Design and Simulation of Voting Machine using Verilog HDL

Tejo Priya Boddu

Rajiv Gandhi University Of Knowledge Technologies, Nuzvid,521202, Andhra Pradesh, India
boddutejopriya@gmail.com

Abstract. This project presents the design and implementation of a simple digital voting machine based on Finite State Machines (FSM) using Verilog Hardware Description Language (HDL). The proposed system allows users to cast their votes for one of the four candidates using dedicated input buttons. To ensure valid voting and eliminate accidental presses, the system incorporates a hold-time validation mechanism, registering a vote only when the corresponding button is continuously pressed for a specified duration. Additionally, the design includes a display mode that enables real-time visualization of individual vote counts. The FSM architecture efficiently manages voting, debouncing, validation, and display operations, ensuring reliable and predictable system behavior. This project demonstrates the effectiveness of FSMs in control-oriented digital systems and highlights the practical application of HDL design methodologies in embedded electronic voting systems.

1 Introduction:

Voting systems are critical components of democratic processes that require high reliability, accuracy, and user-friendliness. This project aims to design a simple, efficient, and robust digital voting machine using Finite State Machines (FSM) in Verilog. The system allows users to cast votes for one of the four candidates using dedicated physical buttons.

To ensure vote integrity, a button press is only registered as a valid vote when held continuously for a specific duration, thereby minimizing the chances of false or accidental inputs. The design also incorporates a display mode that allows users or administrators to view the number of votes each candidate has received. This functionality ensures transparency and enhances user interaction.

The project uses a Mealy FSM model to manage the voting process, validate inputs, update vote counts, and efficiently control transitions. Implementing the system in Verilog highlights the application of hardware description languages in real-world control system designs and demonstrates how FSM principles can be applied to create secure and responsive electronic voting solutions.

1.1 The technology of Voting Machine.

The voting machine is designed using digital logic principles, specifically finite-state machines (FSM) implemented in Verilog HDL. It utilizes a Mealy FSM to ensure responsive vote registration based on real-time button inputs. Votes are securely counted only after a validated button hold duration, preventing accidental input. The system features a display mode to view individual candidate vote counts, enhancing transparency and reliability.

1.2 Introduction to Finite State Machines (FSM):

Finite-state machines (FSMs) are mathematical models of computation widely used to design both hardware and software systems that react to sequences of input. An FSM is defined by a finite set of states, transitions between those states based on inputs, and actions or outputs that result from those transitions. FSMs provide a structured approach to design complex control logic with predictable behavior.

FSMs are broadly classified into two types based on how outputs are generated: Mealy FSMs and Moore FSMs.

Moore FSM: In a Moore machine, the output depends only on the current state, not on the input. The output changes only when transitioning to a different state.

Mealy FSM: In a Mealy machine, the output depends on both the current state and the current input. This often results in faster system responses because outputs can change immediately in response to inputs, even without a state change. In this project, a Mealy FSM model is utilized due to its responsiveness and efficiency. In a voting machine, it is crucial to detect button presses and generate corresponding outputs (such as vote increments) immediately based on user interaction without necessarily waiting for a full state change. using a Mealy FSM: The system quickly validates and registers a vote when the correct conditions (button held for required time) are met.

Real-time display updates and transitions between voting and display modes become faster and more resource-efficient. Thus, the Mealy FSM approach provides a more dynamic and responsive behavior suitable for interactive applications like voting machines.

1.3 Working Principle:

- The working of the voting machine follows the flowchart shown above. Initially, the voting machine is in an idle state. Once a button is pressed, it selects which mode to operate. There are two modes of operation:
  If the input logic is '1', the machine enters the voting mode.
  If the input logic is '0', the machine enters the tallying (display) mode.

