ELL201 Project: Electronic Voting Machine (EVM)

Dhruv Joshi 2022EE32079
Athipatla Rahul 2022MT11277
Nakshat Pandey 2022EE11436
Vibhor Agarwal 2022EE11426

Introduction

This project aims to implement an electronic voting machine suitable for handling up to 4 candidates and 31 votes for each candidate on a CPLD board. It utilizes three 7-segment displays to showcase the results. The system is designed with two primary states: a Voting state and a Display state.

Implementation Methodology

0.1 Inputs:

• **Toggle Switch for Input:** Increments the number of votes cast for a specific candidate using adders within the system.

• **Vote Counters:** Four adders store the respective vote counters for each candidate.

• **LED Indicators:** Four LEDs on the CPLD board light up respectively when the corresponding vote has been cast.

• **Enable Switch:** Acts as a button switch. The first press triggers the voting state, and the second press triggers the display state.

• **Next Result Switch:** Upon triggering the enable for the second time, this switch triggers a single counter that controls which candidate's vote count should be displayed on the three 7-segment displays.
  – The first 7-segment display shows the candidate number (indexed from 0).
  – The second and third 7-segment displays show the vote count in decimal.

Voting State:

• Initiated when the enable switch is set to high (switch 7 on the CPLD, line 42 of the code).

• Voting takes place entirely on the CPLD, where designated switches (1 to 4) correspond to each of the 4 candidates (variable "v" assigned on line 41).

• To cast a vote for candidate n, the voter flips the nth switch, making the nth bit of v equal to "1".

• The voter can confirm their vote by observing the corresponding LED on the CPLD (LEDs assigned to the variable "to_display"), indicating their vote has been cast for the nth candidate.

• The register cn stores the total number of votes cast for the nth candidate and is updated every time the respective switch is flipped.

• An if-else block handles overflow conditions.

Display State:

- Activated by setting the enable switch to low.
- The results are displayed on the three 7-segment LED displays.
- Inputs to the LED displays: ldsc1 (10's place of vote count), ldsc2 (unit place of vote count), and ldsv (candidate number).
- The variable next_result_ff (controlled by switch 6 on the CPLD) toggles the results from one candidate to another.
- Flipping the switch cycles through the candidates, and after the 4th candidate, it circles back to the 1st candidate.
- The vote count is initially stored in the variable ldsc, which is then resolved into ldsc1 and ldsc2 by an if-else block.

Truth Table

Reset	Clock	Enable	V	C0	C1	C2	C3	Overflow	Next_result	ldsv	ldsc1	ldsc2
0	1	0	0001	00000	00000	00000	00000	0	0	00	00	0000
0	2	0	0010	00000	00000	00000	00000	0	0	00	00	0000
0	3	1	0000	00000	00000	00000	00000	0	0	00	00	0000
0	4	0	0001	00001	00000	00000	00000	0	0	00	00	0000
0	5	0	0010	00001	00001	00000	00000	0	0	00	00	0000
1	6	x	0001	00000	00000	00000	00000	0	0	00	00	0000
0	7	0	0001	00001	00000	00000	00000	0	0	00	00	0000
0	8	0	0010	00001	00001	00000	00000	0	0	00	00	0000
0	9	0	0100	00010	00001	00001	00000	0	0	00	00	0000
0	10	0	1000	00010	00001	00001	00001	0	0	00	00	0000
0	11	0	0010	00010	00010	00001	00001	0	0	00	00	0000
0	12	0	0100	00010	00010	00010	00001	0	0	00	00	0000
0	13	0	0100	00010	00010	00011	00001	0	0	00	00	0000
0	14	0	0010	00010	00011	00011	00001	0	0	00	00	0000
0	15	0	0010	00010	00101	00011	00001	0	0	00	00	0000
0	16	0	0001	00011	00101	00011	00001	0	0	00	00	0000
0	17	0	0001	00101	00101	00011	00001	0	0	00	00	0000
0	18	0	0001	00110	00101	00011	00001	0	0	00	00	0000
0	19	0	0001	00111	00101	00011	00001	0	0	00	00	0000
0	20	0	0001	01000	00101	00011	00001	0	0	00	00	0000
0	21	0	0001	01001	00101	00011	00001	0	0	00	00	0000
0	22	0	0001	01010	00101	00011	00001	0	0	00	00	0000
0	23	0	0001	01011	00101	00011	00001	0	0	00	00	0000
0	24	0	0001	01100	00101	00011	00001	0	0	00	00	0000
0	25	0	0001	01101	00101	00011	00001	0	0	00	00	0000
0	26	0	0001	01110	00101	00011	00001	0	0	00	00	0000
0	27	0	0001	01111	00101	00011	00001	0	0	00	00	0000
0	28	0	0001	10000	00101	00011	00001	0	0	00	00	0000
0	29	0	0001	10001	00101	00011	00001	0	0	00	00	0000
0	30	0	0001	10010	00101	00011	00001	0	0	00	00	0000
0	31	0	0001	10011	00101	00011	00001	0	0	00	00	0000
0	32	0	0001	10100	00101	00011	00001	0	0	00	00	0000
0	33	0	0001	10101	00101	00011	00001	0	0	00	00	0000
0	34	0	0001	10110	00101	00011	00001	0	0	00	00	0000

