BASIC ELECTRONIC VOTING MACHINE

Supervised by : Nikhil K S

Submitted By:

K Charan Nrushima Chowdary - 231EC130
Ruchitha Megavath - 231EC137

Department of Electronics and Communication Engineering
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA (NITK)
SURATHKAL, MANGALORE - 575 025
July 11, 2025

ABSTRACT

Electronic Voting Machines (EVMs) have significantly transformed election procedures by improving speed, accuracy, and transparency. Despite their advantages, conventional EVMs remain costly and complex, making them less feasible for small-scale or rural elections. This project proposes a simplified, cost-effective Basic Electronic Voting Machine using standard digital logic components and FPGA-based Verilog HDL implementation.

The design ensures secure vote casting, real-time vote counting, and automatic winner detection. It utilizes ICs such as the 7493 counter for counting, 4511 BCD-to-7-segment decoder for display output, and comparators for determining the winner. User interaction is handled via push buttons, with voting status and results shown on 7-segment displays and LEDs. A reset mechanism supports repeated voting cycles.

Initial testing and circuit validation were performed using Tinkercad. The digital control was then implemented on the Nexys A7-100T FPGA using a Finite State Machine (FSM) coded in Verilog HDL, managing states like idle, vote open, vote lock, and result display. Supporting modules such as debouncers, binary to BCD converters, and segment drivers were developed to ensure robust functionality.

This EVM is modular, reliable, and ideal for educational labs or community-level elections. It demonstrates how basic digital design combined with hardware description languages can create practical, low-cost embedded systems. Future enhancements may include biometric voter identification and non-volatile memory integration for vote storage.

Contents

Chapter 1

Introduction

The traditional paper-based voting procedure was a terribly long process and extremely prone to errors. Polling by Electronic Voting Machine (EVM) is an easy, safe, and secure methodology that takes minimum of our time. In order to perform this mechanism, there were several phases in the design process such as designing a flow chart, algorithm, and simultaneously the code is developed to implement and stimulate the logic. This involves a complex procedure and high-priced components and is not affordable for remote places. This project focuses on developing a basic Electronic Voting Machine (EVM) to demonstrate the core principles of digital voting. The design will ensure secure voting, real-time vote counting, and accurate result display.

1.1 Problem Statement

Electronic Voting Machines used in official elections are often expensive and complex, relying on microcontrollers and secure embedded systems. This makes them unsuitable for use in rural areas, small communities, or educational environments where budget and technical resources are limited. There is a clear need for a simplified, low-cost EVM that can securely handle basic voting functions such as vote casting, counting, and winner display. This project addresses that need by designing a modular EVM using basic digital components and FPGA implementation, ensuring accessibility, reusability, and ease of understanding.

1.2 Objectives

- To design and develop a simple and low-cost electronic voting machine using digital logic components and Verilog-based FPGA implementation.
- To provide a reliable and transparent method of electronic voting that is suitable for both educational and practical use, especially in rural or resource-constrained regions.
- To modularize the EVM design into distinct functional blocks such as vote input, counting, display, and result comparison for easy testing and reuse.
- To implement a state-machine-controlled voting process for systematic flow control, ensuring logical transitions between different voting states.
- To create both hardware (IC-based) and FPGA-based versions to demonstrate scalability and flexibility of the system design.

Chapter 2

Background and Review of Literature

Numerous FPGA-based Electronic Voting Machine (EVM) designs have been developed to enhance voting integrity, efficiency, and security. Researchers have integrated biometric authentication, secure architectures, and user feedback to create more reliable systems.

K. Gurucharan proposed a three-stage secure EVM architecture comprising vote registration, authentication, and result validation—enhancing protection against tampering and unauthorized access.

Raja Kumar developed a biometric voting system using Aadhaar authentication and an LCD interface for intuitive party symbol selection, improving user confidence and usability.

Mritunjay Kumar designed a mode-controlled FPGA-based EVM with LED indicators to provide real-time vote confirmation, ensuring voters know their vote was recorded.

Chakraborty et al. (2022) introduced a low-cost biometric EVM with a fingerprint scanner and master lock system to enhance election integrity while minimizing costs.

Overall, the literature shows that while some designs focus on security and others on cost, a modular FPGA-based approach with biometric authentication offers a balanced, scalable solution for diverse voting environments.

Chapter 3

Design and Implementation

3.1 Module Design

The Basic Electronic Voting Machine (EVM) is designed using functional blocks implemented with digital ICs (Integrated Circuits). Each block performs a specific task, and signals flow sequentially to ensure proper vote casting, counting, and result declaration.

