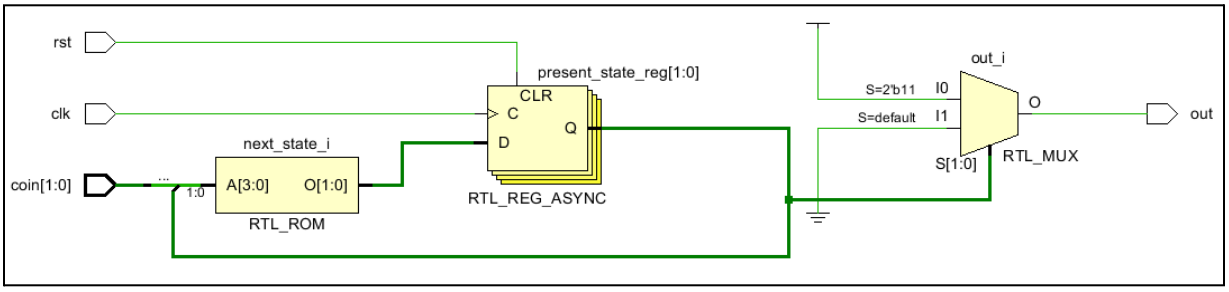
Moore Implementation Verilog Code

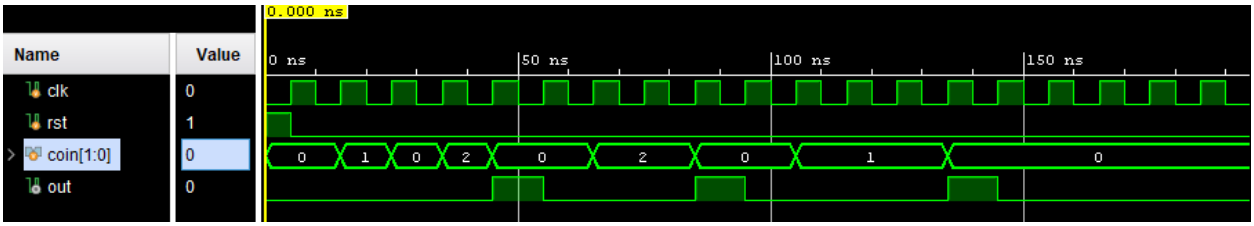
```
module moore(
    input clk, rst,
    input [1:0] coin, //00 = 0 cent, 01 = 5 cents, 10 = 10 cents
    output reg out);
localparam S0 = 2'b00, S5 = 2'b01, S10 = 2'b10, S15 = 2'b11 ;
reg [1:0] present_state, next_state;
//NEXT STATE COMBINATIONAL LOGIC (Depends upon Present State and Inputs)
always @ (*) begin
    casez ({coin,present_state})
        {2'b00,S0}: next_state = S0;
        {2'b01,S0}, {2'b00,S5}: next_state = S5;
        {2'b10,S0}, {2'b01,S5}, {2'b00,S10}: next_state = S10;
        {2'b10,S5}, {2'b01,S10}, {2'b10,S10}: next_state = S15;
        default: next_state = S0;
    endcase
end
// STATE REGISTER SEQUENTIAL LOGIC
always @ (posedge clk or posedge rst) begin
    if (rst) present_state <= S0;
    else present_state <= next_state;
end
//OUTPUT COMBINATIONAL LOGIC
always @(present_state) begin
    case (present_state)
        S15: out = 1'b1;
        default: out = 1'b0;
    endcase
end
endmodule
```



Moore Implementation Testbench

```
`timescale 1ns/1ps
module tb_moore;
    // Testbench signals
    reg clk, rst;
    reg [1:0] coin;   // 00 = 0¢, 01 = 5¢, 10 = 10¢
    wire out;
    // Instantiate DUT (Device Under Test)
    moore uut (
        .clk(clk),.rst(rst),.coin(coin),.out(out)
    );
    // Clock generation: 10 ns period (100 MHz)
    always #5 clk = ~clk;
    // Test sequence
    initial begin
        // Initialize
```

