NAME:PRATIK PATIL

USN: 02FE21BEC063

ROLL NO: 58

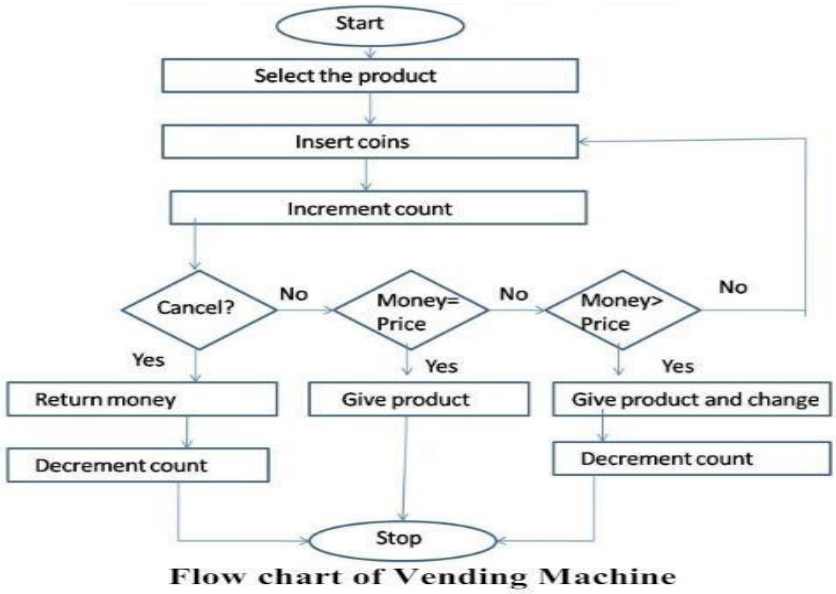
PROBLEM STATEMENT

Develop a Verilog HDL based design for a vending machine, controller system. the vending machine should support multiple products, each with its own price and quantity . the system should be allow users to select a product, insert coins, and receive the selected product along with any change owed. additionally, the vending machine should have the capability to handle errors such as insufficient change, out- of stock items, and invalid product selections. the design should be modular, allowing for easy integration of additional features such as product inventory monitoring and sales tracking.

Vending Machine - Verilog HDL

- The design of the vending machine was accomplished through the creation of a Finite State Machine (FSM) model, which defined the machine's different states, inputs, and outputs, as well as the transitions between the states.
- The FSM model was implemented using Verilog code, which defined the different states and their corresponding logic for accepting coins, dispensing products, and returning change.
- The project involved designing a vending machine that could dispense four different products with varying prices and has the additional feature of returning change when a higher denomination coin was inserted.
- This project required knowledge of Verilog, FSMs, and digital design principles and provided a challenging and rewarding opportunity to better understand how vending machines operate.

Blockdiagram



Flow chart of Vending Machine

Design description

- Designing a Verilog-based vending machine controller system is quite complex, and it's not something that can be fully detailed within this conversation.
- This project required knowledge of Verilog, FSMs, and digital design principles and provided a challenging and rewarding opportunity to better understand how vending machines operate.
- The completed project was a functional and efficient vending machine that could dispense products and return change with ease.

Code

```
`timescale 1ns / 1ps

module VendingMachine(
    input clk,                  // Clock signal
    input rst,                  // Reset signal
    input [2:0] product_code,     // Product selection input signal
    input online_payment,         // Online payment signal
```

```
    input start,                // Start signal
    input cancel,               // Cancel signal
    input [6:0] total_coin_value, // Single coin input representing the total value of coins inserted

    output reg [3:0] state,             // Output signal to indicate the current state
    output reg dispense_product, // Output signal to dispense the product
    output reg [6:0] return_change,  // Output signal to return the change
    output reg [6:0] product_price // Output signal to indicate the price of the selected product
);

// Internal states of the vending machine
localparam IDLE_STATE = 4'b0000;
localparam SELECT_PRODUCT_STATE = 4'b0001;
localparam PEN_SELECTION_STATE = 4'b0010;
localparam NOTEBOOK_SELECTION_STATE = 4'b0011;
localparam COKE_SELECTION_STATE = 4'b0100;
localparam LAYS_SELECTION_STATE = 4'b0101;
localparam WATER_BOTTLE_SELECTION_STATE = 4'b0110;
localparam DISPENSE_AND_RETURN_STATE = 4'b0111;

// Parameters for product prices
parameter WATER_BOTTLE_PRICE = 7'd20;
parameter LAYS_PRICE = 7'd35;
parameter COKE_PRICE = 7'd30;
parameter PEN_PRICE = 7'd15;
parameter NOTEBOOK_PRICE = 7'd50;
```

