EL467
Digital Programming

Vending Machine

Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar, Gujarat

Date of Submission
22/11/2024

Table of Contents

1. Abstract

This project focuses on the design and implementation of a vending machine using Verilog HDL that demonstrates core digital system design principles. The vending machine offers four items priced at 15, 20, 25, and 30 coins and accepts only 5 and 10 coin denominations. Users can select an item by providing the appropriate amount. If the inserted amount exceeds the item's price, the machine calculates and dispenses the correct change using 5 and 10 coin coins. The project employs a Finite State Machine (FSM) for control, ensuring smooth operation and logical transitions between states. Simulated in a Verilog environment, the vending machine was validated for accuracy and reliability, showcasing its ability to operate effectively under diverse user inputs.

2. Introduction

This project addresses the fundamental operations of a vending machine, including:

- Allowing the user to select an item from four available options, priced at 15, 20, 25, and 30 coins.
- Accepting payments exclusively in the form of 5 and 10 coin denominations.
- Verifying if the amount inserted is sufficient for the chosen item.
- Dispensing the selected product and, if necessary, returning the exact change to the user.

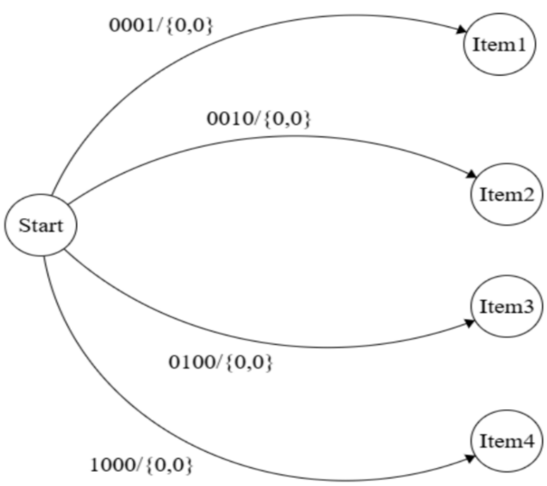
The core logic of the machine is built using FSMs, ensuring efficient and logical state transitions, such as waiting for payment, verifying inputs, dispensing items, and calculating change. The modular approach to the design enhances code readability, testing, and scalability, while Verilog simulation ensures accurate and real-time validation of the system.

The vending machine design includes the following modules:

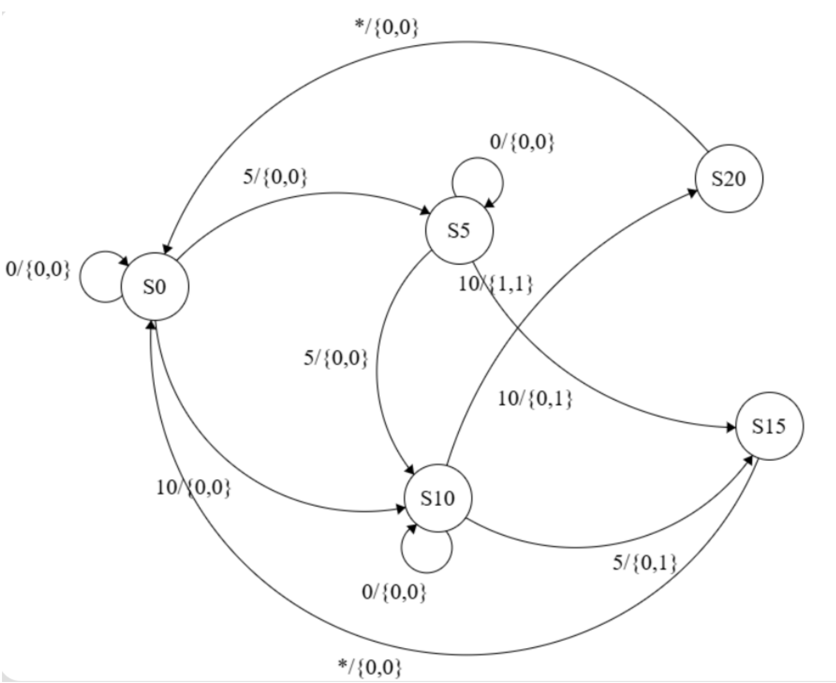
1. **Main Module:** A module that control vending machine.
2. **Item One:** A module that calculates change and whether to dispense item one or not.
3. **Item Two:** A module that calculates change and whether to dispense item two or not.
4. **Item Three:** A module that calculates change and whether to dispense item three or not.
5. **Item Four:** A module that calculates change and whether to dispense item four or not.