0	35	0	0001	10111	00101	00011	00001	0	0	00	00	0000
0	36	0	0001	11000	00101	00011	00001	0	0	00	00	0000
0	37	0	0001	11001	00101	00011	00001	0	0	00	00	0000
0	38	0	0001	11010	00101	00011	00001	0	0	00	00	0000
0	39	0	0001	11011	00101	00011	00001	0	0	00	00	0000
0	40	0	0001	11100	00101	00011	00001	0	0	00	00	0000
0	41	0	0001	11101	00101	00011	00001	0	0	00	00	0000
0	42	0	0001	11110	00101	00011	00001	0	0	00	00	0000
0	43	0	0001	11111	00101	00011	00001	0	0	00	00	0000
0	44	0	0001	11111	00101	00011	00001	1	0	00	00	0000
0	45	1	xxxx	11111	00101	00011	00001	1	0	00	11	0001
0	46	0	xxxx	11111	00101	00011	00001	1	1	01	00	0101
0	47	0	xxxx	11111	00101	00011	00001	1	1	10	00	0011
0	48	0	xxxx	11111	00101	00011	00001	1	1	11	00	0001
0	49	1	xxxx	11111	00101	00011	00001	1	1	00	11	0001

Table 2: Legend

Color	**Meaning**
🟨	Voting Starts
🟪	Reset Of Votes
🟥	Overflow condition
🟦	End of Voting and Display of Results
🟩	Cycles Back to First Candidate