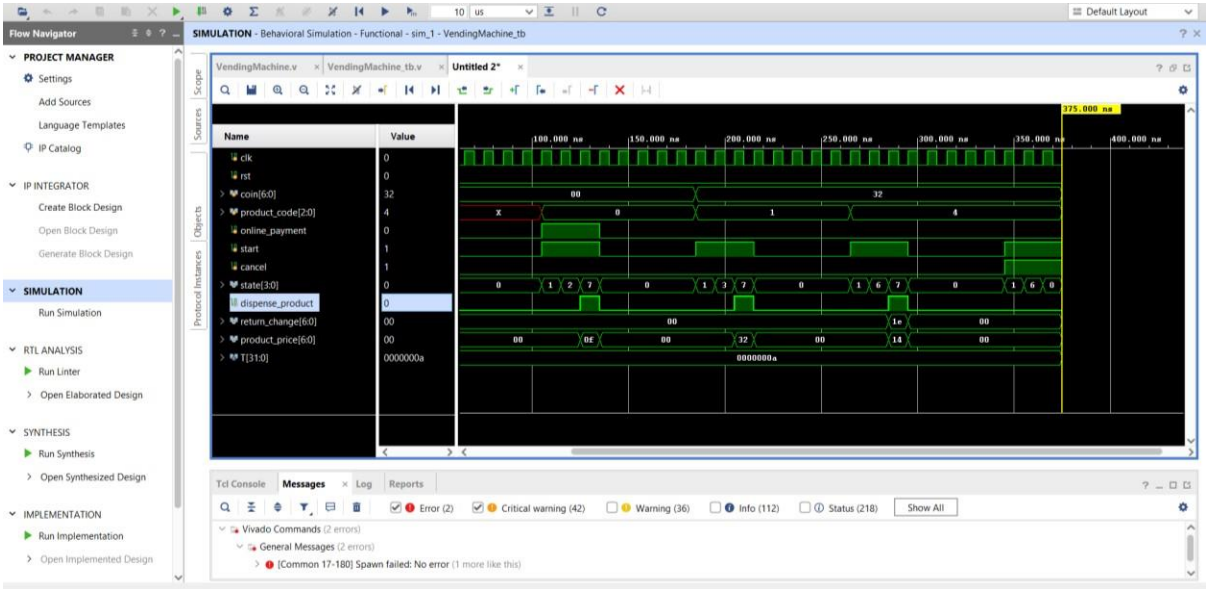
```
// Internal signals
reg [3:0] current_state;
reg [3:0] next_state;
reg [6:0] product_price_reg;
reg [6:0] return_change_reg;

// Sequential State Registers
always @(posedge clk or posedge rst) begin
    if (rst) begin
        current_state <= IDLE_STATE;
        product_price_reg <= 0;
        return_change_reg <= 0;
    end else begin
        current_state <= next_state;
        product_price_reg <= product_price_reg; // No change during the same state
        return_change_reg <= return_change_reg; // No change during the same state
    end
end

// State transition logic
always @(*) begin
    case (current_state)
        IDLE_STATE: begin
            if (start)
      next_state = SELECT_PRODUCT_STATE;
            else if (cancel)
```