3. Finite State Machine

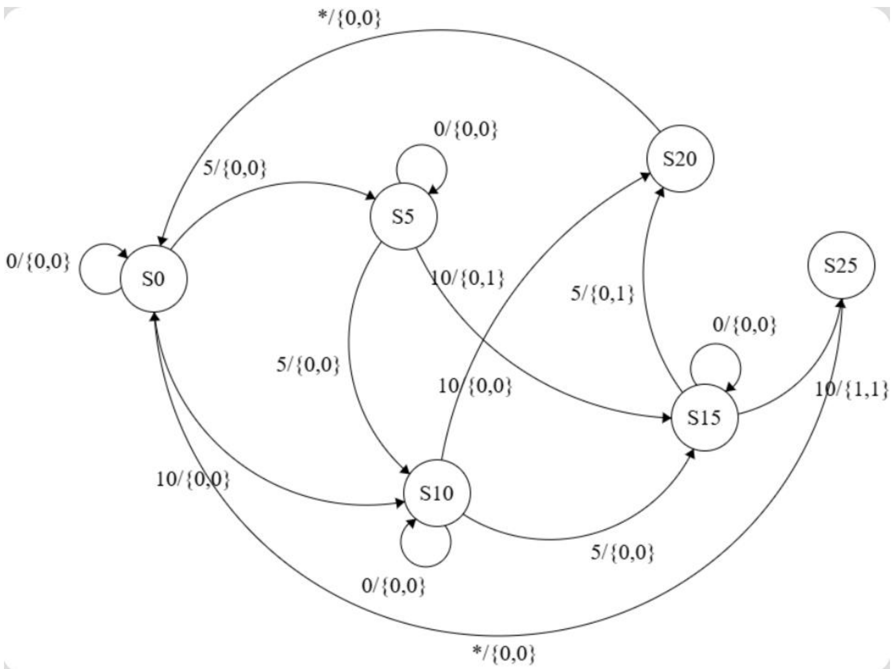
3.1 Main Module



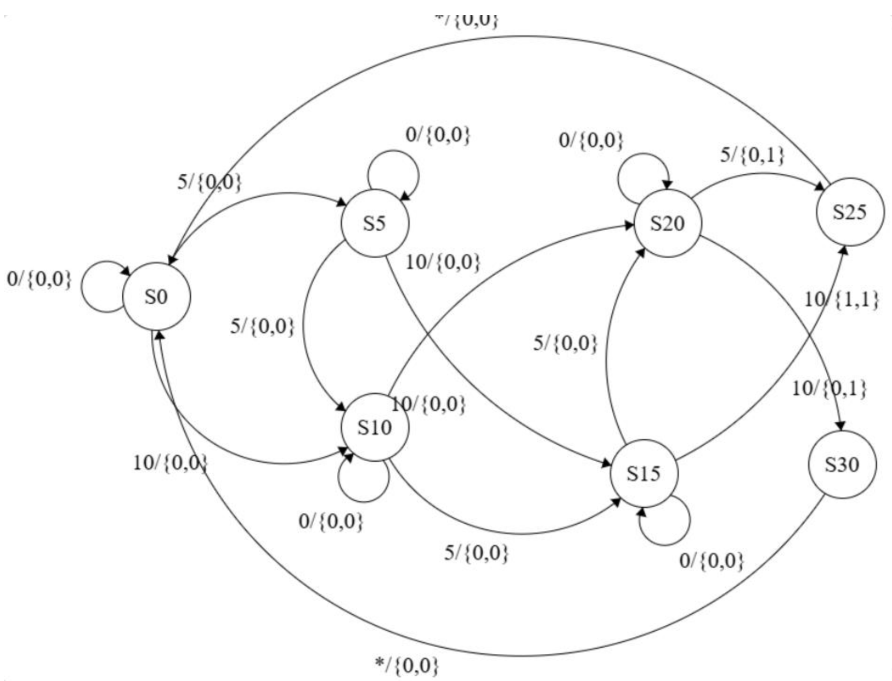
3.2 Item One



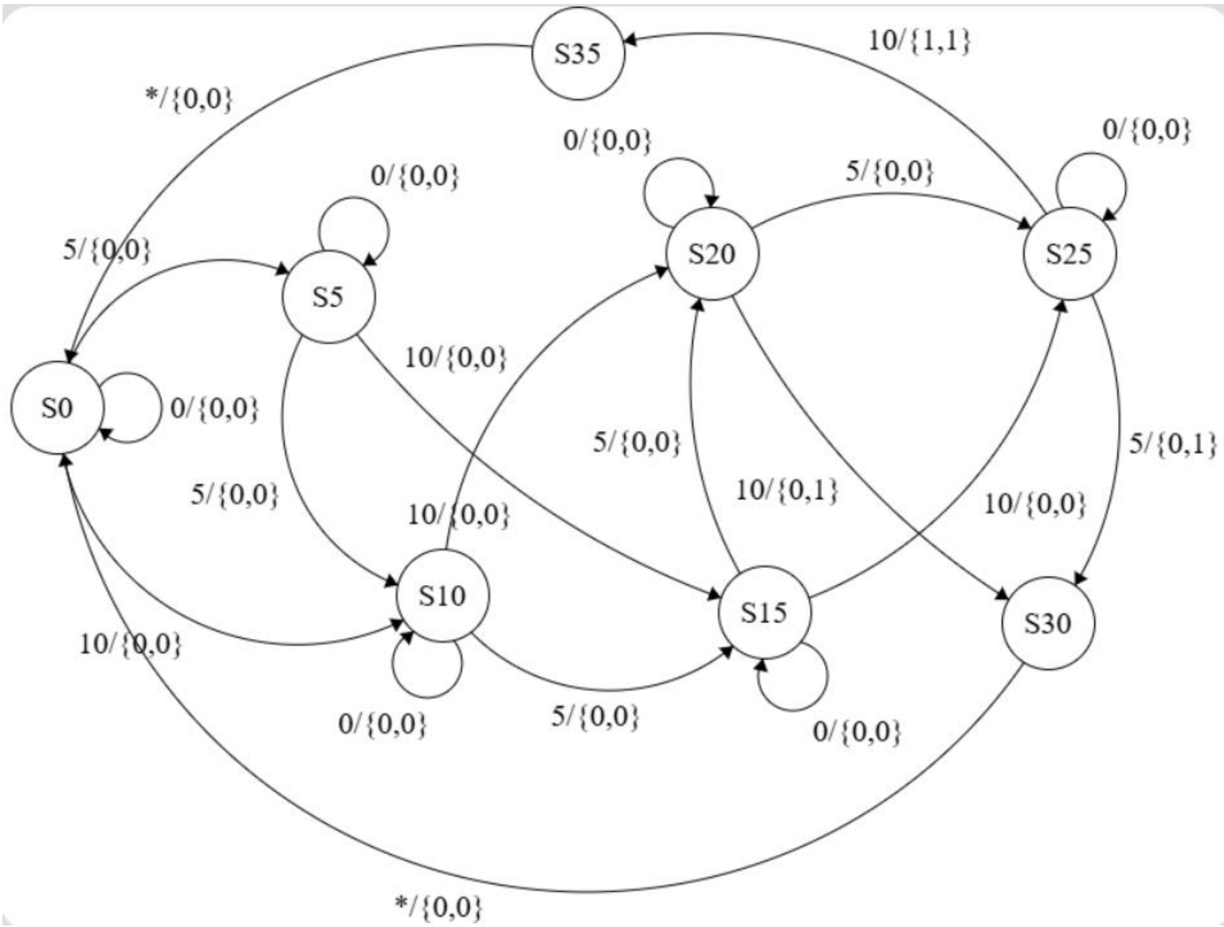
3.3 Item Two



3.4 Item Three



3.5 Item Four


S35
10/{1,1}
*/{0,0}
0/{0,0}
0/{0,0}
5/{0,0}
0/{0,0}
5/{0,0}
S20
S25
S5
10/{0,0}
S0
0/{0,0}
5/{0,0}
5/{0,1}
5/{0,0}
10/{0,1}
10/{0,0}
10/{0,0}
S10
S15
S30
10/{0,0}
0/{0,0}
5/{0,0}
0/{0,0}
*/{0,0}

4. Verilog Implementation

Code:

4.1 Main Module

```
// Main Vending Machine Module
module VendingMachine(
    input a, b, c, d, nickel_in, dime_in, clock,
    output reg nickel_out, dispense
);

    // Wires for nickel_out and dispense from item modules
    wire No1, No2, No3, No4;
    wire D1, D2, D3, D4;

    // Item selection encoded as a 4-bit number
    wire [3:0] item_number;
    assign item_number = {d, c, b, a};

    // Instantiate modules for each item
    Item_One   IO(.nickel_in(nickel_in), .dime_in(dime_in), .clock(clock),
.nickel_out(No1), .dispense(D1));
    Item_Two   ITW(.nickel_in(nickel_in), .dime_in(dime_in), .clock(clock),
.nickel_out(No2), .dispense(D2));
    Item_Three ITH(.nickel_in(nickel_in), .dime_in(dime_in), .clock(clock),
.nickel_out(No3), .dispense(D3));
    Item_Four  IF(.nickel_in(nickel_in), .dime_in(dime_in), .clock(clock),
.nickel_out(No4), .dispense(D4));

    // Output values based on the selected item
    always @(*) begin
        case (item_number)
            4'b0001: begin
