Logic Circuit Project - Report

"Let's make a vending machine"

Sina Hakimzadeh

40131857

1. Divide our problem to smaller pieces

1.Customer

This module developed to check customer money and the number of products in the vending machine.If everything is ok, the product will be given to the customer.Otherwise , the red-light will turn on.

2.Owner

This module was developed to give the owner the ability to change the number of supplies in the vending machine.And of course the ability to retrieve money from the vending machine.

3.Display

As a rule we need to display our output with LEDs, LCDs and etc.So for this project we are using 7-segment lights.Every output we get from another module will pass to the display module to compute the sequence of bright lights.

4.Main (Head)

Finally, we will connect all modules in this one and also read the vending machine information from the file.

2. Customer mode “Let’s buy some stuff ”

First we declare our inputs and outputs.

```
module Customer(
    input [1:0]mode,
    input clk,
    input [3:0]price, //from array
    input [3:0]amount, //from customer
    input [6:0]money,//from customer
    input [8:0]mahcineAcc,//from array
    input  [3:0]supply,//from array
    output reg redLight,
    output reg [8:0]machineAcc_out,
    output reg [3:0]supply_out, //supply left for array
    output reg [6:0]remaining_money
    );
```

To ensure that every output and wire we need has an acceptable value, we have some initial values in place.

```
        initial redLight = 0;
        wire [10:0]Wire = (price * amount);
        initial  machineAcc_out = 9'd0;
        initial remaining_money = 7'd0;
```

There is an always block that works with a positive edge clock.

First `if` check the mode and if the mode is not correct the outputs take inputs unchanged values.Otherwise, we compare the customer money with supply prices and if every condition is true, we change the inputs values and give them to the outputs, and if not the redlight will turn on.

Keep in mind that we considered the price and supply number that has been inputted is about the supply type that customer wants to buy,this part handled in the main module.

```
always @(posedge clk)
                begin
                if (mode == 2'b01)
                begin
                        if (money >= (Wire) && (amount <= supply))
                                begin
                                supply_out <= supply - amount;
                                machineAcc_out <= (mahcineAcc + (Wire));
```

```
                        remaining_money <= money - (amount * price);
                        redLight <= 1'b0;
                        end
                    else begin
                        supply_out <= supply;
                        machineAcc_out <= mahcineAcc;
                        remaining_money <= money;
                        redLight <= 1'b1;
                    end
            end
            else begin
                    supply_out <= supply;
                    remaining_money <= money;
                    machineAcc_out <= mahcineAcc;
            end
        end
endmodule
```

3.Owner mode “Let’s charge vending machine”

Like the previous module, we declare our inputs and outputs and also initial the values.

In “always block” we check the mode and then the condition of this module(The vending machine is capable of being filled with this number of supplies.). If the condition becomes wrong the redlight will turn on and the outputs take unchanged input values.

```
        always@(posedge clk)
        begin
        if (mode == 2'b10)
        begin
               if ((Wire) > 4'b1111) begin
                    supply_out <= supply;
                    redLight <= 1'b1;
            end
             else
                    begin
                    supply_out <= Wire;
                    redLight <= 1'b0;
                    end
            end
            else
                    supply_out <= supply;
        end
```

4.Owner mode “Let’s become rich”