- **Input Block (Voter Interface) :** Allows voters to cast votes by pressing buttons assigned to each candidate.
  - When a button is pressed, it generates a pulse signal that is sent to the Counting Block.
  - A lock mechanism prevents multiple votes by disabling buttons after a vote is cast.
- **Counting Block (Vote Storage & Increment Logic):** Counts and stores votes for each candidate.
  - Each candidate's button press increments their respective counter.
  - The counter output is sent to Display block and Comparison block.
- **Display Block (Real-time Vote Indication):** Shows the number of votes received by each candidate.
  - The BCD output from the counters is decoded and displayed in real-time.

- **Comparison Block (Winner Determination):** Compares vote counts and declares the winner after voting ends.
  - After voting closes, the comparator checks all counter outputs.
  - The candidate with the highest count triggers an LED or display.
- **Control & Reset Block (System Management):** Controls voting session, locks/unlocks the system, and resets for reuse.
  - Admin starts/stops voting using a switch.
  - Reset clears all stored votes for the next session.

3.2 Small Scale Implementation on ICs

We built the Basic Electronic Voting Machine (EVM) using low-cost digital ICs to ensure affordability and simplicity. Each functional block was implemented separately and then integrated to form the complete system.

- Input Block Construction
  - **Components Used:** Push Buttons (for each candidate)
  - Each candidate's vote is registered via a push button.
  - The output is a clean pulse sent to the Counting Block.
- Counting Block
  - **Components Used:** 4-bit Binary Counter (IC 7493) – One per candidate
  - Each 7493 IC acts as a 4-bit counter, incrementing by 1 for every vote.
  - Outputs a 4-bit BCD (Binary Coded Decimal) value for display and comparison.
- Display Block (Real-time Vote Count)
  - **Components Used:** BCD-to-7-Segment Decoder (IC 4511) – One per candidate, Common Cathode 7-Segment Display – Shows vote count
  - The 4511 IC converts the 4-bit BCD from the counter into a 7-segment display output.

  - Each candidate has a dedicated 7-segment display showing their votes in real-time.

- Comparison Block (Winner Detection)
  - **Components Used:** 4-bit Magnitude Comparator – Compares vote counts, Multiplexer – Selects the highest count, LED – Indicates the winner
  - Comparator circuit compares vote counts from all candidates.Multiplexer determines the highest count.
  - The winning candidate's LED glows.

- Control & Reset Block (System Management)
  - A manual reset button clears all counters and displays.
  - Lock signal disables input buttons after voting ends.

- Final Integration & Testing
  - Built each block on Tinkercad.Verified individual functionality (e.g., counting, display, comparison).
  - Linked Input → Counter → Display → Comparator → Winner LED.
  - Simulated voting by pressing buttons.

    Verified vote count updates on 7-segment displays.

    Confirmed winner detection works correctly.