                nickel_out = No1;
                dispense = D1;
            end
            4'b0010: begin
                nickel_out = No2;
                dispense = D2;
            end
            4'b0100: begin
                nickel_out = No3;
```

```
                dispense = D3;
            end

            4'b1000: begin
                nickel_out = No4;
                dispense = D4;
            end
            default: begin
                nickel_out = 0;  // Default values
                dispense = 0;
            end
        endcase
    end
endmodule
```

4.2 Item One

```
module Item_One(nickel_in, dime_in, clock, nickel_out, dispense);
    input nickel_in, dime_in, clock;
    output reg nickel_out, dispense;
    reg [4:0] current_state, next_state;

    // Defining the states
    localparam  S0  = 5'b00001,
                S5  = 5'b00010,
                S10 = 5'b00100,
                S15 = 5'b01000,
                S20 = 5'b10000;

    // A state will change when there is a positive edge in clock or reset
    always @(posedge clock) begin
        current_state <= next_state;
    end

    // Defining the relationships among the states
    always @(*) begin
        case (current_state)
            S0: begin
                if (nickel_in) begin
                    next_state = S5;
                    {nickel_out, dispense} = 2'b00;
                end else if (dime_in) begin
                    next_state = S10;
                    {nickel_out, dispense} = 2'b00;
                end else begin
                    next_state = S0;
                    {nickel_out, dispense} = 2'b00;
                end

            end

            S5: begin
                if (nickel_in) begin
                    next_state = S10;
                    {nickel_out, dispense} = 2'b00;
                end else if (dime_in) begin
                    next_state = S15;
                    {nickel_out, dispense} = 2'b01;
```

```
                end else begin
                    next_state = S5;
                    {nickel_out, dispense} = 2'b00;
                end
            end

            S10: begin
                if (nickel_in) begin
                    next_state = S15;
                    {nickel_out, dispense} = 2'b01;
                end else if (dime_in) begin
                    next_state = S20;
                    {nickel_out, dispense} = 2'b11;
                end else begin
                    next_state = S10;
                    {nickel_out, dispense} = 2'b00;
                end
            end

            S15: begin
                next_state = S0;
                {nickel_out, dispense} = 2'b00;
            end

            S20: begin
                next_state = S0;
                {nickel_out, dispense} = 2'b00;
            end

            default: begin
                next_state = S0;
                {nickel_out, dispense} = 2'b00;
            end
        endcase
    end
endmodule
```

