode driver(sub module)

```
module anode_driver(
input clk,
input clr,
input clk_en,
output reg[1:0]S,
output reg[3:0]AN
);
always @(posedge clk)begin
if (clr == 1) begin
S = 2'b00;
end

if(clk_en ) begin
S= S + 1;
//S=2'b11;
end

end



always @(S) begin
if (S==2'b00)
AN = 4'b1110;
else if (S==2'b01)
AN = 4'b1101;
else if (S==2'b10)
AN= 4'b1011;
else if (S==2'b11)
AN = 4'b0111;
end

endmodule
```

Appendix C.2 mux4to1 (sub module)

```
module mux4to1(
input [1:0] S,
input [3:0]Dig1,
input [3:0]Dig2,
input [3:0]Dig3,
input [3:0]Dig4,
output reg [3:0] x
);
always @(*) begin
case(S)
   2'b00: x=Dig1;
   2'b01: x=Dig2;
   2'b10: x=Dig3;
   2'b11: x=Dig4;
   default: x=4'b1111;
endcase
end

endmodule
```

Appendix C.3 hex to seven segment(sub module)

```
(
input [3:0] x,
output reg [6:0] ca
);

// Define 7-segment display patterns for hexadecimal digits
always @(*)
begin
   case (x)
      4'b0000: ca = 7'b0000001;
      4'b0001: ca = 7'b1001111;
      4'b0010: ca = 7'b0010010;
      4'b0011: ca = 7'b0000110;
      4'b0100: ca = 7'b1001100;
      4'b0101: ca = 7'b0100100;
      4'b0110: ca = 7'b0100000;
      4'b0111: ca = 7'b0001111;
      4'b1000: ca = 7'b0000000;
      4'b1001: ca = 7'b0000100;
      4'b1010: ca = 7'b0001000;
      4'b1011: ca = 7'b1100000;
      4'b1100: ca = 7'b0110001;
      4'b1101: ca = 7'b1000010;
      4'b1110: ca = 7'b0110000;
      4'b1111: ca = 7'b0111000;
      default: ca = 7'bxxxxxxx;
   endcase
end

endmodule
```

Appendix D top module

```
   module top(
input clk, input reset , input conf_switch, input i_nickel , input i_dime, input i_quarter,
input sw0, input sw1, input sw2, input sw3, output oled_Soda, output oled_Chips, output oled_ProteinBar,
output oled_Water, output oled_Error, output LED15,

output [6:0] ca, output [3:0] AN,

   output wire CS,
   output wire SDIN,
   output wire SCLK,
   output wire DC,
   output wire RES,
   output wire VBAT,
   output wire VDD



   );
   wire [3:0]o_moneyMSB; wire [3:0] o_moneyLSB;
   wire [3:0]o_costMSB; wire [3:0] o_costLSB;
   wire clk_en;
   wire clk_en2;
```

```
    wire[127:0] Page0, Page1, Page2, Page3;

    //mapping EN to 10hz clock
    clk_enableEN clkenable3(
    .clk(clk),.clr(reset),.q(clk_en2));

    //mapping 1hz clock to LED15
    clk_enablevmc clkdivider2(
    .clk(clk),.clr(reset),.q(LED15));

    clk_enablevmc clkdivider(
    .clk(clk),.clr(reset),.q(clk_en));

    Vending_Machine_Controller uut2(
    .clk(clk_en),  .reset(reset),.conf_switch(conf_switch),
.i_nickel(i_nickel),
    .i_dime(i_dime),  .i_quarter(i_quarter),  .sw0(sw0),
.sw1(sw1), .sw2(sw2),.sw3(sw3),.oled_Soda(oled_Soda),

.oled_Chips(oled_Chips),.oled_ProteinBar(oled_ProteinBar
),.oled_Water(oled_Water),
    .oled_Error(oled_Error),

.o_moneyMSB(o_moneyMSB),.o_moneyLSB(o_moneyLS
B),
    .o_costMSB(o_costMSB), .o_costLSB(o_costLSB),
    .Page0(Page0),.Page1(Page1),
.Page2(Page2),.Page3(Page3)
    );

    display ddut(
    .clr(reset),.clk(clk),
.Dig4(o_costMSB),.Dig3(o_costLSB),
.Dig2(o_moneyMSB),.Dig1(o_moneyLSB),.ca(ca),
.AN(AN));

    PmodOLEDCtrl test2 (

      .CLK(clk),
      .RST(reset),
      .EN(clk_en2),
      .Page0(Page0),
      .Page1(Page1),
      .Page2(Page2),
      .Page3(Page3),
      .CS(CS),
      .SDIN(SDIN),
      .SCLK(SCLK),
      .DC(DC),
      .RES(RES),
      .VBAT(VBAT),
      .VDD(VDD)

    );

  endmodule
```