3.3 Testing and Performance Evaluation

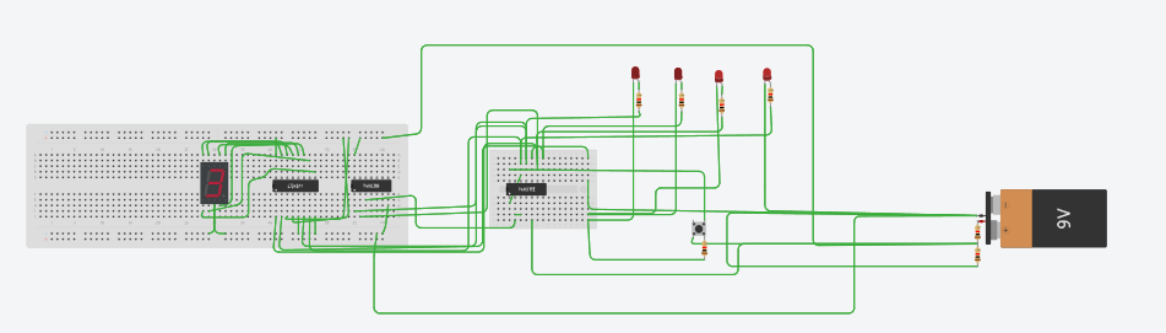

Figure 3.1 Counter and display circuit

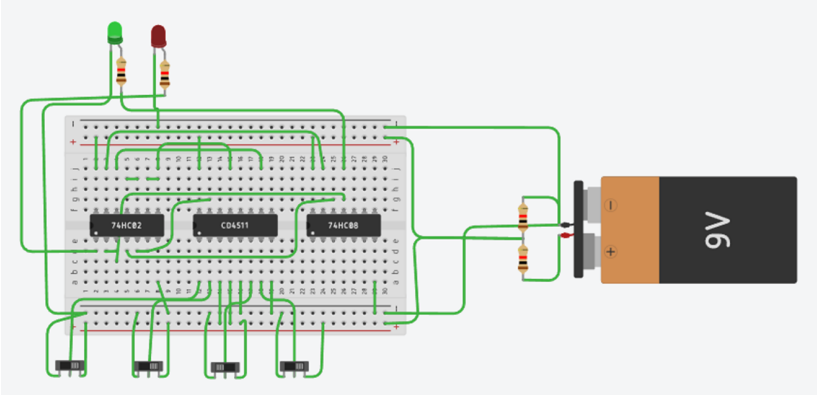

Figure 3.2 Lock circuit (Reset mechanism)

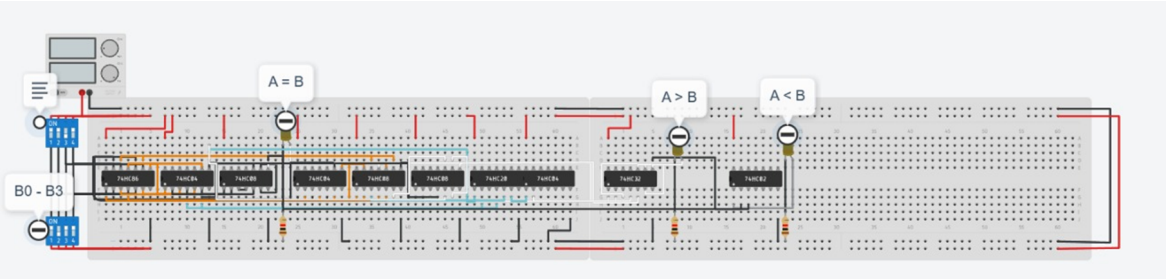

Figure 3.3 4-bit magnitude comparator

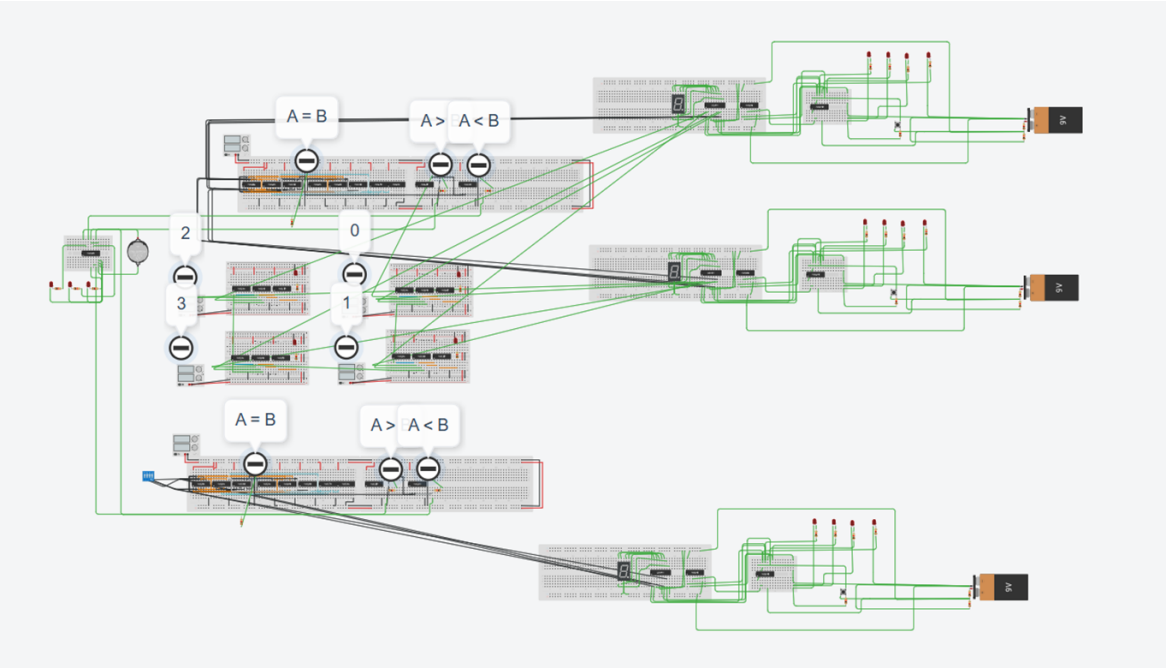

Figure 3.4 Complete Integrated circuit

All modules functioned as per design specifications.