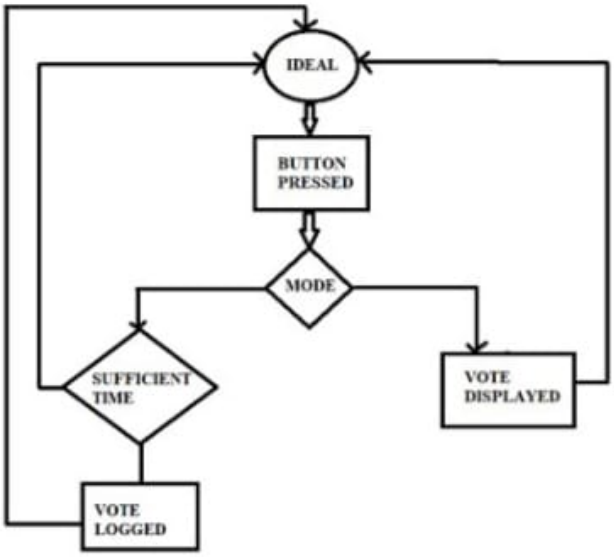

Fig. 1. Flow chart of Voting Machine

According to the selected mode, the machine either displays the vote counts or allows people to cast their votes. When entering voting mode, the voter must press and hold the candidate's button for a specified duration to successfully register their vote. If the button is not pressed for the required time, the vote will not be logged.
Additionally, a priority mechanism is included: if two buttons are pressed simultaneously, the vote is assigned to the candidate with the lower button number. For example, if buttons 2 and 3 are pressed together, Candidate 2 will receive the vote.
After completing the voting process or the display mode, the machine automatically returns to the idle state.

2 Methodology:

– The design of the voting machine mainly have one main module and three sub modules.

Main Module:
Voting Machine
Sub Modules:
button control
mode controller
vote logger

2.1 Verilog Implementation:

Functions of the Four Modules
1. Button Control:
Ensures that a vote is logged only after the corresponding button has been pressed and held for a specific duration. Prevents multiple votes from being registered for a single prolonged button press, ensuring only one vote is counted per action.
Verilog code:

```
//BUTTON CONTROL MODULE
module button_control(
input clk,rst,button,
output reg valid_vote);
reg[30:0]counter;
always @(posedge clk)
begin
    if(rst)
        counter <= 0;
    else
    begin
        if(button & counter <11)
            counter <= counter + 1;
        else if(!button)
            counter <= 0;
    end
end

always @(posedge clk)
begin
    if(rst)
        valid_vote<=1'b0;
    else if(counter == 10)
        valid_vote <=1'b1;
    else
        valid_vote <=1'b0;
end
```

```
endmodule
```

2. Vote Logger:

Records valid votes when the system is in voting mode.
Responsible for resetting all vote counts at the start of operation to ensure a clean initialization.
Verilog code:

```
module voteLogger(
input clk,rst,mode,
input cand1_vote_valid,
input cand2_vote_valid,
input cand3_vote_valid,
input cand4_vote_valid,
output reg [7:0] cand1_vote_recvd,
output reg [7:0] cand2_vote_recvd,
output reg [7:0] cand3_vote_recvd,
output reg [7:0] cand4_vote_recvd);
always @ (posedge clk)
begin
    if(rst)
    begin
        cand1_vote_recvd<=0;
        cand2_vote_recvd<=0;
        cand3_vote_recvd<=0;
        cand4_vote_recvd<=0;
    end
    else
    begin
        if(cand1_vote_valid & mode==0)
            cand1_vote_recvd <= cand1_vote_recvd + 1;
        else if(cand2_vote_valid & mode==0)
            cand2_vote_recvd <= cand2_vote_recvd + 1;
        else if(cand3_vote_valid & mode==0)
            cand3_vote_recvd <= cand3_vote_recvd + 1;
        else if(cand4_vote_valid & mode==0)
            cand4_vote_recvd <= cand4_vote_recvd + 1;
    end
end
endmodule
```

3. Mode Control:

Manages system modes with the help of LEDs for visual feedback. In voting mode, LEDs briefly light up for one second to confirm that a vote has been successfully cast. In tally (display) mode, LEDs display the number of votes for

each candidate in binary format.
Verilog code:

```
module modeControl(
input clk,rst,mode,valid_vote_casted,
input  [7:0] candidate1_vote,
input  [7:0] candidate2_vote,
input  [7:0] candidate3_vote,
input  [7:0] candidate4_vote,
input candidate1_button_press,
input candidate2_button_press,
input candidate3_button_press,
input candidate4_button_press,
output reg [7:0]leds
);
reg [30:0] counter;
always @ (posedge clk)
begin
    if(rst)
        counter <= 0;
    else if(valid_vote_casted)
        counter <= counter+1;
    else if(counter != 0 & counter < 10)
        counter <= counter+1;
    else
        counter <= 0;
end

always @(posedge clk)
begin
    if(rst)
        leds <= 0;
    else
    begin
        if(mode==0 & counter > 0)
            leds<=8'hFF;
        else if(mode==0)
            leds<=8'h00;
        else if(mode==1)
        begin
            if(candidate1_button_press)
                leds<=candidate1_vote;
            else if(candidate2_button_press)
                leds<=candidate2_vote;
            else if(candidate3_button_press)
                leds<=candidate3_vote;
```

```
            else if(candidate4_button_press)
                leds<=candidate4_vote;
        end
    end
end
endmodule
```