4.3 Item Two

```
module Item_Two(nickel_in, dime_in, clock, nickel_out, dispense);
    input nickel_in, dime_in, clock;
    output reg nickel_out, dispense;

    reg [5:0] current_state, next_state;

    localparam  S0  = 6'b000001,
                S5  = 6'b000010,
                S10 = 6'b000100,
                S15 = 6'b001000,
                S20 = 6'b010000,
                S25 = 6'b100000;

    // State update logic
    always @(posedge clock) begin
        current_state <= next_state;
    end

    // Combinational logic for next state and outputs
    always @(*) begin
        // Default assignments
        next_state = current_state;  // Default to hold the current state
        {nickel_out, dispense} = 2'b00; // Default output values

        case (current_state)
            S0: begin
                if (nickel_in) begin
                    next_state = S5;
                end else if (dime_in) begin
                    next_state = S10;
                end
            end

            S5: begin
                if (nickel_in) begin
                    next_state = S10;
                end else if (dime_in) begin
                    next_state = S15;
                end
            end
```

```
            S10: begin
                if (nickel_in) begin
                    next_state = S15;

                end else if (dime_in) begin
                    next_state = S20;
                    {nickel_out, dispense} = 2'b01; // Output when transitioning
to S20
                end
            end

            S15: begin
                if (nickel_in) begin
                    next_state = S20;
                    {nickel_out, dispense} = 2'b01; // Output when transitioning
to S20
                end else if (dime_in) begin
                    next_state = S25;
                    {nickel_out, dispense} = 2'b11; // Output when transitioning
to S25
                end
            end

            S20: begin
                next_state = S0;
            end

            S25: begin
                next_state = S0;
            end

            default: begin
                next_state = S0;
            end
        endcase
    end
endmodule
```

4.4 Item Three

```
module Item_Three(nickel_in, dime_in, clock, nickel_out, dispense);
    input nickel_in, dime_in, clock;
    output reg nickel_out, dispense;

    reg [6:0] current_state, next_state;

    localparam  S0  = 7'b0000001,
                S5  = 7'b0000010,
                S10 = 7'b0000100,
                S15 = 7'b0001000,
                S20 = 7'b0010000,
                S25 = 7'b0100000,
                S30 = 7'b1000000;

    // State update logic
    always @(posedge clock) begin
        current_state <= next_state;
    end

    // Combinational logic for next state and outputs
    always @(*) begin
        // Default assignments
        next_state = current_state;  // Default to hold the current state
        {nickel_out, dispense} = 2'b00; // Default output values

        case (current_state)
            S0: begin
                if (nickel_in) begin
                    next_state = S5;
                end else if (dime_in) begin
                    next_state = S10;
                end
            end

            S5: begin
                if (nickel_in) begin
                    next_state = S10;
                end else if (dime_in) begin
                    next_state = S15;
```

```
                end
            end

            S10: begin
                if (nickel_in) begin
                    next_state = S15;
                end else if (dime_in) begin
                    next_state = S20;
                end
            end

            S15: begin
                if (nickel_in) begin
                    next_state = S20;
                end else if (dime_in) begin
                    next_state = S25;
                    {nickel_out, dispense} = 2'b01; // Output when transitioning
to S25
                end
            end

            S20: begin
                if (nickel_in) begin
                    next_state = S25;
                    {nickel_out, dispense} = 2'b01; // Output when transitioning
to S25
                end else if (dime_in) begin
                    next_state = S30;
                    {nickel_out, dispense} = 2'b11; // Output when transitioning
to S30
                end
            end

            S25: begin
                next_state = S0;
            end

            S30: begin
                next_state = S0;
            end

            default: begin
```

```
                next_state = S0;
            end
        endcase
    end
endmodule
```