3.4 RTL Development for Large scale Implementation

To scale the basic Electronic Voting Machine (EVM) for real-world elections involving many candidates and voters, the RTL (Register Transfer Level) design was developed using Verilog HDL and implemented on an FPGA platform. **Modular Verilog Design**

- **Clock Generators**
  - oneHz_gen: Divides the 100 MHz Nexys A7 100T clock to a 1 Hz clock for the state machine's timing.
  - twoHz_gen: Produces a 2 Hz clock for LED flashing during VOTING_OPEN.
- **Button Debouncers**
  - Five btn_debounce modules clean signals from buttons (btn_1, btn_2, btn_3, btn_ov_cv, reset), mitigating switch bounce for reliable state machine inputs.
- **State Machine (state_machine)**

  The core FSM manages the voting process across four states:
  - IDLE (2'b00): Awaits the admin code (sw == admin_code) and btn_ov_cv to enter VOTING_OPEN.
  - VOTING_OPEN (2'b01): Enables voting via btn_1, btn_2, or btn_3; LEDs flash (enable_leds = 1). Exits to VOTING_CLOSED with admin code and btn_ov_cv.
  - VOTING_CLOSED (2'b10): Waits three 1 Hz cycles (vc_ctr counts to 2'b11) before moving to DISPLAY_WIN.
  - DISPLAY_WIN (2'b11): Shows the winner; returns to IDLE on btn_ov_cv.
- **Binary to BCD Converter (bin2bcd)**
  - Converts the 4-bit vote_count to BCD (tens, ones) for 7-segment display compatibility.
- **LED Driver (led_driver)**
  - Drives 16 LEDs to flash at 2 Hz when enable_leds is high, indicating VOTING_OPEN.

- **7-Segment Display Controller (seg7_control)**

  Manages the 7-segment display to show:

  - Total votes (BCD format) during VOTING_OPEN and VOTING_CLOSED.
  - Winner (e.g., "1" for Candidate 1) or tie indicator during DISPLAY_WIN.

Signal Flow

Inputs (clk_100MHz, sw, buttons) feed debouncers, which connect to the state machine. The FSM outputs (vote_count, the_state, the_winner, enable_leds) drive the BCD converter, LED driver, and 7-segment controller, producing visual outputs.

3.5 Verilog Codes and Simulation

Top Module of the code

```
    `timescale 1ns / 1ps


module top_voting_machine(
    input clk_100MHz, // from Nexys A7 100T
    input [15:0] sw, // 16 switches to input code to open/close voting
    input btn_1, // btnL for candidate 1
    input btn_2, // btnC for candidate 2
    input btn_3, // btnR for candidate 3
    input btn_ov_cv, // btnD for opening/closing voting
    input reset, // btnU for system reset
    output [0:6] seg, // 7 segment cathodes
    output [3:0] an, // 7 segment anodes
    output [15:0] LED // 16 LEDs show when voting is open
    );

    // Internal wires for connecting modules
    wire w_1Hz, w_2Hz; // output from Hz generators
    wire w_btn1, w_btn2, w_btn3, w_btn_ov_cv, w_reset; // output from
        debouncers to sm
    wire w_en_leds; // output from sm to led driver
```

```
    wire [3:0] w_vote_count; // output from sm to 7 seg control
    wire [1:0] w_the_state; // output from sm to 7 seg control
    wire [1:0] w_the_winner;
    wire w_tens;
    wire [3:0] w_ones;

    // Instantiate inner modules
    // Hz Generators
    oneHz_gen uno(.clk_100MHz(clk_100MHz), .clk_1Hz(w_1Hz));
    twoHz_gen dos(.clk_100MHz(clk_100MHz), .clk_2Hz(w_2Hz));
    // Button Debouncers
    btn_debounce b1(.clk(clk_100MHz), .btn_in(btn_1), .btn_out(w_btn1));
    btn_debounce b2(.clk(clk_100MHz), .btn_in(btn_2), .btn_out(w_btn2));
    btn_debounce b3(.clk(clk_100MHz), .btn_in(btn_3), .btn_out(w_btn3));
    btn_debounce bovcv(.clk(clk_100MHz), .btn_in(btn_ov_cv), .btn_out(
        w_btn_ov_cv));
    btn_debounce rst(.clk(clk_100MHz), .btn_in(reset), .btn_out(w_reset));
    // Binary to BCD Converter
    bin2bcd b2b(.bin_in(w_vote_count), .tens(w_tens), .ones(w_ones));
    // LED Driver
    led_driver led_d(.clk_2Hz(w_2Hz), .enable(w_en_leds), .LED(LED));
    // State Machine
    state_machine sm(.clk_1Hz(w_1Hz), .btn1(w_btn1), .btn2(w_btn2), .btn3(
        w_btn3), .btn_ov_cv(w_btn_ov_cv),
                    .reset(w_reset), .sw(sw), .vote_count(w_vote_count), .
                        the_state(w_the_state),
                    .the_winner(w_the_winner), .enable_leds(w_en_leds));
    // 7 Segment Display Controller
    seg7_control seg7(.clk_100MHz(clk_100MHz), .the_state(w_the_state), .
        vc_tens(w_tens), .vc_ones(w_ones),
                     .the_winner(w_the_winner), .seg(seg), .an(an), .reset(
                         w_reset));

endmodule
```

