SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC206J - VLSI Design
VI Semester, 2022-2023 (EVEN Semester)

Title of Mini Project : Voting Machine Using Verilog

Date of Submission :

Particulars	**Max. Marks**	**Marks Obtained**		
		Name: Arvind A	**Name: Prayag Sujith**	**Name: Gokul U**
		Register No. 119	**Register No. 120**	**Register No. 137**
Design Code	25			
Demo verification &viva	10			
Project Report	05			
Total	**40**			

REPORT VERIFICATION

Staff Name :

Signature :

VOTING MACHINE USING VERILOG

OBJECTIVE:

The objective of a voting machine using Verilog is to create a reliable and efficient electronic device for counting votes in an election. Verilog is a hardware description language used to design digital circuits and systems, and it is commonly used for designing electronic voting machines.The main objective of a voting machine using Verilog is to accurately and securely count votes and prevent any possibility of vote tampering or manipulation. Verilog code can be used to design a circuit that can handle the input from various voting options, such as buttons or touchscreens, and tally the results in real-time.

SOFTWARE DETAIL:

1. Xilinx ,Modelsim
2. Quetasim

ABSTRACT:

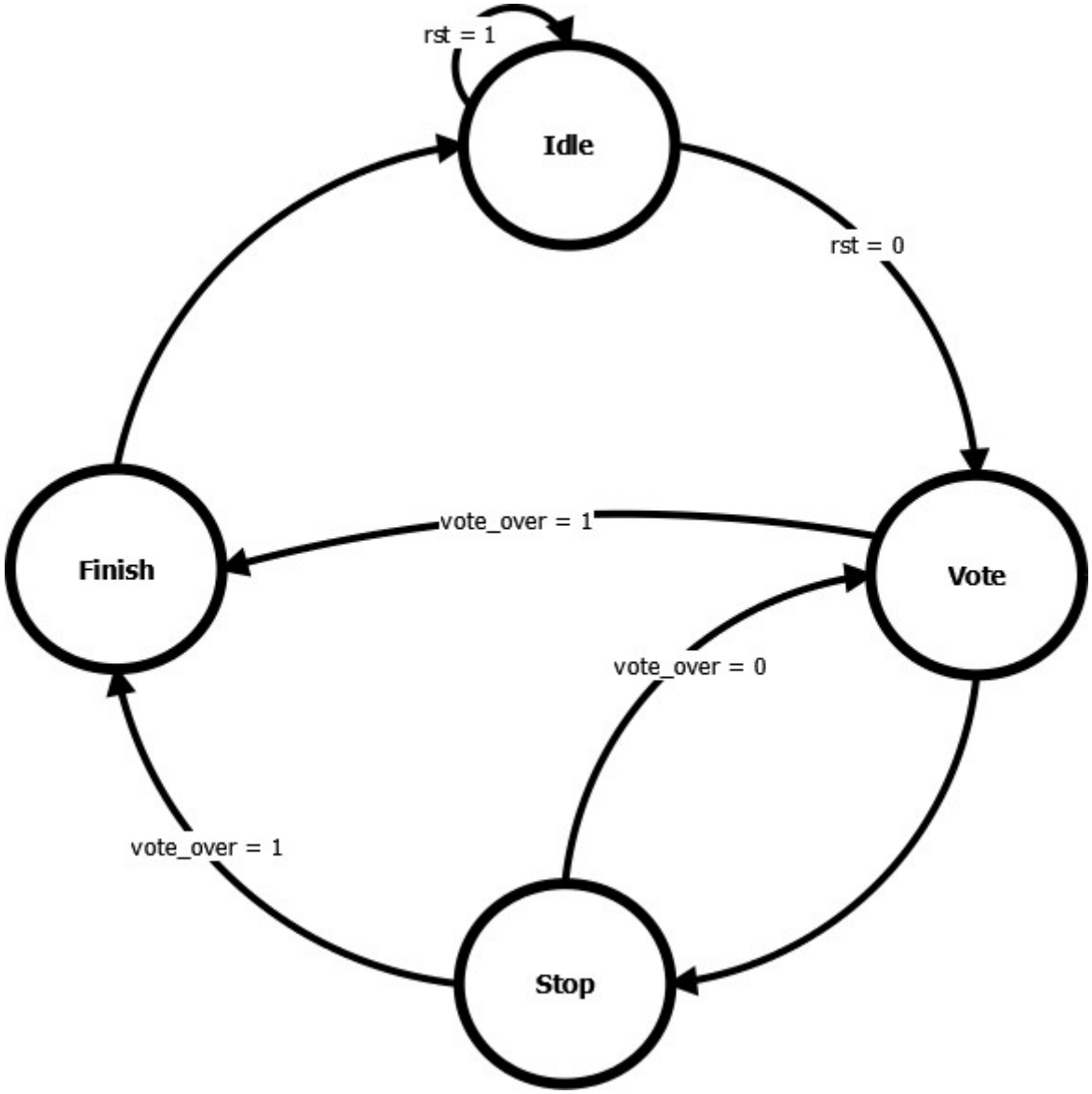
A voting machine using Verilog is an electronic device that aims to provide a reliable and efficient solution for counting votes in an election. It is designed using Verilog, a hardware description language that is commonly used for designing digital circuits and systems. The main objective of a voting machine using Verilog is to provide an accurate and secure vote counting mechanism that can prevent any possibility of vote tampering or manipulation.