Appendix E simulation code #1

```
    module vending_fsm_tb;
  reg clk;reg reset;reg conf_switch;reg nickel;reg dime;reg
quarter;
  reg sw0;reg sw1;reg sw2;reg sw3;
  wire oled_Soda;wire  oled_Chips; wire oled_ProteinBar;
wire oled_Water;
  wire oled_Error;
  wire [1:0] state;
  wire [3:0] o_costMSB; wire [3:0] o_costLSB;
  wire [3:0] o_moneyMSB; wire [3:0] o_moneyLSB;

  Vending_Machine_Controller uut (
   .clk(clk),  .reset(reset),  .conf_switch(conf_switch),
.i_nickel(nickel),.i_dime(dime),.i_quarter(quarter),
   .sw0(sw0),  .sw1(sw1),  .sw2(sw2),  .sw3(sw3),
.oled_Soda(oled_Soda),

.oled_Chips(oled_Chips),.oled_ProteinBar(oled_ProteinBar
), .oled_Water(oled_Water),.oled_Error(oled_Error),
    .state(state),.o_costMSB(o_costMSB),
.o_costLSB(o_costLSB),
   .o_moneyMSB(o_moneyMSB),
.o_moneyLSB(o_moneyLSB)
   );
   initial begin
   clk= 0;
   forever #5 clk = ~clk;

   end
   initial begin
   reset=1;
   #10
   reset=0;
   #10
   conf_switch =0;
   nickel =0;
   dime = 0;
   quarter =0;
   sw0 =0;
   sw1= 0;
   sw2= 0;
   sw3 = 0;
  #10
  sw0=1;
  quarter=1;
   #10
   quarter=0;
   #10
   conf_switch =1;
  #100

   conf_switch=0;
   sw0=0;
```

```
    end

   endmodule
```

Appendix E.1 simulation code #2

```
      module vending_fsm_tb;
    reg clk;reg reset;reg conf_switch;reg nickel;reg dime;reg
quarter;
   reg sw0;reg sw1;reg sw2;reg sw3;
   wire oled_Soda;wire  oled_Chips; wire oled_ProteinBar;
wire oled_Water;
   wire oled_Error;
   wire [1:0] state;
   wire [3:0] o_costMSB; wire [3:0] o_costLSB;
   wire [3:0] o_moneyMSB; wire [3:0] o_moneyLSB;

   Vending_Machine_Controller uut (
    .clk(clk),      .reset(reset),      .conf_switch(conf_switch),
.i_nickel(nickel),.i_dime(dime),.i_quarter(quarter),
    .sw0(sw0),      .sw1(sw1),      .sw2(sw2),      .sw3(sw3),
.oled_Soda(oled_Soda),

.oled_Chips(oled_Chips),.oled_ProteinBar(oled_ProteinBar
), .oled_Water(oled_Water),.oled_Error(oled_Error),
     .state(state),.o_costMSB(o_costMSB),
.o_costLSB(o_costLSB),
    .o_moneyMSB(o_moneyMSB),
.o_moneyLSB(o_moneyLSB)
    );
    initial begin
    clk= 0;
    forever #5 clk = ~clk;

    end
    initial begin
    reset=1;
    #10
    reset=0;
    #10
    conf_switch =0;
    nickel =0;
    dime = 0;
    quarter =0;
    sw0 =0;
    sw1= 0;
    sw2= 0;
    sw3 = 0;
   #10
   sw0=1;
   sw1=1;
    quarter=1;
    #10
    quarter=0;
    #10
    quarter=1;
    #10
    quarter=0;
    #10
   dime=1;
    #10
    dime=0;
    #10
    conf_switch =1;
   #100

    conf_switch=0;
   sw1=0;

    end

   endmodule
```

Appendix E.2 simulation code #3

```
      module vending_fsm_tb;
    reg clk;reg reset;reg conf_switch;reg nickel;reg dime;reg
quarter;
   reg sw0;reg sw1;reg sw2;reg sw3;
   wire oled_Soda;wire  oled_Chips; wire oled_ProteinBar;
wire oled_Water;
   wire oled_Error;
   wire [1:0] state;
   wire [3:0] o_costMSB; wire [3:0] o_costLSB;
   wire [3:0] o_moneyMSB; wire [3:0] o_moneyLSB;

   Vending_Machine_Controller uut (
    .clk(clk),      .reset(reset),      .conf_switch(conf_switch),
.i_nickel(nickel),.i_dime(dime),.i_quarter(quarter),
    .sw0(sw0),      .sw1(sw1),      .sw2(sw2),      .sw3(sw3),
.oled_Soda(oled_Soda),

.oled_Chips(oled_Chips),.oled_ProteinBar(oled_ProteinBar
), .oled_Water(oled_Water),.oled_Error(oled_Error),
     .state(state),.o_costMSB(o_costMSB),
.o_costLSB(o_costLSB),
    .o_moneyMSB(o_moneyMSB),
.o_moneyLSB(o_moneyLSB)
    );
    initial begin
    clk= 0;
    forever #5 clk = ~clk;

    end
    initial begin
    reset=1;
    #10
    reset=0;
    #10
    conf_switch =0;
    nickel =0;
    dime = 0;
    quarter =0;
    sw0 =0;
    sw1= 0;
    sw2= 0;
```

```
 sw3 = 0;
#10
sw1=1;
quarter=1;
 #10
 quarter=0;
 #10
 conf_switch =1;
#50
quarter=1;
 #10
 quarter=0;
 #10
 conf_switch=0;
sw1=0;

 end

endmodule
```