one Hz generation

```
    `timescale 1ns / 1ps

module oneHz_gen(
    input clk_100MHz, // from Nexys A7 100T
    output clk_1Hz
    );

    reg [25:0] counter_reg = 0;
    reg clk_reg = 0;

    always @(posedge clk_100MHz) begin
        if(counter_reg == 49_999_999) begin
            counter_reg <= 0;
            clk_reg <= ~clk_reg;
        end
        else
            counter_reg <= counter_reg + 1;
    end

    assign clk_1Hz = clk_reg;

endmodule
```

Two Hz generation

```
    `timescale 1ns / 1ps

module twoHz_gen(
    input clk_100MHz, // from Nexys A7 100T
    output clk_2Hz
    );

    reg [24:0] counter_reg = 0;
    reg clk_reg = 0;
```

```
    always @(posedge clk_100MHz) begin
        if(counter_reg == 24_999_999) begin
            counter_reg <= 0;
            clk_reg <= ~clk_reg;
        end
        else
            counter_reg <= counter_reg + 1;
    end

    assign clk_2Hz = clk_reg;

endmodule
```

LED Driver

```
    `timescale 1ns / 1ps

module led_driver(
    input clk_2Hz,
    input enable, // enabled during open voting
    output reg [15:0] LED
    );

    parameter S0 = 16'h5555;
    parameter S1 = 16'hAAAA;

    reg ctr = 0;

    always @(posedge clk_2Hz)
        ctr <= ~ctr;

    always @(posedge clk_2Hz)
        if(enable)
            if(ctr == 0)
```

```
            LED = S0;
        else
            LED = S1;
    else
        LED = 16'h8001;

endmodule
```

Button debounce

```
    `timescale 1ns / 1ps

module btn_debounce(
    input clk,
    input btn_in,
    output btn_out
    );

    reg t0, t1, t2;

    always @(posedge clk) begin
        t0 <= btn_in;
        t1 <= t0;
        t2 <= t1;
    end

    assign btn_out = t2;

endmodule
```