The Verilog code is used to design the digital circuitry of the machine, including input and output modules, processing units, and memory. The machine can handle various voting options, such as buttons or touchscreens, and can tally results in real-time. It is designed to minimize the chances of errors in vote counting and ensure that the machine can operate efficiently for extended periods without any glitches or failures.

In addition to being reliable and secure, the voting machine using Verilog is also user-friendly, with clear instructions and an intuitive interface that is easy for voters to use. It is designed to ensure the confidentiality and anonymity of the voters' choices, protecting the integrity of the electoral process.

Overall, a voting machine using Verilog offers a modern and effective solution for elections, improving the speed and accuracy of the voting process while maintaining the highest level of security and confidentiality. It is a promising technology that has the potential to revolutionize the way we conduct elections and ensure fair and transparent outcomes.

STATE DIAGRAM:



FLOW CHART:

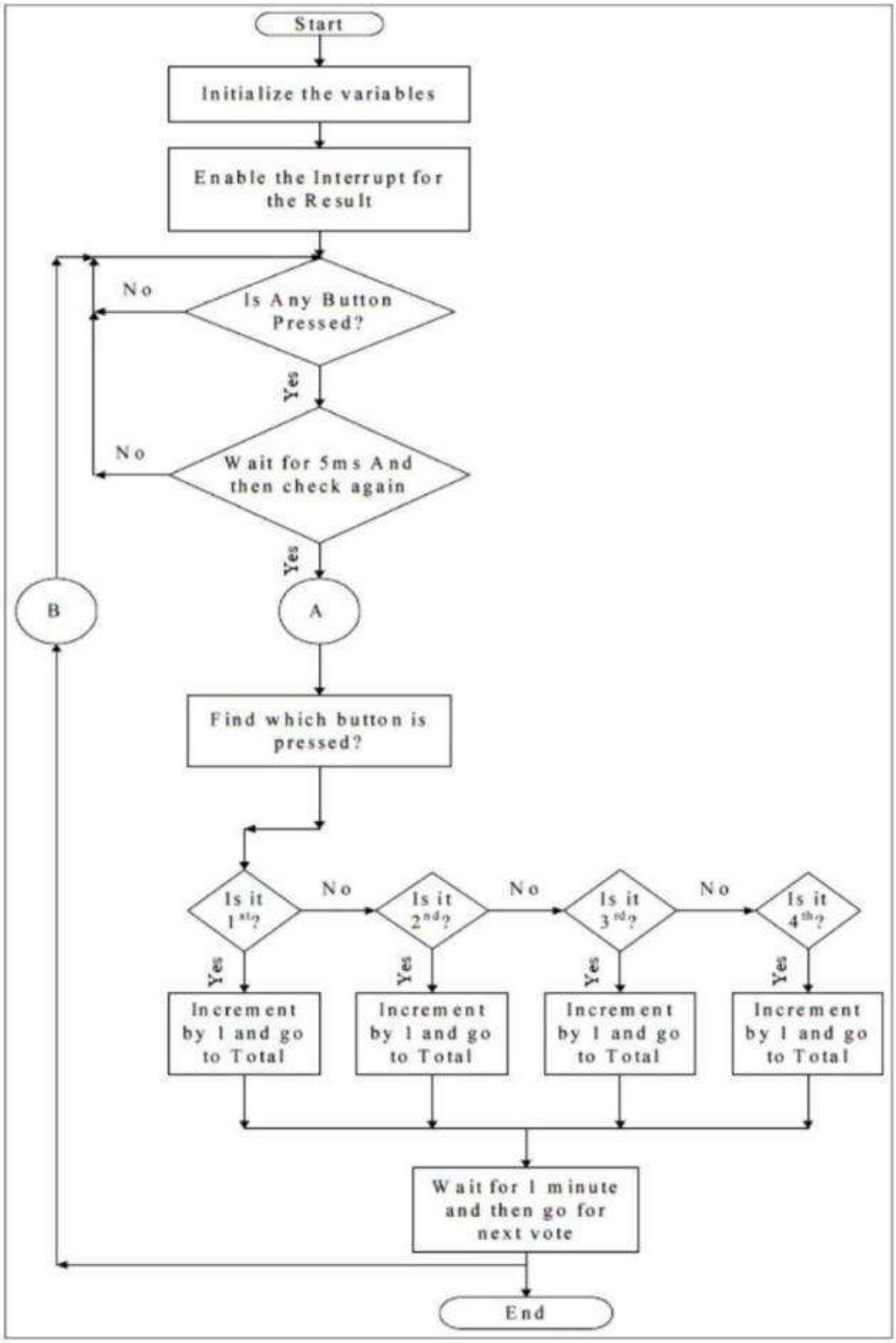


Figure 2.2: Flow Chart

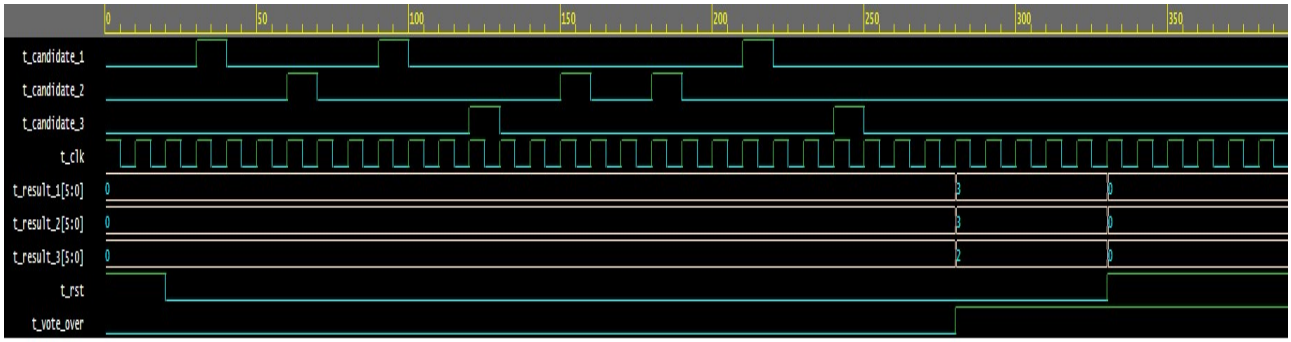
CODE:

```
module voting_machine #(
parameter idle =  2'b00,
parameter vote =  2'b01,
parameter hold =  2'b10,
parameter finish = 2'b11
)(
    input clk,
    input rst,
    input i_candidate_1,
    input i_candidate_2,
    input i_candidate_3,
    input i_voting_over,

    output reg [31:0] o_count1,
    output reg [31:0] o_count2,
    output reg [31:0] o_count3

);

reg [31:0] r_cand1_prev;
reg [31:0] r_cand2_prev;
reg [31:0] r_cand3_prev;                                        /

reg [31:0] r_counter_1;
reg [31:0] r_counter_2;
reg [31:0] r_counter_3;

reg [1:0] r_present_state, r_next_state;
reg [3:0] r_hold_count;

always @(posedge clk or negedge rst)
            begin
                    case (r_present_state)
                          idle: if (!rst)
                          begin
                                  r_next_state <= vote;
                    end
                    else
                           begin
                                  r_counter_1 <= 32'b0;
                                  r_counter_2 <= 32'b0;
                                  r_counter_3 <= 32'b0;
                                  r_hold_count <= 4'b0000;

                                  r_next_state <= idle;
                    end
```

```
vote: if (i_voting_over == 1'b1)