Circuit Diagrams

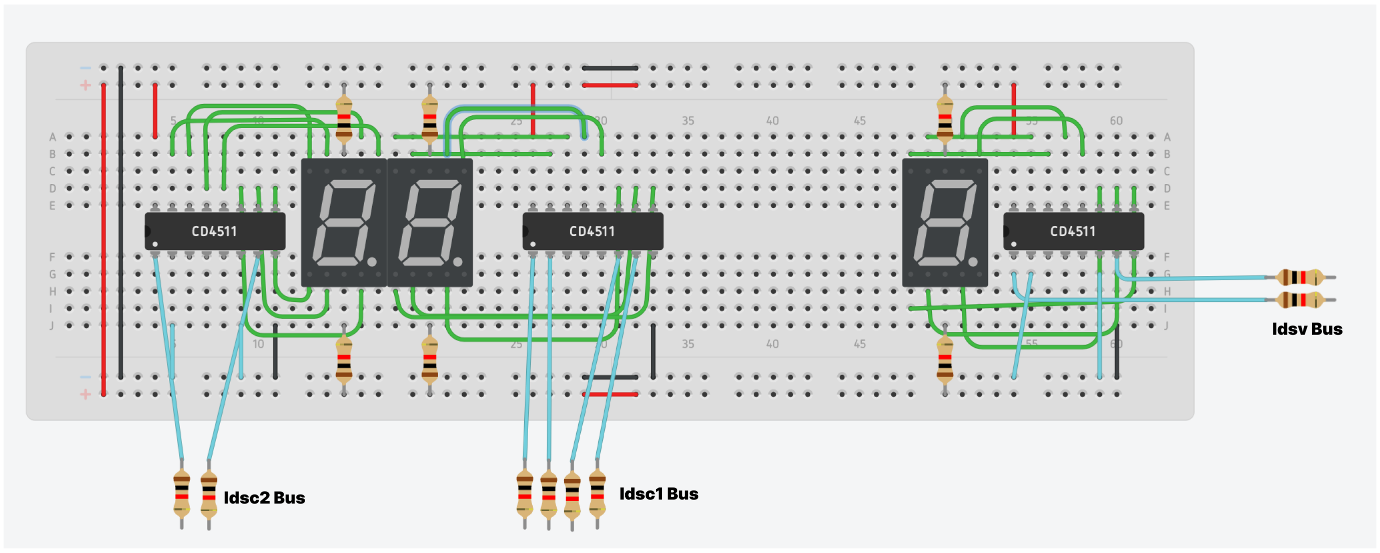

Figure 1: Circuit overview for 7-segment driving breadboard with I/O mapping to CPLD (source: Tinkercad) [due to unavailability of CD4511 ICs, 7447 ICs were used in the final circuit instead; I/O endpoints are represented by floating resistors]

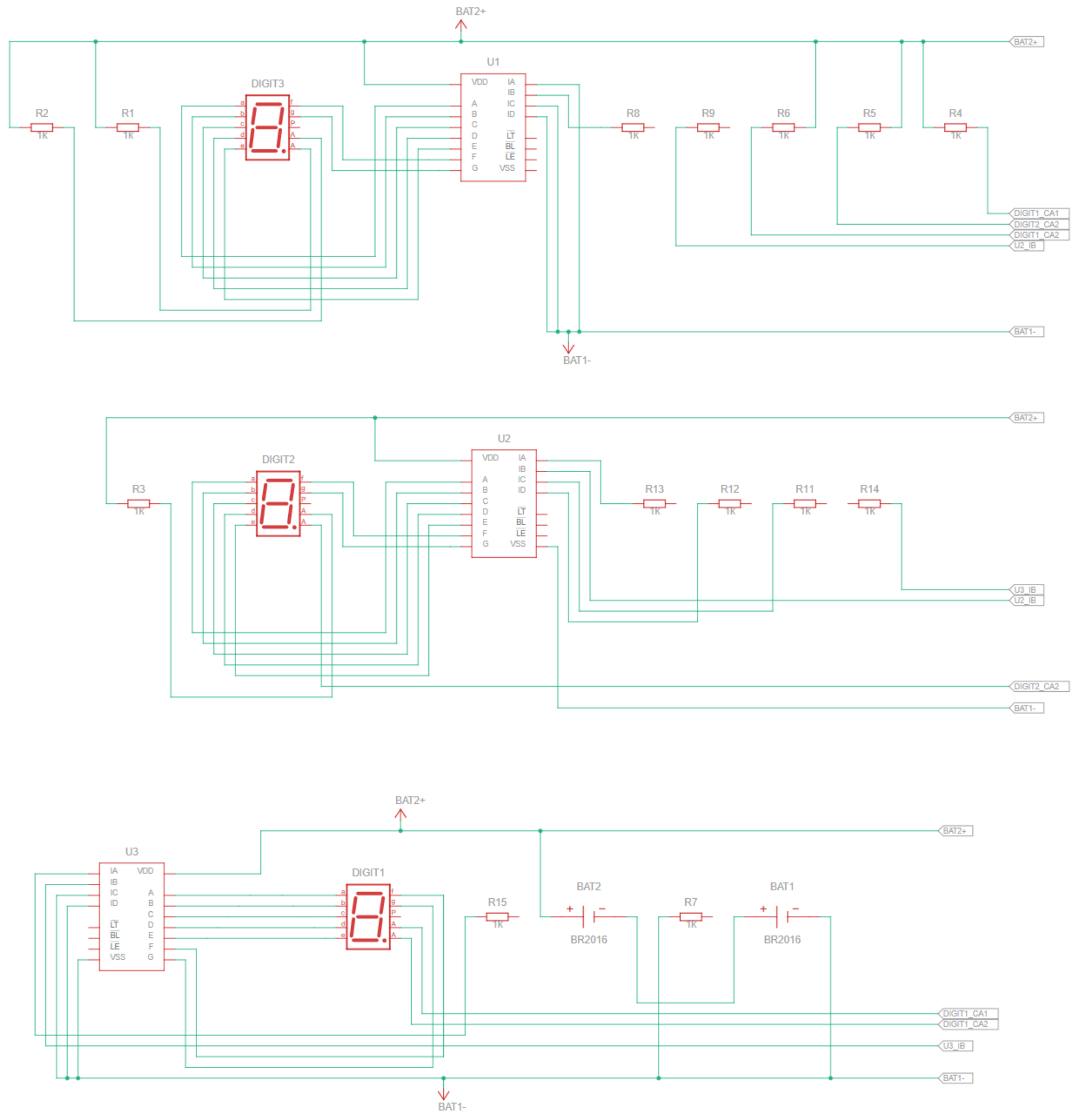

Figure 2: Circuit schematic for 7-segment driving breadboard (source: Tinkercad) [here a 5V power supply is modeled approximately using a 6V rail formed between two 3V cells; I/O endpoints are represented by floating resistors]