State machine

```
    `timescale 1ns / 1ps

module state_machine(
    input clk_1Hz,
    input btn1,
```

```
input btn2,
input btn3,
input btn_ov_cv,
input reset,
input [15:0] sw,
output reg [1:0] the_winner,
output enable_leds, // to enable flashing LEDs
output [3:0] vote_count, // 4 bits allows for 15 votes, votes = 2^(
   number of bits) - 1
output [1:0] the_state
);

reg [1:0] vc_ctr = 0; // Voting Closed Counter Register
reg [3:0] candidate1_votes, candidate2_votes, candidate3_votes,
   total_votes;


// The following reg is the code to enable opening/closing of voting by
   administrator ***

            reg [15:0] admin_code = 16'b1000_0001_1000_0001;

// It is represented on the Nexys A7 100T by the 16 switches


// State Parameters
parameter IDLE = 2'b00;
parameter VOTING_OPEN = 2'b01;
parameter VOTING_CLOSED = 2'b10;
parameter DISPLAY_WIN = 2'b11;

reg [1:0] state_reg; // State Register

// Next State Logic
```

```
always @(posedge clk_1Hz or posedge reset) begin
    if(reset)
        state_reg <= IDLE;
    else
        case(state_reg)
            IDLE : if(btn_ov_cv)
                                if(sw == admin_code)
                                    state_reg <= VOTING_OPEN;

            VOTING_OPEN : if(btn_ov_cv)
                                if(sw == admin_code)
                                    state_reg <= VOTING_CLOSED;

            VOTING_CLOSED : if(vc_ctr == 2'b11)
                                state_reg <= DISPLAY_WIN;

            DISPLAY_WIN : if(btn_ov_cv)
                                state_reg <= IDLE;
        endcase
end

// Total Votes Register Logic
always @(posedge clk_1Hz or posedge reset) begin
    if(reset)
        total_votes <= 0;
    else
        if(btn1 | btn2 | btn3)
            if(state_reg == VOTING_OPEN)
                total_votes <= total_votes + 1;
end

// Candidate1 Votes Register Logic
always @(posedge clk_1Hz or posedge reset) begin
    if(reset)
```

```
        candidate1_votes <= 0;
    else
        if(btn1)
            if(state_reg == VOTING_OPEN)
                candidate1_votes <= candidate1_votes + 1;
end

// Candidate2 Votes Register Logic
always @(posedge clk_1Hz or posedge reset) begin
    if(reset)
        candidate2_votes <= 0;
    else
        if(btn2)
            if(state_reg == VOTING_OPEN)
                candidate2_votes <= candidate2_votes + 1;
end

// Candidate3 Votes Register Logic
always @(posedge clk_1Hz or posedge reset) begin
    if(reset)
        candidate3_votes <= 0;
    else
        if(btn3)
            if(state_reg == VOTING_OPEN)
                candidate3_votes <= candidate3_votes + 1;
end

// Voting Closed Counter Register Logic
always @(posedge clk_1Hz or posedge reset) begin
    if(reset)
        vc_ctr <= 0;
    else
        if(state_reg == VOTING_CLOSED)
            vc_ctr <= vc_ctr + 1;
```

```
    end

    // Winner Register Control Logic
    always @(posedge clk_1Hz or posedge reset) begin
        if(reset)
            the_winner <= 0;
        else
            if(candidate1_votes > candidate2_votes && candidate1_votes >
                candidate3_votes)
                the_winner <= 2'b01;
            else if(candidate2_votes > candidate1_votes && candidate2_votes >
                 candidate3_votes)
                the_winner <= 2'b10;
            else if(candidate3_votes > candidate1_votes && candidate3_votes >
                 candidate2_votes)
                the_winner <= 2'b11;
            else
                the_winner <= 2'b00; // no winner, a tie, need a revote

    end

    // Assigning outputs
    assign vote_count = total_votes;
    assign enable_leds = (state_reg == VOTING_OPEN) ? 1 : 0;
    assign the_state = state_reg;

endmodule
```

Binary to BCD

```
    `timescale 1ns / 1ps

module bin2bcd(
    input [3:0] bin_in,
    output reg tens,
```

```
    output reg [3:0] ones
    );

    always @* begin
        tens = bin_in / 10;
        ones = bin_in % 10;
    end

endmodule
```