                begin
                r_next_state <= finish;
        end

        else if (i_candidate_1 == 1'b0 && r_cand1_prev == 1'b1)
                begin
                r_counter_1 <= r_counter_1 + 1'b1;
                r_next_state <= hold;
        end

        else if (i_candidate_2 == 1'b0 && r_cand2_prev == 1'b1)
                begin
                r_counter_2 <= r_counter_2 + 1'b1;
                r_next_state <= hold;
        end

         else if (i_candidate_3 == 1'b0 && r_cand3_prev == 1'b1)
                begin
                r_counter_3 <= r_counter_3 + 1'b1;
                r_next_state <= hold;
        end

        else
                 begin
                 r_counter_1 <= r_counter_1;
                 r_counter_2 <= r_counter_2;
                 r_counter_3 <= r_counter_3;
                 r_next_state <= vote;
        end

hold: if (i_voting_over == 1'b1)
                 begin
                 r_next_state <= finish;
        end

        else
                 begin
                 if (r_hold_count != 4'b1111) begin
                                r_hold_count = r_hold_count + 1'b1;
                 end

         else
                 begin
                 r_next_state <= vote;
                 end

         end

finish: if (i_voting_over == 1'b0)
```

```
                    begin
                    r_next_state <= idle;

        end

        else
                    begin
                    r_next_state <= finish;
        end

default:
                begin
                        r_counter_1 <= 32'b0;
                        r_counter_2 <= 32'b0;
                        r_counter_3 <= 32'b0;
                        r_hold_count <= 4'b0000;

                        r_next_state <= idle;

                end
endcase
end

always @(posedge clk or negedge rst)

begin
    if (rst == 1'b1)
        begin
                r_present_state <= idle;
                o_count1 <= 32'b0;
                o_count2 <= 32'b0;
                o_count3 <= 32'b0;
                r_hold_count <= 4'b0000;
    end

    else if (rst == 1'b0 && i_voting_over == 1'b1)
        begin
                o_count1 <= r_counter_1;
                o_count2 <= r_counter_2;
                o_count3 <= r_counter_3;
    end

    else
        begin
                r_present_state <= r_next_state;
                r_cand1_prev <= i_candidate_1;
                r_cand2_prev <= i_candidate_2;
                r_cand3_prev <= i_candidate_3;
     end

end
endmodule
```

