Project 86: Vending Machine

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma , Ayush Jain , Gati Goyal, Nikunj Agrawal

Documentation Specialist: Dhruv Patel & Nandini Maheshwari

Contents

1 Project Overview

A vending machine is an automated device designed to dispense items such as snacks, beverages, or other products in exchange for coins or currency. In this project, the vending machine is modeled using SystemVerilog, a hardware description language. The design incorporates state machines to manage the machine's operations, including collecting coins, dispensing selected items, and returning change.

The code simulates a vending machine that:

- Accepts coins of specified values.
- Allows users to select items from a predefined menu.
- Dispenses items if sufficient balance is available.
- Returns the appropriate change for overpayments.

2 Vending Machine

2.1 Key Components of Vending Machine

1. Finite State Machine (FSM):

- The vending machine is modeled as an FSM with four states: IDLE, COLLECT, DISPENSE, and RETURN_CHANGE.
- Transitions between states depend on user inputs and system conditions.

2. Inputs:

- clk: Clock signal for synchronization.
- reset: Resets the machine to its initial state.
- coin: Value of the coin inserted.
- item_select: ID of the item to be purchased.

3. Outputs:

- change: Amount to be returned to the user.
- dispense: Signal to indicate item dispensing.
- balance: Current balance in the machine.

4. Data Storage:

An array stores predefined item prices corresponding to item IDs.

4. Control Logic:

Determines actions like accepting coins, calculating change, and dispensing items.

2.2 Working of Vending Machine

Initialization:

- Item prices are predefined in an array during the initial block.
- The reset signal sets the machine to the IDLE state, clears the balance, and resets all outputs.

Idle State (IDLE):

- The machine waits for a coin to be inserted.
- When a coin is detected (coin ¿ 0), it transitions to the COLLECT state.

Collect State (COLLECT):

- The machine adds the inserted coin's value to the balance.
- It checks if the balance is sufficient to purchase the selected item (item_price[item_select]).
- If sufficient:

    If overpayment, transitions to RETURN_CHANGE.

    If exact payment, transitions to DISPENSE.

- If insufficient, remains in the COLLECT state, waiting for additional coins.

Dispense State (DISPENSE):

- Activates the dispense signal, indicating the item is released to the user.
- Resets the balance to 0.
- Returns to the IDLE state, ready for the next transaction.

Return Change State (RETURN_CHANGE):

- Calculates the excess amount to be returned (change = balance + coin - item_price[item_select]).
- Outputs the calculated change.
- Transitions to the DISPENSE state to complete the transaction.

Balance Update:

- The balance is updated in the COLLECT state as coins are inserted.
- It is reset to 0 in the DISPENSE and RETURN_CHANGE states.

Outputs:

- dispense: Indicates the item is being released.
- change: Provides the calculated change to the user.
- balance: Displays the current accumulated balance.

2.3 RTL Code

Listing 1: Vending Machine

```


module vending_machine (
    input  logic clk,
    input  logic reset,
    input  logic [3:0] coin,          // Input coin value (in cents: 5, 10, 25)
    input  logic select,              // Select signal for an item
    input  logic [1:0] item_code,     // Code for the selected item
    output logic dispense,            // Dispense signal
    output logic [3:0] change         // Change returned
);

    typedef enum logic [1:0] {IDLE, COLLECT, DISPENSE} state_t;
    state_t current_state, next_state;

    logic [6:0] total_amount; // Total collected amount in cents
    logic [6:0] item_price;   // Price of the selected item

    // Define prices for items (Item 0: 30, Item 1: 50, Item 2: 70, Item 3: 90)
    always_comb begin
        case (item_code)
            2'd0: item_price = 7'd30;
            2'd1: item_price = 7'd50;
            2'd2: item_price = 7'd70;
            2'd3: item_price = 7'd90;
            default: item_price = 7'd0;
        endcase
    end

    // State transition logic
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            current_state <= IDLE;
            total_amount <= 7'd0;
            dispense <= 1'b0;
            change <= 4'd0;
        end else begin
            current_state <= next_state;
            if (current_state == COLLECT && coin > 0)
                total_amount <= total_amount + coin;
            if (current_state == DISPENSE) begin
                dispense <= 1'b1;
                change <= total_amount - item_price;
                total_amount <= 7'd0;
            end else begin
                dispense <= 1'b0;
                change <= 4'd0;
            end
        end
    end

    // Next state logic
    always_comb begin
```

```
        case (current_state)
            IDLE: next_state = (select) ? COLLECT : IDLE;
            COLLECT: next_state = (total_amount >= item_price) ?
               DISPENSE : COLLECT;
            DISPENSE: next_state = IDLE;
        endcase
    end

endmodule
```