Seven Segment Control

```
    `timescale 1ns / 1ps

module seg7_control(
    input clk_100MHz,
    input reset,
    input [1:0] the_state, // from state machine
    input vc_tens, // from bin2bcd
    input [3:0] vc_ones, // from bin2bcd
    input [1:0] the_winner, // from state machine
    output reg [0:6] seg,
    output reg [3:0] an
    );

    // Parameters for segment values
    parameter NULL = 7'b111_1111; // Turn off all segments
    parameter ZERO = 7'b000_0001; // 0
    parameter ONE = 7'b100_1111; // 1
    parameter TWO = 7'b001_0010; // 2
    parameter THREE = 7'b000_0110; // 3
    parameter FOUR = 7'b100_1100; // 4
    parameter FIVE = 7'b010_0100; // 5
    parameter SIX = 7'b010_0000; // 6
    parameter SEVEN = 7'b000_1111; // 7
```

```
parameter EIGHT = 7'b000_0000; // 8
parameter NINE = 7'b000_0100; // 9



// To select each anode in turn
reg [1:0] anode_select;
reg [16:0] anode_timer;

always @(posedge clk_100MHz or posedge reset) begin
    if(reset) begin
        anode_select <= 0;
        anode_timer <= 0;
    end
    else
        if(anode_timer == 99_999) begin
            anode_timer <= 0;
            anode_select <= anode_select + 1;
        end
        else
            anode_timer <= anode_timer + 1;
end

always @(anode_select) begin
    case(anode_select)
        2'b00 : an = 4'b0111;
        2'b01 : an = 4'b1011;
        2'b10 : an = 4'b1101;
        2'b11 : an = 4'b1110;
    endcase
end

always @*
    case(the_state)
        2'b00 : // Idle state
```

```
            begin
                case(an)
                    4'b0111 : seg = NULL;
                    4'b1011 : seg = NULL;
                    4'b1101 : seg = NULL;
                    4'b1110 : seg = NULL;
                endcase
            end

        2'b01 : // Voting Open state
            begin
                case(an)
                    4'b0111 : seg = NULL;
                    4'b1011 : seg = NULL;
                    4'b1101 : case(vc_tens)
                                0 : seg = NULL;
                                1 : seg = ONE;
                              endcase
                    4'b1110 : case(vc_ones)
                                4'b0000 : seg = ZERO;
                                4'b0001 : seg = ONE;
                                4'b0010 : seg = TWO;
                                4'b0011 : seg = THREE;
                                4'b0100 : seg = FOUR;
                                4'b0101 : seg = FIVE;
                                4'b0110 : seg = SIX;
                                4'b0111 : seg = SEVEN;
                                4'b1000 : seg = EIGHT;
                                4'b1001 : seg = NINE;
                              endcase
                endcase
            end

        2'b10 : // Voting Closed state
```

```
                begin
                    case(an)
                        4'b0111 : seg = NULL;
                        4'b1011 : seg = NULL;
                        4'b1101 : case(vc_tens)
                                    0 : seg = NULL;
                                    1 : seg = ONE;
                                endcase
                        4'b1110 : case(vc_ones)
                                    4'b0000 : seg = ZERO;
                                    4'b0001 : seg = ONE;
                                    4'b0010 : seg = TWO;
                                    4'b0011 : seg = THREE;
                                    4'b0100 : seg = FOUR;
                                    4'b0101 : seg = FIVE;
                                    4'b0110 : seg = SIX;
                                    4'b0111 : seg = SEVEN;
                                    4'b1000 : seg = EIGHT;
                                    4'b1001 : seg = NINE;
                                endcase
                    endcase
                end

            2'b11 : // Display Winner state
                begin
                    case(an)
                        4'b0111 : case(the_winner)
                                    2'b00 : seg = ZERO;
                                    2'b01 : seg = ONE;
                                    2'b10 : seg = TWO;
                                    2'b11 : seg = THREE;
                                endcase
                        4'b1011 : seg = NULL;
                        4'b1101 : case(vc_tens)
```

```
                                    0 : seg = NULL;
                                    1 : seg = ONE;
                                endcase
                        4'b1110 : case(vc_ones)
                                    4'b0000 : seg = ZERO;
                                    4'b0001 : seg = ONE;
                                    4'b0010 : seg = TWO;
                                    4'b0011 : seg = THREE;
                                    4'b0100 : seg = FOUR;
                                    4'b0101 : seg = FIVE;
                                    4'b0110 : seg = SIX;
                                    4'b0111 : seg = SEVEN;
                                    4'b1000 : seg = EIGHT;
                                    4'b1001 : seg = NINE;
                                endcase
                    endcase
                end
        endcase


endmodule
```