2.2 Testbench Design:

– This test testbench module have five test cases for different test cases.After simulating the testbench we can observe some waveforms.

```
‘timescale 1ns / 1ps
module voting_machine_tb;

  // Testbench inputs
  reg clk;
  reg rst;
  reg mode;
  reg button1;
  reg button2;
  reg button3;
  reg button4;

  // Testbench outputs
  wire [7:0] led;

  // Instantiate the voting_machine module
  voting_machine uut (
    .clk(clk),
    .rst(rst),
    .mode(mode),
    .button1(button1),
    .button2(button2),
    .button3(button3),
    .button4(button4),
    .led(led)
  );

  initial begin
   clk = 0;
   forever #5 clk = ~clk;
  end

  initial begin
    rst = 1;
```

```
    mode = 0;
    button1 = 0;
    button2 = 0;
    button3 = 0;
    button4 = 0;
  #100;

    #100 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 1; button2 = 0; button3 = 0; button4 = 0;
    #10 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 1; button2 = 0; button3 = 0; button4 = 0;
    #200 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #10 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;

    #5 rst = 0; mode = 0; button1 = 0; button2 = 1; button3 = 0; button4 = 0;
    #200 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #10 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;

    #5 rst = 0; mode = 0; button1 = 0; button2 = 1; button3 = 1; button4 = 0;
    #200 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #10 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;

    #5 rst = 0; mode = 1; button1 = 0; button2 = 1; button3 = 0; button4 = 0;
    #200 rst = 0; mode = 1; button1 = 0; button2 = 0; button3 = 1; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #10 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;

    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 1; button4 = 0;
    #200 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #10 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;
    #5 rst = 0; mode = 0; button1 = 0; button2 = 0; button3 = 0; button4 = 0;

    $finish;
end

endmodule
```

3 Results

3.1 Waveforms:

After creating the xilinx project with these two verilog modules, run the simulation and we get some waveforms to verify the working of this Voting Machine.

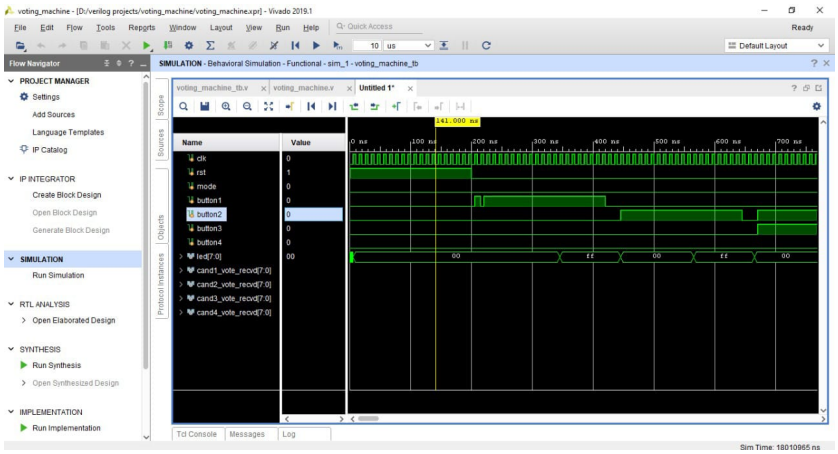

Fig. 2. IDEAL STATE

Reset condition: The simulation results verify the voting machine functionality. Initially the voting machine in ideal state.In this state every input is in logic'0'.