2.4 Testbench

Listing 2: Vending Machine

```

module vending_machine_tb;

    logic clk;
    logic reset;
    logic [3:0] coin;
    logic select;
    logic [1:0] item_code;
    logic dispense;
    logic [3:0] change;

    // DUT instantiation
    vending_machine dut (
        .clk(clk),
        .reset(reset),
        .coin(coin),
        .select(select),
        .item_code(item_code),
        .dispense(dispense),
        .change(change)
    );

    // Clock generation
    initial clk = 0;
    always #5 clk = ~clk; // 10ns clock period

    // Test sequence
    initial begin
        $dumpfile("vending_machine.vcd");
        $dumpvars(0, vending_machine_tb);

        reset = 1; select = 0; coin = 0; item_code = 2'd0;
        #10 reset = 0;

        // Select item 1 (price = 50 cents) and insert coins
        #10 select = 1; item_code = 2'd1;
        #10 select = 0; coin = 4'd10; // Insert 10 cents
        #10 coin = 4'd25;             // Insert 25 cents
        #10 coin = 4'd10;             // Insert 10 cents

        // Wait for dispensing
        #10 coin = 4'd0;
        #20;

```

```
        // Select item 2 (price = 70 cents) and insert coins
        #10 select = 1; item_code = 2'd2;
        #10 select = 0; coin = 4'd25; // Insert 25 cents
        #10 coin = 4'd25;             // Insert 25 cents
        #10 coin = 4'd25;             // Insert 25 cents

        // Wait for dispensing
        #10 coin = 4'd0;
        #20 $stop;
    end

    // Monitor signals
    initial begin
        $monitor("Time=%0t | Item_Code=%0d | Coin=%0d |
           Total_Amount=%0d | Dispense=%b | Change=%0d",
                 $time, item_code, coin, dut.total_amount, dispense,
                    change);
    end

endmodule
```