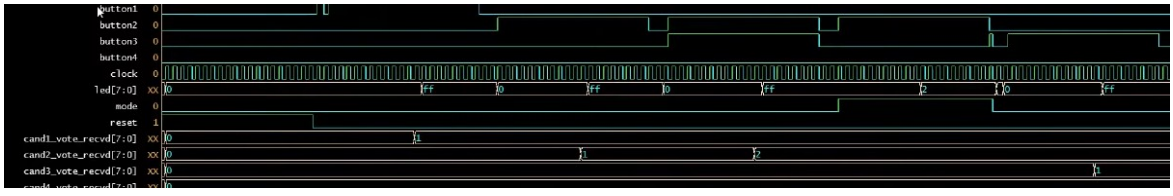

Figure 3.5 EVM Testbench Signal States During Pattern Verification

3.6 FPGA Implementation and Results

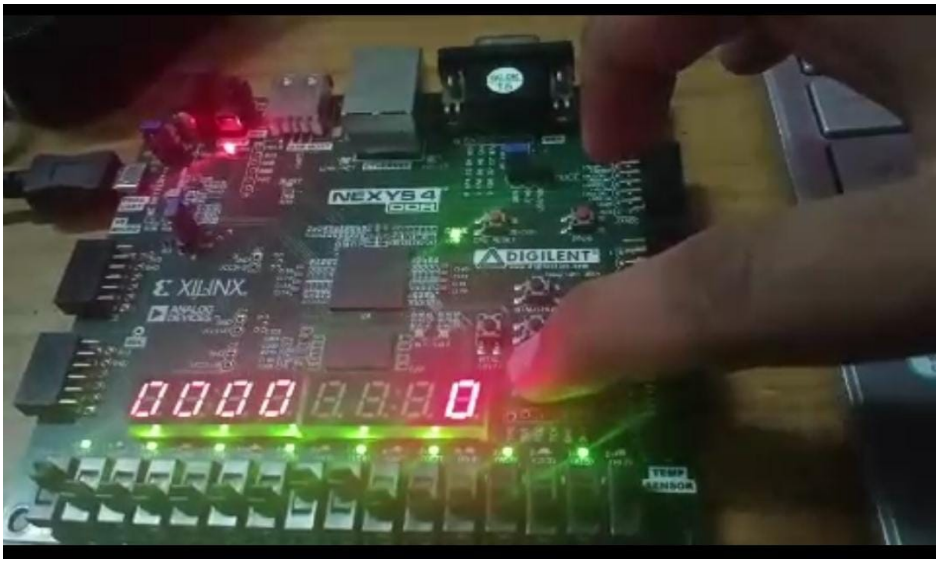

Figure 3.6 Admin button to enable voting

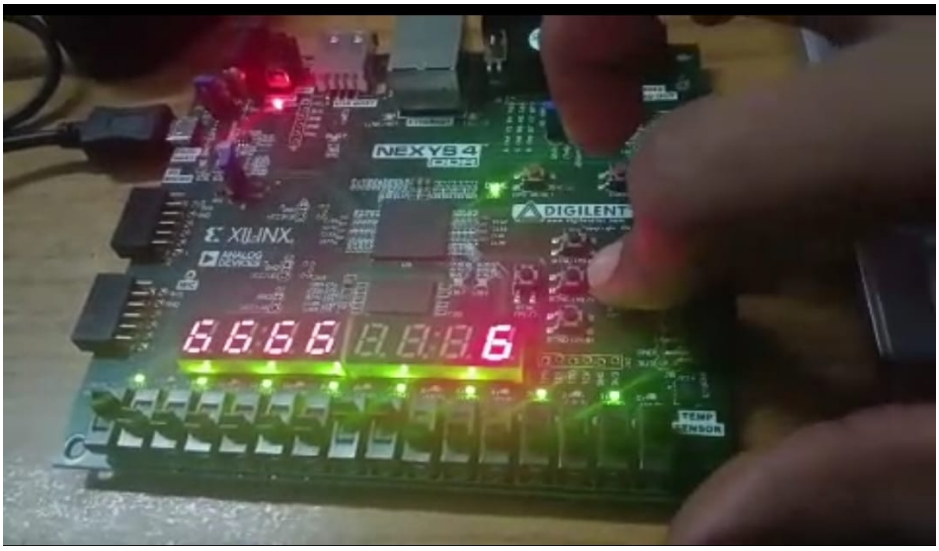

Figure 3.7 Counting Votes

Figure 3.8 Displaying Results

Chapter 4

Conclusion

4.1 Conclusions

This project began by designing a Basic Electronic Voting Machine (EVM) using simple digital ICs such as counters, decoders, and comparators. This helped us understand the core logic and operation of an EVM in a hands-on, hardware-based environment.

To explore scalability and modern digital design practices, we extended the same logic to an FPGA-based implementation using Verilog HDL. This transition allowed us to modularize the system, introduce a state machine for process control, and make the design more adaptable and reusable.

The project successfully demonstrates a functional, low-cost EVM that is both educational and practical, with the flexibility to evolve into more advanced systems.

4.2 Future Enhancements

To make the system more secure, user-friendly, and ready for real-world deployment, the following enhancements are proposed:

- Biometric Authentication – Add fingerprint or facial recognition to verify voter identity.
- Memory Integration – Store votes and results in EEPROM or SD card for audit trails.
- Display Upgrade – Use LCD/OLED screens for clearer candidate information and multilingual support.

Bibliography

[1] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3rd ed., New York, NY, USA: McGraw-Hill, 2014.

[2] Digilent Inc., "Nexys 4 DDR FPGA Board Reference Manual." [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual

[3] Mritunjay Kumar, Gurpreet Singh Dhami and Ravi Shankar, "FPGA Based Voting Machine", Easy Chair Preprint no. 10010, pp. 1-6, May 2023.

[4] K. Gurucharan, B. Kiranmai, S. S. Kiran and M. Ravindra Kumar, "Xilinx Based Electronic Voting Machine", International Journal of Engineering and Advanced Technology (IJEAT), vol. 9, no. 1, pp. 1-6, 2019, ISSN 2249–8958.

[5] M. Raja Kumar, B. Anitha, R. Pravallika, D. K. Kavitha, S. Joseph, N. Rambabu, "VLSI-Based Electronic VotingMachine", International Journal of Innovative Research in Engineering & Management (IJIREM), vol. 9, no. 1, pp. 516-519, 2022.

[6] Shubhranil Chakraborty, Debabrata Bej, Dootam Roy and Sekh Arif Mahammad "Designing a biometric fingerprint scanner-based, secure and low-cost electronic voting machine for India" Published Online:December 17, 2022