Yifan Zhu

Final lab – Self Design

Project

Section 001

Goal:

The purpose of this experiment is to examine the Verilog coding learned in this semester by designing a personal project. Students should be proficient in using the FSM, RTL or ASM learned this semester to write a verilog code, and verify it with the Nexys A7 board.

Task :

The personal project I designed is a vending machine, which is an FSM. We can use clk to select the product for transaction by inserting coins or banknotes. If you pay an amount higher than the price of the product, you can get change before the transaction is completed. Transactions can be canceled at any time and the full amount refunded.

Detailed Design:

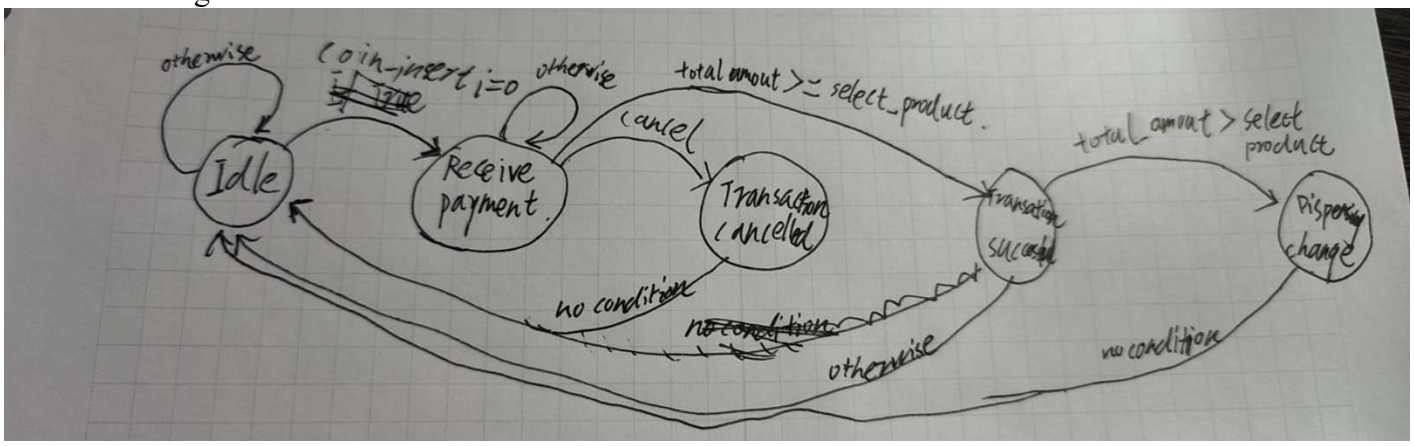

Figure 1

Let me first talk about the basic logic of fsm. I set up 5 states: Idle, Receiving Payment, Transaction Cancelled, Transaction Successful, and Dispensing Change. First we are going to put coins into the vending machine. When we put it in, the number of coins or banknotes will start to accumulate and the total accumulated amount will be calculated. If the amount received is greater than or equal to the product, the system will enter the transaction success state and the trader can get it. merchandise and give change. If the cancel transaction button is pressed during the payment process, the system will refund all inserted coins.

Design Verification:

Next is the verification part, I wrote a test file. I have provided a variety of possible cases to test and all functions work properly. Then I set cancel to SW0, coin_insert to SW2-3, select_product to SW6-8.SW15 in the xdc file. transaction_success is set to led0, change_led is set to led2-5.total_amount_led is set to 12-15.reset is set to BTNU. Then I uploaded the code to Nexys for testing and everything worked fine.

Conclusion

This experiment went very smoothly. I successfully used Nexy7 to create a vending machine.

Resource	Utilization	Available	Utilization %
LUT	26	63400	0.04
FF	12	126800	0.01
IO	13	210	6.19
BUFG	1	32	3.13