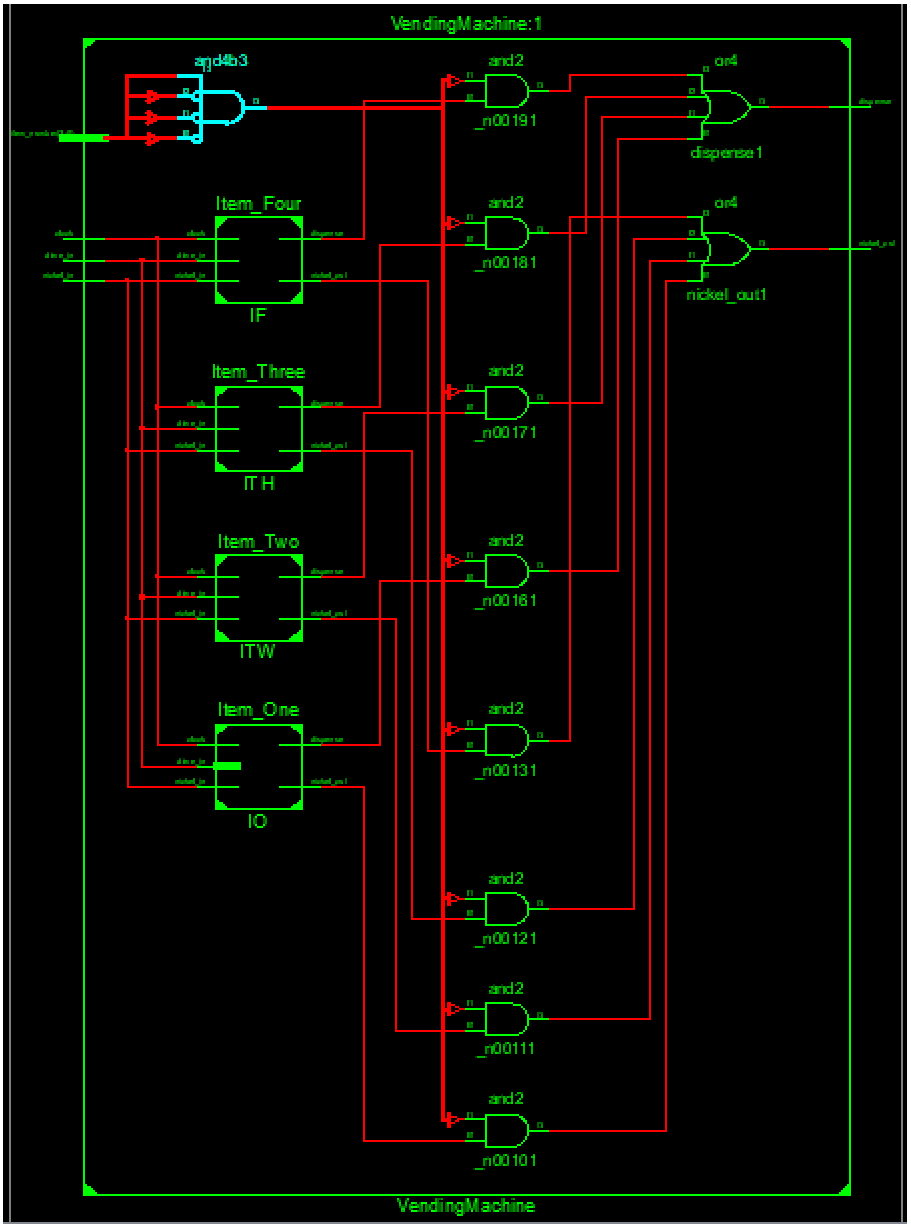
4.5 Item Four

```
module Item_Four(nickel_in, dime_in, clock, nickel_out, dispense);
    input nickel_in, dime_in, clock;
    output reg nickel_out, dispense;

    reg [7:0] current_state, next_state;

    localparam  S0  = 8'b00000001,
                S5  = 8'b00000010,
                S10 = 8'b00000100,
                S15 = 8'b00001000,
                S20 = 8'b00010000,
                S25 = 8'b00100000,
                S30 = 8'b01000000,
                S35 = 8'b10000000;

    // State update logic
    always @(posedge clock) begin
        current_state <= next_state;
    end

    // Combinational logic for next state and outputs
    always @(*) begin
        // Default assignments
        next_state = current_state;  // Default to hold the current state
        {nickel_out, dispense} = 2'b00; // Default output values

        case (current_state)
            S0: begin
                if (nickel_in) begin
                    next_state = S5;
                end else if (dime_in) begin
```

```
                next_state = S10;
            end
        end

        S5: begin
            if (nickel_in) begin
                next_state = S10;
            end else if (dime_in) begin
                next_state = S15;
            end
        end

        S10: begin
            if (nickel_in) begin
                next_state = S15;
            end else if (dime_in) begin
                next_state = S20;
            end
        end

        S15: begin
            if (nickel_in) begin
                next_state = S20;
            end else if (dime_in) begin
                next_state = S25;
                {nickel_out, dispense} = 2'b01; // Output when transitioning
to S25
            end
        end

        S20: begin
            if (nickel_in) begin
                next_state = S25;
                {nickel_out, dispense} = 2'b01; // Output when transitioning
to S25
            end else if (dime_in) begin
                next_state = S30;
                {nickel_out, dispense} = 2'b01; // Output when transitioning
to S30
            end
        end
```

```
            S25: begin
                if (nickel_in) begin
                    next_state = S30;
                    {nickel_out, dispense} = 2'b01; // Output when transitioning
to S30
                end else if (dime_in) begin
                    next_state = S35;
                    {nickel_out, dispense} = 2'b11; // Output when transitioning
to S35
                end
            end

            S30: begin
                next_state = S0;
            end

            S35: begin
                next_state = S0;
            end

            default: begin
                next_state = S0;
            end
        endcase
    end
endmodule
```