3 Results

3.1 Simulation

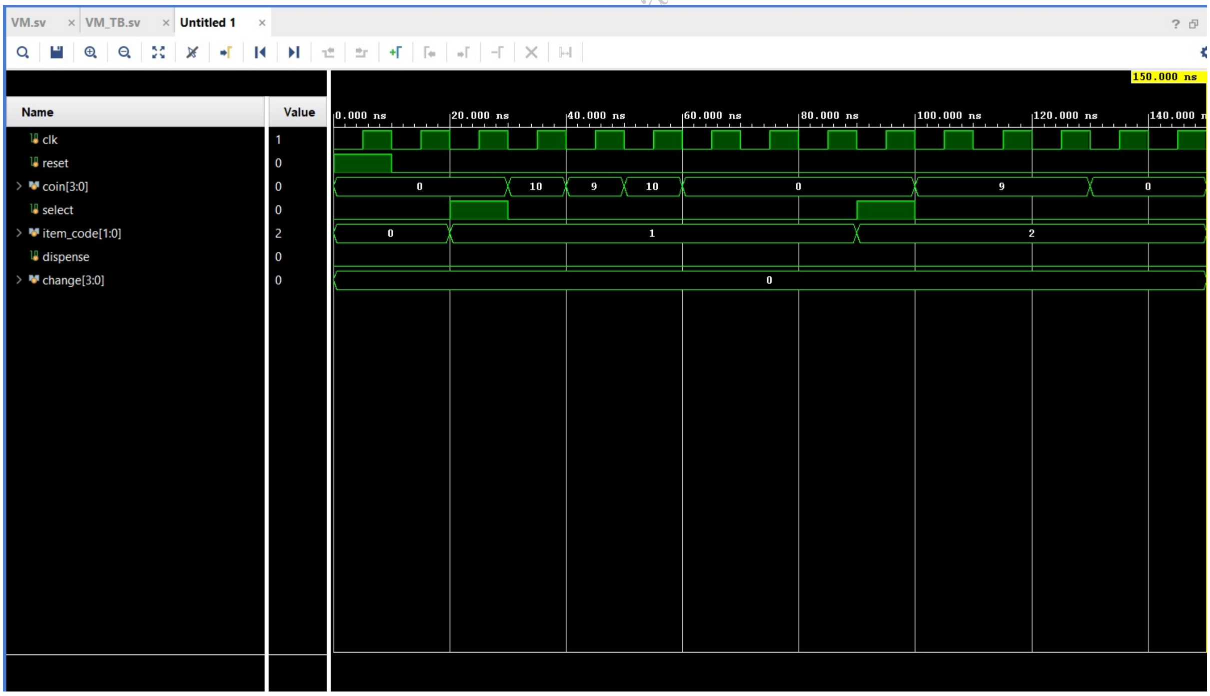

Figure 1: Simulation of Vending Machine

3.2 Schematic

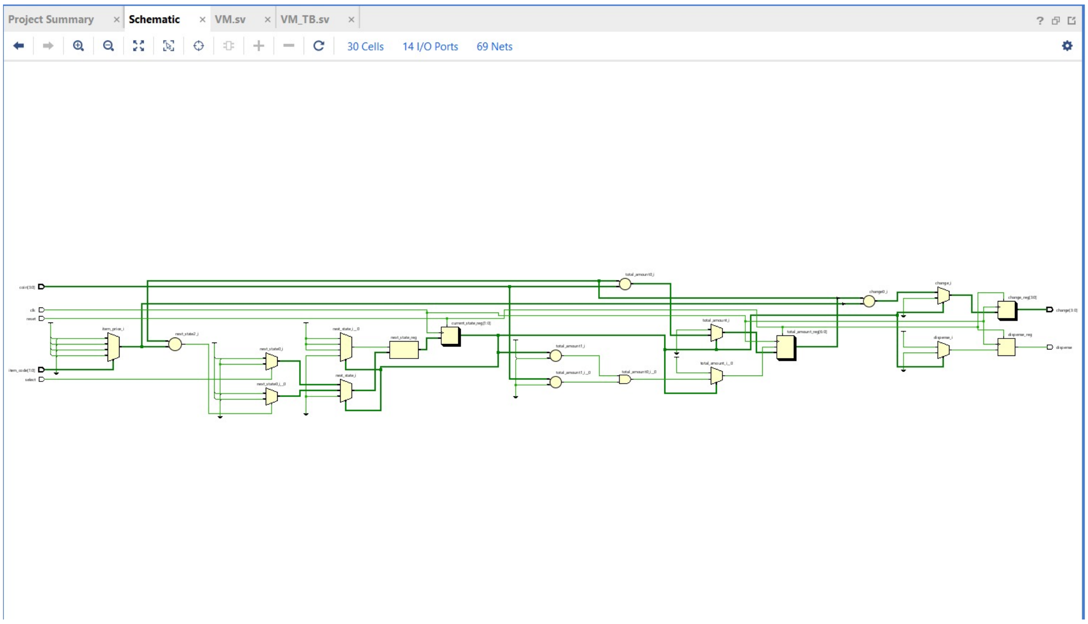

Figure 2: Schematic of Vending Machine

3.3 Synthesis Design

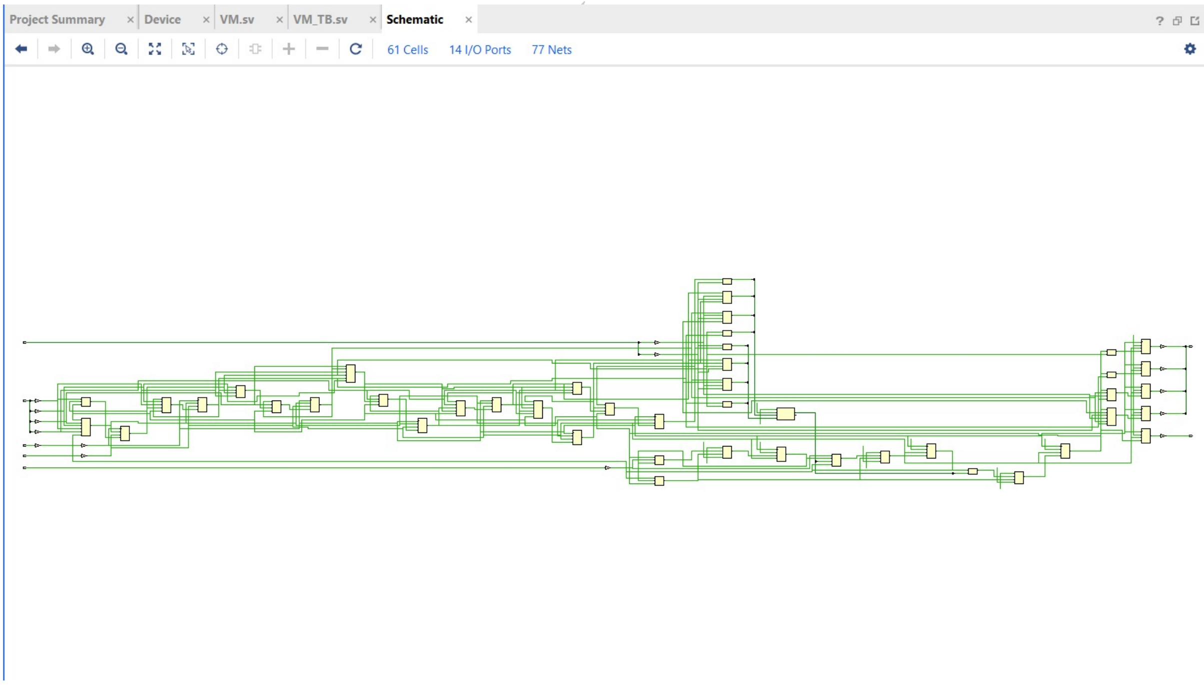

Figure 3: Synthesis Design of Vending Machine

4 Advantages of Vending Machine

Automation: Reduces the need for human operators.

Accuracy: Ensures precise calculation of balance and change.

Efficiency: Handles transactions quickly and reliably.

Scalability: Can support multiple items with varying prices.

User Convenience: Provides a simple interface for purchasing.

5 Disadvantages of Vending Machine

Complexity: Requires robust design to handle various scenarios.

Fault Tolerance: Susceptible to hardware failures, e.g., coin mechanism issues.

Limited Adaptability: Difficult to add new items or modify prices dynamically.

Power Dependency: Inoperative during power outages.

6 Applications of Vending Machine

Retail: Dispensing snacks, drinks, and other items in public places.

Transport: Ticket vending in buses, trains, and metros.

Banking: Cash withdrawal and deposit kiosks.

Education: Automated stationary and book dispensers.

Healthcare: Medicine vending machines in hospitals and pharmacies.

7 Conclusion

The vending machine project demonstrates a practical application of finite state machines in hardware design. By systematically managing states and transitions, the machine efficiently handles tasks like coin collection, item dispensing, and change return. This modular and scalable design is well-suited for real-world implementations, ensuring reliability and user convenience. The project highlights essential digital design principles, making it an excellent exercise in hardware modeling using SystemVerilog.

8 FAQs

Q1 : What is the significance of using a finite state machine in this project?
A: FSM simplifies the control flow by breaking operations into states, each with specific tasks, enabling systematic handling of inputs and outputs.

Q2 : How does the machine calculate the change to be returned?