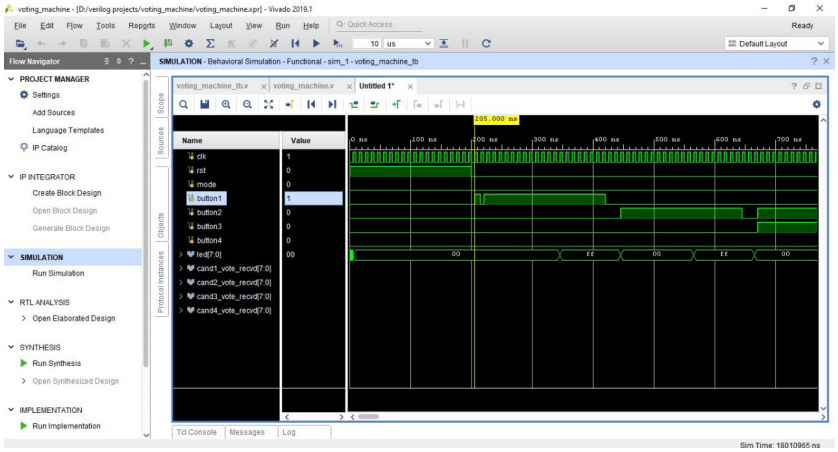

Fig. 3. INVALID VOTE

Invalid voting condition:

– In this waveform button1 has been pressed for very less i.e not sufficent time for vote to register hence the vote will not be registered to cnadidate1.Consider this vote as an invalid vote.

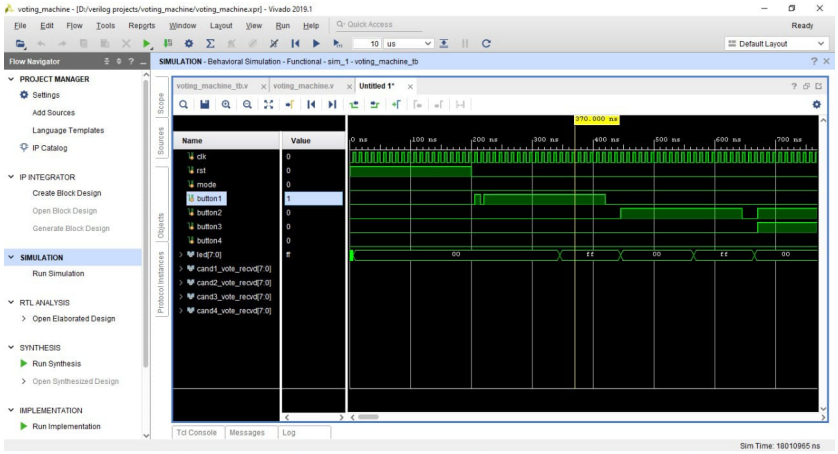

Fig. 4. FIRST VOTE FOR CANDIDATE1

Voting for candidate1 :

– In this waveform button1 has been pressed for required time hence the vote will be registered to cnadidate1.

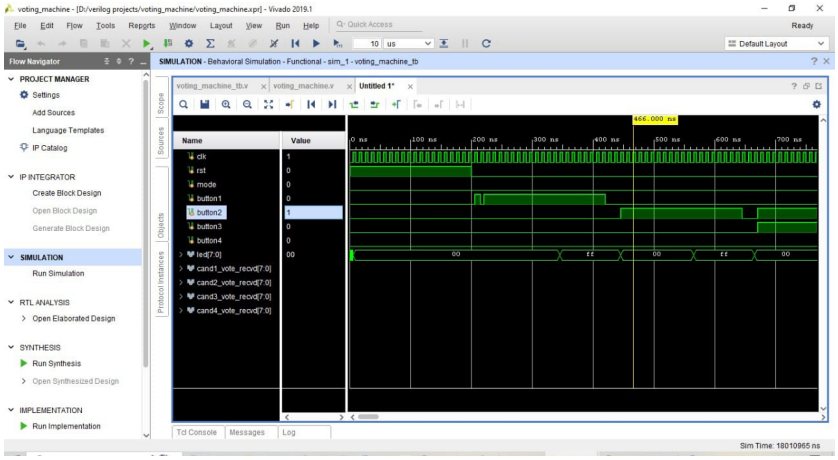

Fig. 5. FIRST VOTE FOR CANDIDATE2

Voting for candidate2 :

– In this waveform button2 has been pressed for required time hence the vote will be registered to cnadidate2.