5. RTL Schematic



6. Test Bench

```
`timescale 1ns / 1ps

module tb_VendingMachine;

    // Testbench signals
    reg [3:0] item_number;
    reg nickel_in, dime_in, clock;
    wire nickel_out, dispense;

    // Instantiate the VendingMachine
    VendingMachine vm (
        .item_number(item_number),
        .nickel_in(nickel_in),
        .dime_in(dime_in),
        .clock(clock),
        .nickel_out(nickel_out),
        .dispense(dispense)
    );

    // Clock generation
    initial begin
        clock = 0;
        forever #1 clock = ~clock;  // 10 ns clock period
    end

    // Test procedure
    initial begin
        // Initialize signals
        nickel_in = 0;
        dime_in = 0;
        item_number = 4'b0000;

        // Reset the machine
        // #10 reset = 0;  // Release reset
        // #10 reset = 1;  // Assert reset again

        // Test case 1: Select Item 1 and check outputs
        item_number = 4'b0001;  // Select Item 1
        #10 nickel_in = 1;  // Insert nickel
```

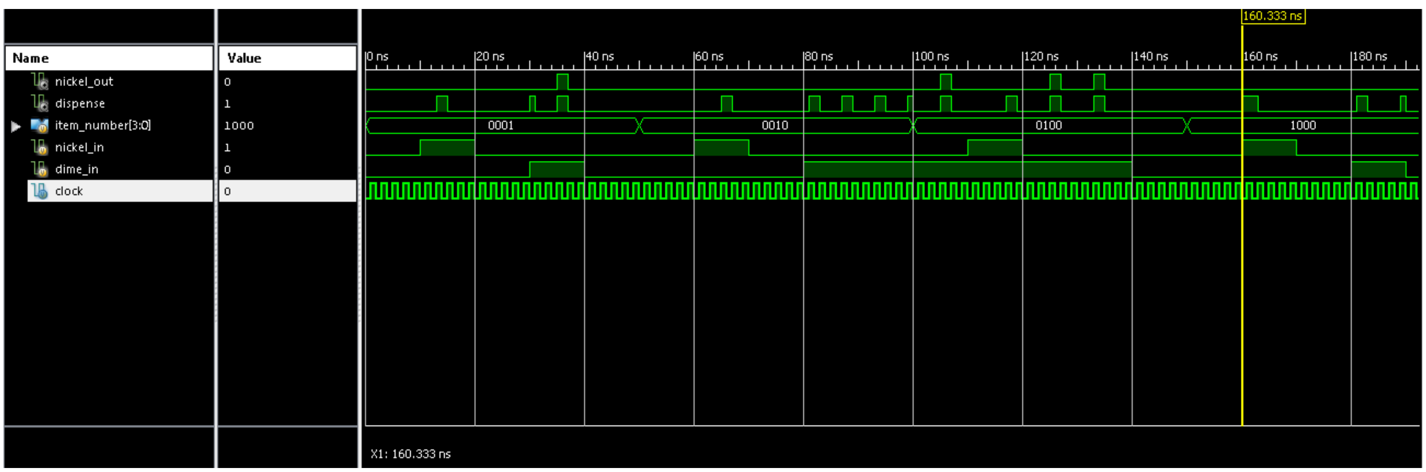
```
        #10 nickel_in = 0;  // Release nickel
        #10 dime_in = 1;    // Insert dime
        #10 dime_in = 0;    // Release dime
        #10;

        // Check outputs for Item 1
        $display("Item 1: nickel_out = %b, dispense = %b", nickel_out,
dispense);

        // Test case 2: Select Item 2
        item_number = 4'b0010;  // Select Item 2
        #10 nickel_in = 1;      // Insert nickel
        #10 nickel_in = 0;      // Release nickel
        #10 dime_in = 1;        // Insert dime
        #10 dime_in = 1;        // Release dime
        #10;

        // Check outputs for Item 2
        $display("Item 2: nickel_out = %b, dispense = %b", nickel_out,
dispense);

        // Test case 3: Select Item 3
        item_number = 4'b0100;  // Select Item 3
        #10 nickel_in = 1;      // Insert nickel
        #10 nickel_in = 0;      // Release nickel
        #10 dime_in = 1;        // Insert dime
        #10 dime_in = 0;        // Release dime
        #10;

        // Check outputs for Item 3
        $display("Item 3: nickel_out = %b, dispense = %b", nickel_out,
dispense);

        // Test case 4: Select Item 4
        item_number = 4'b1000;  // Select Item 4
        #10 nickel_in = 1;      // Insert nickel
        #10 nickel_in = 0;      // Release nickel
        #10 dime_in = 1;        // Insert dime
        #10 dime_in = 0;        // Release dime
        #10;

        // Check outputs for Item 4
```

```
        $display("Item 4: nickel_out = %b, dispense = %b", nickel_out, dispense);

        // Test case 5: No item selected
        item_number = 4'b0000;  // No item selected
        #10;
        $display("No item: nickel_out = %b, dispense = %b", nickel_out, dispense);

    end

endmodule
```

7. Simulation

Waveforms:

Simulation waveforms demonstrate the transitions between FSM states and the correctness of outputs.



8. Results

The vending machine design was successfully implemented and verified using simulation tools. Key results include:

- Accurate Transaction Processing: The machine correctly identifies the amount inserted, verifies if it meets the selected item's price, and dispenses the product accordingly.
- Change Dispensing: When excess payment is made, the machine returns the correct change in 5 and 10 coin denominations.

9. Conclusion & Future Scope

- Dynamic Pricing: Allow prices to be updated via a user interface or software.
- Support for More Denominations: Enable acceptance of additional coins like ₹1, ₹2, or ₹50.
- Digital Payments: Integrate UPI, card readers, or NFC for cashless transactions.
- More Items: Expand to support a greater variety of products.
- Smart Change Dispensing: Optimize change returned to minimize the number of coins.