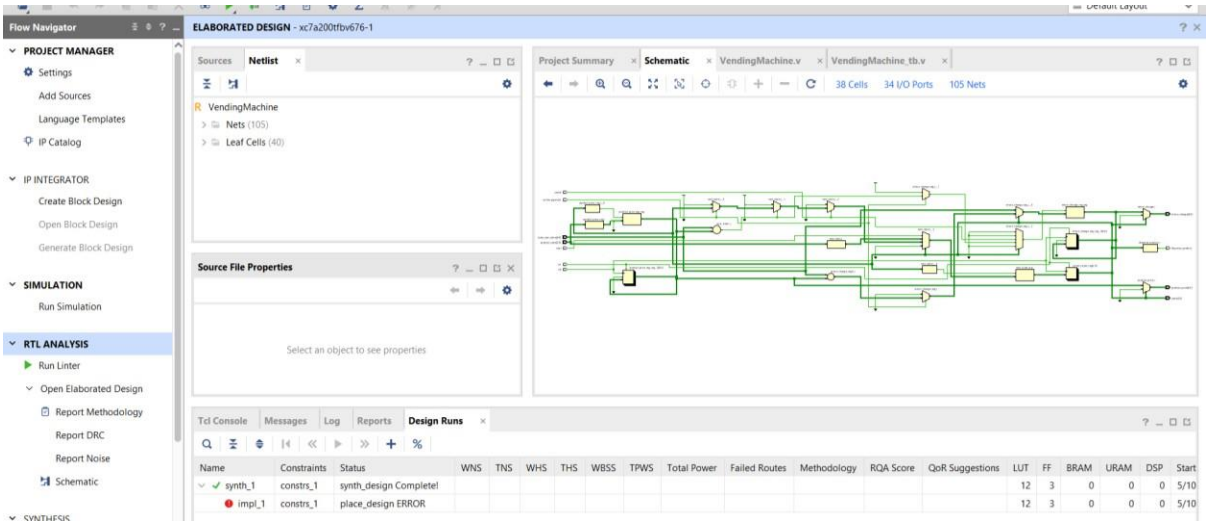
```
            next_state = IDLE_STATE;
        else
            next_state = IDLE_STATE;
    end
    SELECT_PRODUCT_STATE: begin
        case (product_code)
            3'b000: begin
                next_state = PEN_SELECTION_STATE; // Pen is selected
                product_price_reg = PEN_PRICE;
            end
            3'b001: begin
                next_state = NOTEBOOK_SELECTION_STATE; // Notebook is selected
                product_price_reg = NOTEBOOK_PRICE;
            end
            3'b010: begin
                next_state = COKE_SELECTION_STATE; // Coke is Selected
                product_price_reg = COKE_PRICE;
            end
            3'b011: begin
                next_state = LAYS_SELECTION_STATE; // Lays is selected
                product_price_reg = LAYS_PRICE;
            end
            3'b100: begin
                next_state = WATER_BOTTLE_SELECTION_STATE; // Water bottle is
selected
                product_price_reg = WATER_BOTTLE_PRICE;
            end
```

```
default: begin
next_state = IDLE_STATE; // Invalid product selection, go back to IDLE
product_price_reg = 0;
end
endcase
end
pen_selection_state,notebook_selection_state,    coke_selection_state,lays_selection_state, water_bottle_selection_state: begin
if (cancel) begin
next_state = IDLE_STATE;
return_change_reg = total_coin_value;
end
else if (total_coin_value >= product_price_reg)
next_state = DISPENSE_AND_RETURN_STATE;
else if (online_payment)
next_state = DISPENSE_AND_RETURN_STATE;
else
next_state = current_state; // Stay in the current state until enough money or online payment
end
DISPENSE_AND_RETURN_STATE: begin
next_state = IDLE_STATE;
if (online_payment)
return_change_reg = 0; // No return change in case of online payment
else if (total_coin_value >= product_price_reg)
return_change_reg = total_coin_value - product_price_reg;
end
endcase
```

```
end
always @(*) begin
state = current_state;
case (current_state)
DISPENSE_AND_RETURN_STATE: begin
dispense_product = 1;
return_change = return_change_reg;
product_price = product_price_reg;
end
default: begin
dispense_product = 0;
return_change = 0;
product_price = 0; // Set to 0 when not in DISPENSE_AND_RETURN_STATE
end
endcase
end
endmodule
```

SIMULATION





CONCLUSION

- the Verilog-based vending machine controller system employs a modular design to efficiently manage product selection, coin validation, change dispensing, and error handling**.**

REFLECTION

- This approach enables easy integration of additional features like product inventory monitoring and sales tracking, enhancing the vending machine's adaptability and scalability."