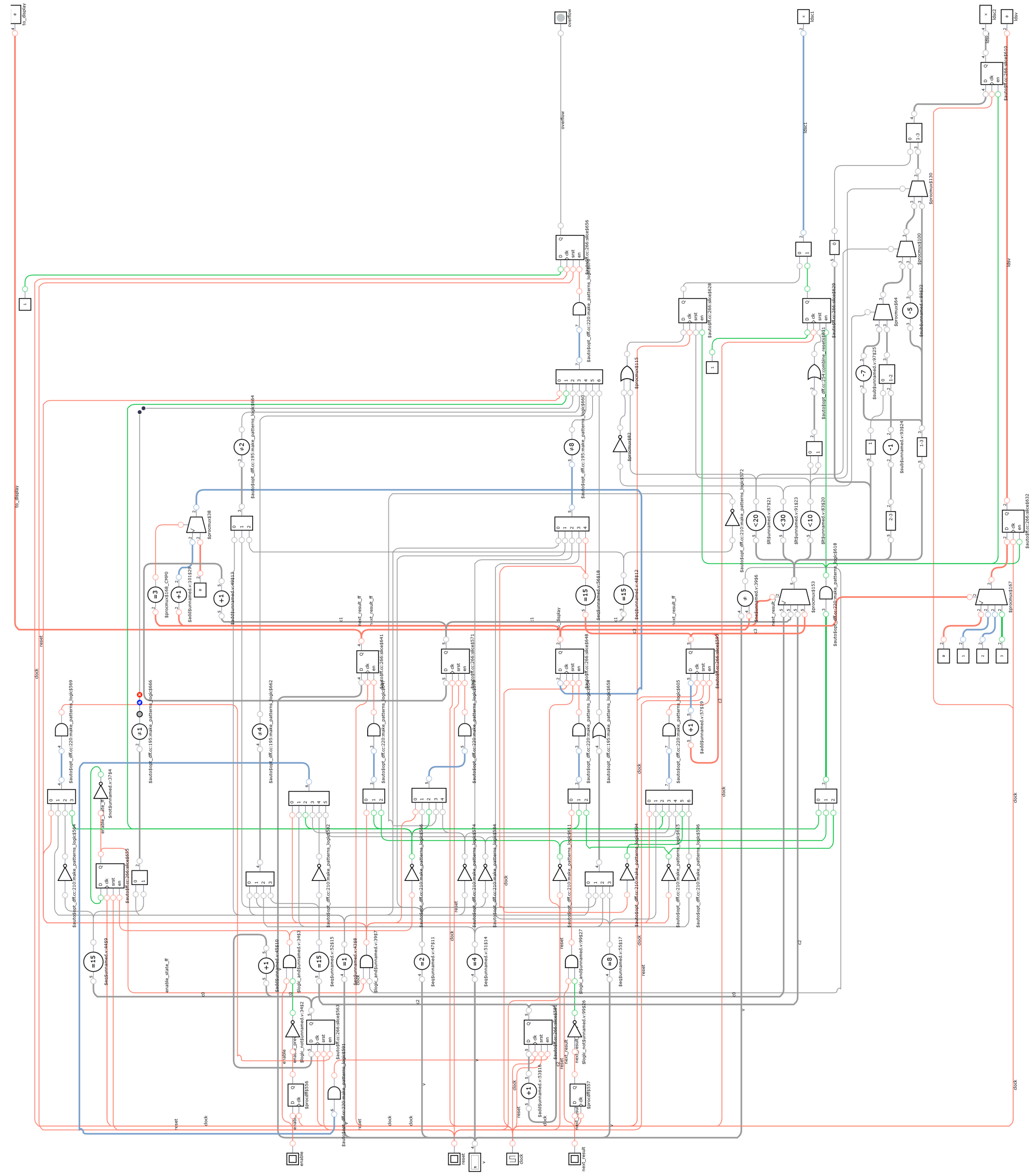

Figure 3: Synthesis of Verilog code; this is the simplest possible FPGA/CPLD circuit (source: Yosys open-source synthesizer)