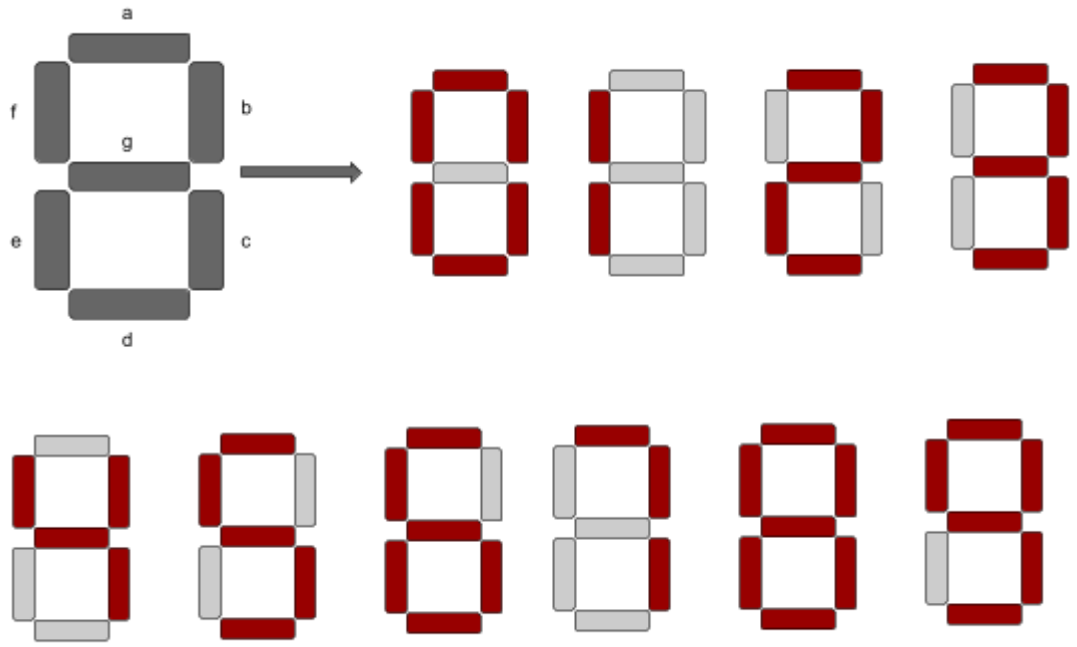
Similar to the previous module, we declare our inputs and outputs and initialize their values as well. This module is only developed to handle the total money that has been retrieved by the owner and record it.However the primary purpose of adding this mode is to enable the owner to retrieve all the money that is in the vending machine,This part handled in Main module to prevent use of a wire for output with the fixed value of 0.

```
always @(posedge clk)
	begin
	if (mode == 2'b11)
		begin
		totalMoneyRetrieve_out <= totalMoneyRetrieve + machineAcc;
		if (machineAcc == 1'b0)
			redLight <= 1;
		else
			redLight <= 0;
		end		else
		totalMoneyRetrieve_out <= totalMoneyRetrieve;

	end
```

5.Display “Let’s play with lights”

We need to show our output to customers and it must be very simple to help them understand the message, for this we are using 7-segments.



We input a number to display the module and the output will contain an 0s and 1s string that represent one of these modes. 3ns delayed to ensure the data is updated.

```
        parameter s0 = 7'b1111110;
        parameter s1 = 7'b0000110;
        parameter s2 = 7'b1101101;
        parameter s3 = 7'b1111001;
        parameter s4 = 7'b0110011;
        parameter s5 = 7'b1011011;
        parameter s6 = 7'b1011111;
        parameter s7 = 7'b1110000;
        parameter s8 = 7'b1111111;
        parameter s9 = 7'b1111011;

       always@(posedge clk)begin
               #3
               case(data % (4'd10))
                       4'b0000 : first7S = s0;
                       4'b0001 : first7S = s1;
                       4'b0010 : first7S = s2;
                       4'b0011 : first7S = s3;
                       4'b0100 : first7S = s4;
                       4'b0101 : first7S = s5;
                       4'b0110 : first7S = s6;
                       4'b0111 : first7S = s7;
                       4'b1000 : first7S = s8;
                       4'b1001 : first7S = s9;
               endcase
               if (data > 4'd9)
                       second7S = s1;
               else
                       second7S = s0;

       end
```

Each parameter represents a distinct state. Therefore, if the number consists of two parts, the Most Significant Bit (MSB) becomes 1, and the Least Significant Bit (LSB) becomes one of the declared states.

For the error signal we made a separate module because it needs only 1 light.

6.Main(Head) “Let’s connect and give meaning”

We developed all of these modules to reach this point.Connecting all modules and using them as they are working.