A: The machine calculates change as balance + coin - item_price[item_select] in the RETURN_CHANGE state.

Q3 : What are the advantages of using SystemVerilog for this project?
A: SystemVerilog provides a structured, hardware-oriented language that supports concurrent processing and ease of testbench creation.

Q4 : How can the design be extended to handle more items?
A: By increasing the size of the item_price array and modifying the item_select input width.

Q5 : How is the machine reset to its initial state?
A: The reset signal initializes the state to IDLE, clears the balance, and resets outputs.

Q6 : What happens if an invalid item is selected?
A: The machine remains in the COLLECT state until a valid item is selected.

Q7 : How is the simulation tested?
A: Using a SystemVerilog testbench that applies various coin and item inputs to validate state transitions and outputs.

Q8 : What are potential failure points in the design?
A: Errors in state transitions, balance updates, or incorrect item price configurations.

Q9 : Can this design handle simultaneous coin insertions?
A: No, the current design assumes sequential coin inputs; modifications are needed for parallel handling.

Q10 : How is synthesis different from simulation?
A: Simulation tests functional correctness in software, while synthesis translates the design into hardware logic for FPGA/ASIC implementation.

Q11 : What modifications are needed for a multi-currency vending machine?
A: Introduce a currency detection module and update the balance computation logic.

Q12 : Why is the balance reset in DISPENSE and RETURN_CHANGE states?
A: To prepare the machine for the next transaction, ensuring no residual balance affects future operations.

Q13 : How can this design be optimized for low power?
A: Implement power-down modes for inactive states and use efficient clock gating.

Q14 : What challenges might arise during synthesis?
A: Resource constraints on FPGA, timing violations, and logic optimization issues.

Q15 : What is the role of always_ff and always_comb in this design?
A: always_ff handles sequential logic (state updates), while always_comb manages combinational logic (next-state determination).