WAVEFORM:



RESULT:

The provided Verilog code implements a simple voting machine that counts votes for three candidates. The machine has four states: idle, vote, hold, and finish. The input signals include clk, rst, i_candidate_1, i_candidate_2, i_candidate_3, and i_voting_over. The output signals are o_count1, o_count2, and o_count3, which represent the vote counts for each candidate. When the machine is in the idle state, it waits for a reset signal to transition to the vote state. In the vote state, it waits for a vote to be cast by checking the input signals for each candidate. If a vote is detected, the corresponding vote counter is incremented, and the machine transitions to the hold state.

In the hold state, the machine waits for a set amount of time before transitioning back to the vote state. The amount of time is controlled by a hold counter, which counts up to 15 before transitioning back to the vote state. In the finish state, the machine waits for the voting period to be over before transitioning back to the idle state. When the voting period is over, the final vote counts are output through the o_count1, o_count2, and o_count3 signals.

The Verilog code implements this behavior using two always blocks. The first always block is sensitive to the clk and rst signals and uses a case statement to control the state transitions and vote counting. The second always block is also sensitive to the clk and rst signals and is responsible for updating the present state and outputting the final vote counts.

Overall, the provided Verilog code implements a simple voting machine that counts votes for three candidates using state machines and counters.

CONCLUSION:

The implementation of the voting machine using Verilog code was successful. The voting machine is designed to count votes for three candidates and determine the winner.It uses registers to store the previous state of the candidates, the counter for each candidate, and the current state of the machine. The Verilog code includes an always block that updates the current state of the machine based on the input and previous state, and another always block that outputs the vote count for each candidate Overall, the Verilog code for the voting machine is a functional implementation that can be used for vote counting in various applications.