Like every other module, we start with declaring inputs and outputs.

```
module Main(
input [1:0] mode,
input clk,
//mode 1 inputs:
input [6:0]customer_money, //7 bit for 225 state.
input [2:0]supply_type,  //for mode 1 and 2.
input [3:0]customer_amount,
//mode 2 inputs:
input [3:0]amount_sypply_to_add,
//mode 3 inputs: -

output [6:0]error,
output [6:0]First7_machine,
output [6:0]Second7_machine,
output [6:0]First7_customer,
output [6:0]Second7_customer
    );
```

At the beginning of the module, we make certain declarations that are utilized in other parts of the code.

`reg [10:0] mem_array [7:0];`	To save data that read from file
`reg [3:0]supply[7:0];` `reg [3:0]prices[7:0];`	Separate every type of supply prices and quantities.
`reg [8:0]machineAcc;` `reg [6:0]history;`	The recorded amount of money in the vending machine.

`wire redLight1;`
`wire redLight2;`
`wire redLight3;`
`Wire redLight;`
	To connect them to other modules output and check errors.
	The last redlight will calculate the result of the error status.

`wire [8:0]machineAcc_out;`
`wire [3:0]supply_outC;`
`wire [6:0]remaining_money;`
	To connect them to other modules output to update the arrays and display the customer remaining money.these are for the customer module

`wire [3:0]supply_outO;`	To connect them to other modules output to update the arrays.this is for the owner module
`wire[8:0]totalMoneyRetrieve_out;`	To establish a connection with other modules' outputs and update the history.

Table.1 Note:Keep in mind that the history part is not complete in this project.

We read the vending machine initial values from the file by using the `$readmemb()` function and save them in our arrays. This function is reading the data in binary mode.

```
initial begin
              $readmemb("stuff.txt" , mem_array);
              supply[0] = mem_array[0][7:4];
              prices[0] = mem_array[0][3:0];

              supply[1] = mem_array[1][7:4];
              prices[1] = mem_array[1][3:0];

              supply[2] = mem_array[2][7:4];
              prices[2] = mem_array[2][3:0];

              supply[3] = mem_array[3][7:4];
              prices[3] = mem_array[3][3:0];

              supply[4] = mem_array[4][7:4];
              prices[4] = mem_array[4][3:0];

              supply[5] = mem_array[5][7:4];
              prices[5] = mem_array[5][3:0];

              supply[6] = mem_array[6][7:4];
              prices[6] = mem_array[6][3:0];

              supply[7] = mem_array[7][7:4];
              prices[7] = mem_array[7][3:0];
       end
```

In verilog when we pass a reg or array to another module it always passes a copy of the reg or array unlike programming languages like C, C++, java, python and etc. So we use a wire to take updated data from modules and use them to update our arrays. We need an always block to continuously update our arrays using the wire.

To ensure that our wires have taken a value, we require a delay. This delay is particularly useful for handling the first clock cycle, as the wires may not have any values during that time.

```
always@(posedge clk) begin //always block for updating the array values
              #2
              if (mode == 2'b01)begin
                     supply[supply_type] <= supply_outC;
                     machineAcc <= machineAcc_out;
              end
              else if (mode == 2'b10)
                     supply[supply_type] <= supply_outO;
              else if (mode == 2'b11)begin
                     machineAcc = 5'b00000;
                     history <= totalMoneyRetrieve_out;
```

```
            end
            else if (mode == 2'b00)
                    machineAcc = machineAcc;
    end
```