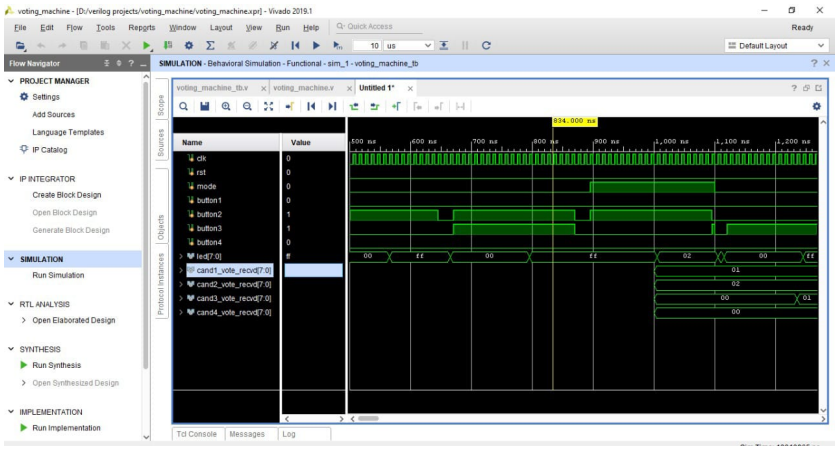

Fig. 6. SECOND VOTE FOR CANDIDATE2

Two buttons pressed simultaneously :

– In the above waveform button2 and button3 has been pressed simultaneously.According the condition we have written, the vote will register for low candidate number.Hence the vote will register for the Candidate2.

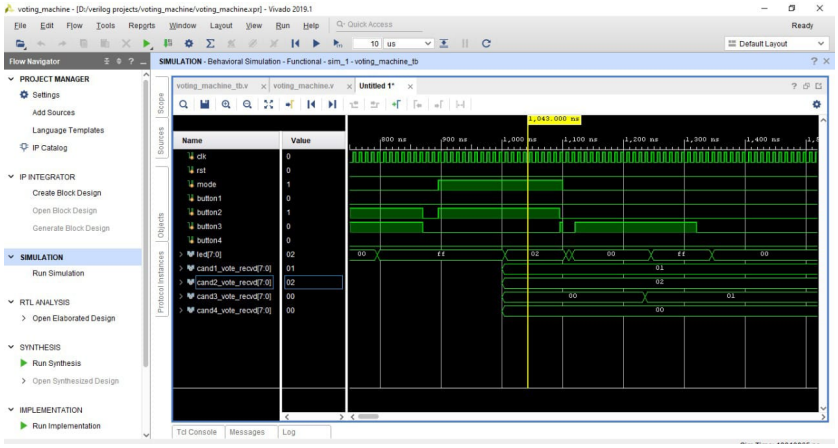

Fig. 7. TALLYING VOTES

Tallying mode:

– The mode of the machine has been changed to logic '1'.Now the machine enters into tallying mode.The above waveform shows that when the machine is in tallying mode then it will not register any vote.As we can see in the waveform that machine is in tallying mode and button2 has been pressed it will not register any vote.

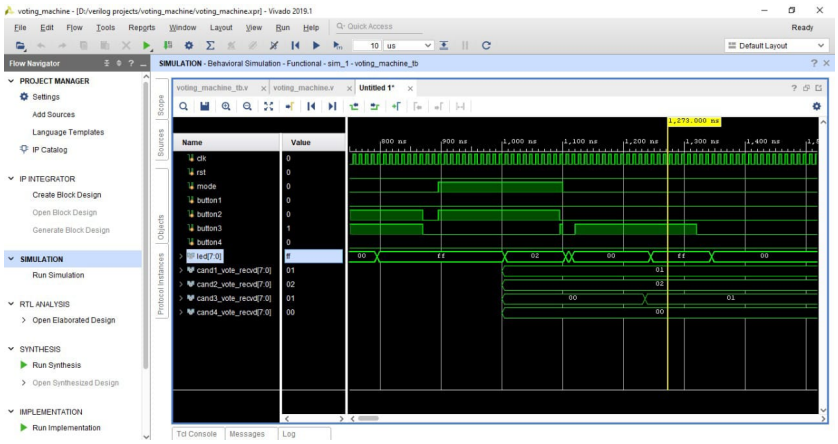

Fig. 8. FIRST VOTE FOR CANDIDATE3

Voting for candidate3 :

– The mode has again changed to logic '0'.Now the machine is in voting mode and the button3 has been pressed for 10 cycles which mean the vote been casted for candidate3.