Figure2

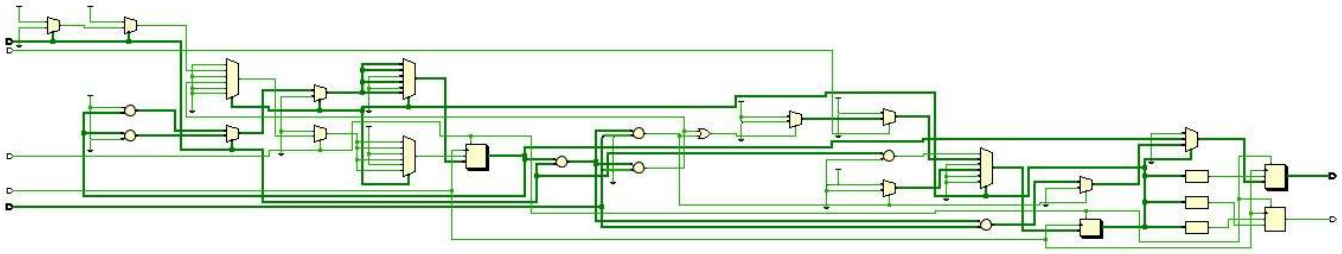

Figure3

#codeing

```
module finaltask2(
    input clk,
    input reset,
    input cancel,
    input [1:0] coin_insert, // 00;no insert 01;insert 0.5 10;insert 1
    input [2:0] select_product, // 00: no, 01: 0.5, 10: 1, 11:1.5, 100:2, 101:2.5, 110:3
    output reg transaction_success,
    output reg [3:0] change_led // LEDs to display change
);


parameter Idle = 3'b000,
          Receiving_Payment = 3'b001,
          Transaction_Cancelled = 3'b010,
          Transaction_Successful = 3'b011,
          Dispensing_Change = 3'b100;

reg [2:0] current_state, next_state;
reg [3:0] total_amount; // accumulated amount
//reg [1:0] previous_coin_insert; // store previos value
//reg coin_insert_event; // label insert coin
```

```
// state logic
always @(posedge clk or posedge reset) begin
   if (reset)
      current_state = Idle;
   else
      current_state = next_state;
end

// fsm logic
always @(current_state or coin_insert or cancel or total_amount or select_product) begin
   case (current_state)
      Idle:
         if (coin_insert != 0)
            next_state = Receiving_Payment;
         else
            next_state = Idle;

      Receiving_Payment:
         if (cancel)
            next_state = Transaction_Cancelled;
         else if (total_amount + coin_insert == select_product | total_amount + coin_insert > select_product)
            next_state = Transaction_Successful;
         else
            next_state = Receiving_Payment;

      Transaction_Cancelled:
         next_state = Idle;

      Transaction_Successful:

         if (total_amount + coin_insert > select_product)
            next_state = Dispensing_Change;
         else
            next_state = Idle;

      Dispensing_Change:
         next_state = Idle;

      default:
         next_state = Idle;
   endcase
end

// ?????????
always @(posedge clk) begin
   if (reset) begin
      total_amount <= 0;
      transaction_success <= 0;
      change_led <= 0;
   end else begin
      case (current_state)
         Idle: begin
```

```
            transaction_success <= 0;
            change_led <= 0;
        end

        Receiving_Payment: begin
        if (coin_insert == 2'b01)
            total_amount = total_amount + 4'b0001; // 0.5 unit
        else if (coin_insert == 2'b10)
            total_amount = total_amount + 4'b0010; // 1 unit
        end


        Transaction_Cancelled: begin
            change_led <= total_amount;
            total_amount <= 0;
        end

        Transaction_Successful: begin
            transaction_success <= 1;
            if (total_amount + coin_insert == select_product)
                total_amount = 0;
        end

       Dispensing_Change: begin
            if (total_amount + coin_insert > select_product)
                change_led <= (total_amount + coin_insert) - select_product; // Dispense change
            else
                change_led <= 0; // No change to dispense

            total_amount <= 0;
        end
    endcase
end
end
endmodule

#testfile
module finalsim2(

    );

    reg clk;
    reg reset;
    reg cancel;
    reg [1:0] coin_insert;
    reg [2:0] select_product;

    wire transaction_success;
    wire [3:0] change_led;

   finaltask2 Dut (
        .clk(clk),
```

```
        .reset(reset),
        .cancel(cancel),
        .coin_insert(coin_insert),
        .select_product(select_product),
        .transaction_success(transaction_success),
        .change_led(change_led)
    );
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 100 MHz clock
    end
// Clock signal generation
    initial begin
        // Initialize signals
        reset = 1;
        cancel = 0;
        coin_insert = 0;
        select_product = 0;
        #10 reset = 0; // Release reset

        // Scenario 1: Exact payment, no change needed
        #50;
        select_product = 3'b110; // Select product worth 1 unit
        #10 coin_insert = 2'b01; // Insert 1 unit
        #10 coin_insert = 2'b10; // Stop inserting coins
        #10 coin_insert = 2'b01;
        #10 coin_insert = 2'b10;
        #20 coin_insert = 2'b00;
        //#30; // Wait for transaction to complete

        #50;
        select_product = 3'b110; // Select product worth 1 unit
        #10 coin_insert = 2'b10; // Insert 1 unit
        #10 coin_insert = 2'b10; // Stop inserting coins
        #10 coin_insert = 2'b10;
        #20 coin_insert = 2'b00;

        // Scenario 3: Transaction cancellation
        #50;
        select_product = 3'b011; // Select product worth 1.5 units
        #10 coin_insert = 2'b10; // Insert 1 unit
        #10 cancel = 1;          // Cancel transaction
        #10 cancel = 0;          // Reset cancel signal
        #20 coin_insert = 2'b00; // Stop inserting coins
        #20 reset = 1;
        #20 reset = 0;

        #60;
        select_product = 3'b101; // Select product worth 1 unit
        #10 coin_insert = 2'b10;
        #10 coin_insert = 2'b01;
        #10 coin_insert = 2'b01;
```

```
        #10 coin_insert = 2'b10;
        #20 coin_insert = 2'b00;


    end

    task resetInputs;
    begin
        cancel = 0;
        coin_insert = 0;
        select_product = 0;
        #10;
    end
    endtask

endmodule
```