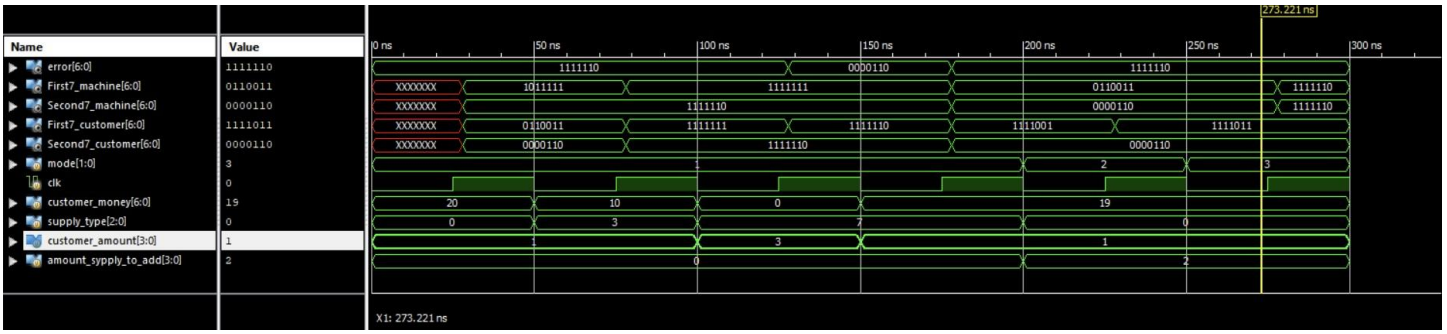
For the last part we pass our values which are inputted or have been read from the file. Please note that we pass the price and quantity of the supply type ordered by the customer or owner, rather than the arrays of prices and quantities.

```
Customer CUSTOMER (
    .mode(mode),
    .clk(clk),
    .price(prices[supply_type]),
    .amount(customer_amount),
    .money(customer_money),
    .mahcineAcc(machineAcc),
    .supply(supply[supply_type]),
    .redLight(redLight1),
    .machineAcc_out(machineAcc_out),
    .supply_out(supply_outC),
        .remaining_money(remaining_money)
    );
Owner_charge OWNER1 (
    .mode(mode),
    .clk(clk),
    .amount(amount_sypply_to_add),
    .supply(supply[supply_type]),
    .redLight(redLight2),
    .supply_out(supply_outO)
    );
Owner_retrieve OWNER2 (
    .mode(mode),
    .clk(clk),
    .machineAcc(machineAcc),
    .totalMoneyRetrieve(history),
    .redLight(redLight3),
    .totalMoneyRetrieve_out(totalMoneyRetrieve_out)
            );
display FirstDisplay (
    .clk(clk),
    .data(machineAcc_out),
    .first7S(First7_machine),
    .second7S(Second7_machine)
    );
display SecondDisplay (
    .clk(clk),
    .data(remaining_money),
    .first7S(First7_customer),
    .second7S(Second7_customer)
    );
       assign redLight = redLight1 | redLight2 | redLight3;
display2 REDLIGHT (
    .clk(clk),
```

```
    .data(redLight),
    .out(error)
    );
```

The last redlight as I pointed in the “table 1” will calculate the result of the error status.

7.TestBench and Results “The final step of our journey”



Input file:

00001100110
00100001111
01011110001
01110110010
10010001000
10101100110
11001100110
11101100110

-The first 4 clocks are utilized for clocking in the Customer mode.

Inputs	Changes	Outputs
Mode = 1 customer _money = 20 Supply_type = 0 customer _amount = 1	type 0 supply quantity changed from 6 to 5.	Error = 0 Machine money = 6 Remaining money = 14
Mode = 1 customer _money = 10 Supply_type = 3 customer _amount = 1	type 3 supply quantity changed from 11 to 10.	Error = 0 Machine money = 8 Remaining money = 8
Mode = 1 customer _money = 0 Supply_type = 7 customer _amount = 1	—	Error = 1 Machine money = 8 Remaining money = 0
Mode = 1 customer _money = 19 Supply_type = 7 customer _amount = 1	type 3 supply quantity changed from 6 to 5.	Error = 0 Machine money = 14 Remaining money = 13

-the 5th clock is utilized for clocking in the Owner mode 1.

Mode = 2 Supply_type = 0 Amount_supply_to_add = 2	type 0 supply quantity changed from 5 to 7.	—

-the 6th clock is utilized for clocking in the Owner mode 2.

Mode = 3	—	Machine money = 0

"It's a great end to a challenging journey."