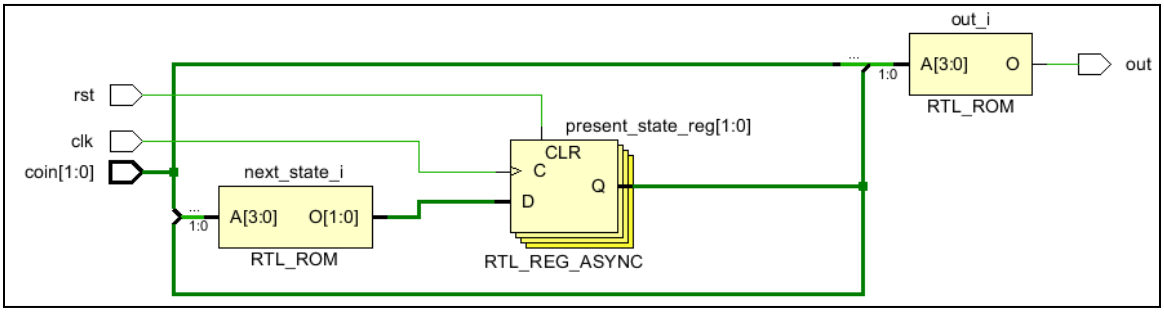
```
        clk = 0; rst = 1; coin = 2'b00;
        // Apply reset for one cycle
        @(posedge clk);
        rst = 0;
        // === Test 1: Insert 5¢ + 10¢ → should output 1 at 15¢ ===
        @(posedge clk); coin = 2'b01;   // insert 5¢
        @(posedge clk); coin = 2'b00;   // no coin
        @(posedge clk); coin = 2'b10;   // insert 10¢
        @(posedge clk); coin = 2'b00;   // FSM should output 1 here
        // === Test 2: Insert 10¢ + 10¢ → output should pulse again ===
        repeat(2) @(posedge clk);       // wait a couple cycles
        coin = 2'b10;                   // insert 10¢
        @(posedge clk); coin = 2'b10;   // another 10¢ (total 20¢)
        @(posedge clk); coin = 2'b00;   // FSM resets after dispensing
        // === Test 3: Insert 5¢ + 5¢ + 5¢ ===
        repeat(2) @(posedge clk);
        coin = 2'b01; @(posedge clk);
        coin = 2'b01; @(posedge clk);
        coin = 2'b01; @(posedge clk);   // total 15¢ now
        coin = 2'b00; @(posedge clk);
        // Finish simulation after a few more cycles
        repeat(5) @(posedge clk);
        $finish; end endmodule
```



Mealy Implementation Verilog Code:

```
module mealy(
    input clk, rst,
    input [1:0] coin, //00 = 0 cent, 01 = 5 cents, 10 = 10 cents
    output reg out);
localparam S0 = 2'b00, S5 = 2'b01, S10 = 2'b10;
reg [1:0] present_state, next_state;
//OUTPUT LOGIC & NEXT STATE COMBINATIONAL LOGIC (Both depend upon Present State and Inputs)
always @ (*) begin
    casez ({coin,present_state})
        {2'b00,S0}: begin
            next_state = S0; out = 0;
        end
        {2'b01,S0}, {2'b00,S5} : begin
            next_state = S5; out = 0;
        end
        {2'b10,S0}, {2'b01,S5}, {2'b00,S10} : begin
            next_state = S10; out = 0;
        end
        {2'b10,S5}, {2'b01,S10}, {2'b10,S10}: begin
            next_state = S0; out = 1;
        end
        default: begin
            next_state = S0; out = 0;
        end
    endcase
end
// STATE REGISTER SEQUENTIAL LOGIC
always @ (posedge clk or posedge rst) begin
```

```
    if (rst) present_state <= S0;
    else present_state <= next_state;
end
endmodule
```




```
`timescale 1ns/1ns
module tb_mealy;
    // Testbench signals
    reg clk, rst;
    reg [1:0] coin;   // 00 = 0¢, 01 = 5¢, 10 = 10¢
    wire out;
    // Instantiate DUT (Device Under Test)
    mealy uut (.clk(clk),.rst(rst),.coin(coin),.out(out));
    // Clock generation: 10 ns period (100 MHz)
    always #5 clk = ~clk;
    // Test sequence
    initial begin
        // Initialize
        clk = 0;rst = 1;coin = 2'b00;
        // Apply reset for one cycle
        @(posedge clk);
        rst = 0;
        // === Test 1: 5¢ + 10¢ → should output 1 when total reaches 15¢ ===
        @(posedge clk); coin = 2'b01;   // Insert 5¢
        @(posedge clk); coin = 2'b00;   // No coin
        @(posedge clk); coin = 2'b10;   // Insert 10¢ (expect out=1 here, Mealy)
        @(posedge clk); coin = 2'b00;   // No coin
        // === Test 2: 10¢ + 10¢ ===
        repeat(2) @(posedge clk);
        coin = 2'b10; @(posedge clk);   // Insert 10¢
        coin = 2'b10; @(posedge clk);   // Another 10¢ (expect out=1)
        coin = 2'b00; @(posedge clk);   // No coin
        // === Test 3: 5¢ + 5¢ + 5¢ ===
        repeat(2) @(posedge clk);
        coin = 2'b01; @(posedge clk);   // 5¢
        coin = 2'b01; @(posedge clk);   // 10¢ total
        coin = 2'b01; @(posedge clk);   // 15¢ (expect out=1)
        coin = 2'b00; @(posedge clk);
        // Finish simulation
        repeat(5) @(posedge clk);
        $finish;end endmodule
```