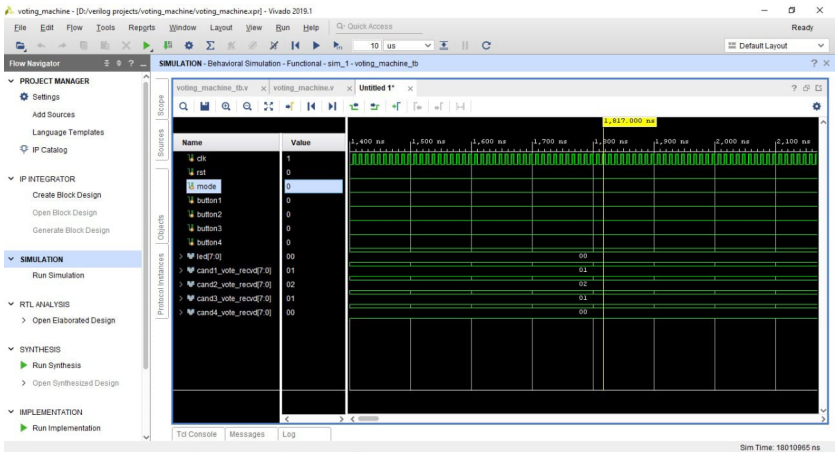

Fig. 9. FINAL VOTE COUNT

Vote Counting:

– The above waveform describe the final vote count of the four candidates.
Candidate1 has 1 vote
Candidate2 has 2 votes
Candidate3 has 1 vote
Candidate4 has 0 vote

3.2 Schematic:

After analyzing all the waveforms, open elaborated design where you can see a schematic with a module having inputs clock,reset,mode, button1, button2, button3, button4 and outputs leds.

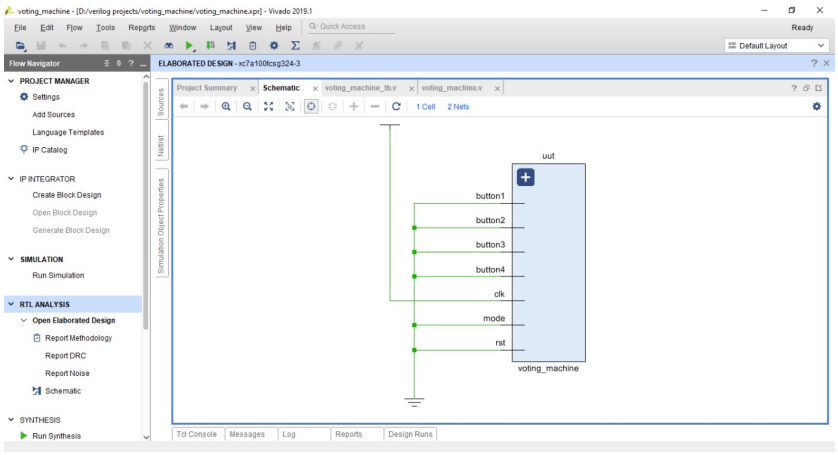

Fig. 10. Schematic of Voting Machine

4 Conclusion

The "Voting Machine using FSM" project successfully demonstrates the practical application of Finite State Machines (FSM) in a real-world system. By carefully designing the system using Verilog HDL, we ensured that voting and tallying operations are both reliable and efficient. The modular approach, including button control, vote logging, and mode control, made the design structured and easy to debug. The voting machine accurately registers valid votes by detecting button presses held for a specific duration, minimizing false triggers. It also provides immediate visual feedback through LEDs for both voting and result display modes. The system was thoroughly verified through testbench simulations, covering a variety of realistic voting scenarios. This project enhanced our understanding of FSM design, hardware description languages, and the importance of synchronization and timing in digital systems. Overall, the project met its intended goals and forms a good foundation for building more advanced and secure voting systems.

5 References

- M. Morris Mano, "Digital Design," 5th Edition, Pearson Education, 2012. (FSM concepts and digital logic design)
- Samir Palnitkar, "Verilog HDL: A Guide to Digital Design and Synthesis," 2nd Edition, Pearson, 2003. (Verilog coding practices)
- IEEE Standard 1364-2005, "IEEE Standard for Verilog Hardware Description Language."