1 Verilog Code

```verilog
/* I need to the following logic in my system:
One section that takes in the four inputs and stores the vote count in binary
The main module handles this as long the enable is turned on.
Once the enable is turned off, any inputs will be disregarded.
My next section is triggered once the enable is low.
This block takes the input from the button switch.
At posedge, it will display first candidate's number (01) and the vote count
When pressed again, it will go to the next candidate's vote count (10) and their vote
    count
This will continue as long as 1. the enable is low and 2. it will loop around back to
    first candidate.
*/

`timescale 1ns / 1ps

module EVM(v, enable, reset, clock, next_result, to_display, ldsv, ldsc1, ldsc2, overflow)
    ;
    input [3:0] v; // takes in a 4 bit vote from switches
    input enable; // triggers whether we are in voting state or result state
    input reset; // resets the voting all the way to zero
    input clock;
    input next_result;
    reg [4:0] c0, c1, c2, c3 = 5'b00000; // stores vote count for each candidate
    output reg [3:0] to_display = 4'b0000;
    output reg [1:0] ldsv; // to show the candidate number
    reg [4:0] ldsc; // to show the candidate vote count
    output reg [1:0] ldsc1;
    output reg [3:0] ldsc2;
    output reg overflow;
    reg enable_state_ff = 1'b0; // made this temporary addition as an output to show the
    reg [1:0] next_result_ff = 2'b00;
    reg enable_prev = 1'b0; // Additional signal to detect the positive edge of enable
    reg next_result_prev = 1'b0;
    // NOTE: I am yet to handle the case where we reset our EVM
    // NOTE: NOTE: Ignore above note
    // Voting logic
    always @(posedge clock or posedge reset or posedge enable or posedge next_result)
        begin
        if (reset) begin
            c0 = 5'b00000;
            c1 = 5'b00000;
            c2 = 5'b00000;
            c3 = 5'b00000;
            enable_state_ff = 1'b0;
            enable_prev = 1'b0;
            next_result_ff = 2'b00;
            overflow = 1'b0;
            next_result_prev = 2'b00;
        end
        else if (enable && !enable_prev) begin
            // Detect positive edge of enable
            // Toggle enable_state_ff on the positive edge of enable
            enable_state_ff = ~enable_state_ff;
        end
        else if (enable_state_ff == 1'b1 && v != to_display) begin // if we are in voting
            mode
            // in voting mode I also want to take out an output that goes to the
               breadboard
            // this goes out to a seven segment display:
            to_display = v;
            if (v == 4'b0001) begin
              if (c0 == 5'b11111) overflow = 1'b1;
                else c0 <= c0 + 5'b00001;
            end
            else if (v == 4'b0010) begin
              if (c1 == 5'b11111) overflow = 1'b1;
                else c1 <= c1 + 5'b00001;
            end
            else if (v == 4'b0100) begin
              if (c2 == 5'b11111) overflow = 1'b1;
                else c2 <= c2 + 5'b00001;
            end
            else if (v == 4'b1000) begin
              if (c3 == 5'b11111) overflow = 1'b1;
                else c3 <= c3 + 5'b00001;
            end
        end
        else begin
            case (next_result_ff)
                2'b00: begin //doing a bit of hard coding here
                    ldsv = 2'b00;
                    ldsc = c0;
                end
                2'b01: begin
                    ldsv = 2'b01;
                    ldsc = c1;
                end
                2'b10: begin
                    ldsv = 2'b10;
                    ldsc = c2;
                end
                2'b11: begin
                    ldsv = 2'b11;
                    ldsc = c3;
                end
            endcase;

            // Convert ldsc into 2 digits
            if (ldsc < 5'b01010) begin
                ldsc1 = 2'b00;
                ldsc2 = ldsc[3:0];
            end
            else if (ldsc < 5'b10100) begin
                ldsc1 = 2'b01;
                ldsc2 = ldsc - 5'b01010;
            end
            else if (ldsc < 5'b11110) begin
                ldsc1 = 2'b10;
                ldsc2 = ldsc - 5'b10100;
            end
            else begin
                ldsc1 = 2'b11;
                ldsc2 = ldsc - 5'b11110;
            end
            if (next_result && !next_result_prev) begin
                if (next_result_ff == 2'b11) next_result_ff = 2'b00; // if we are at 4th
                    press, we go back to candidate 1
                else next_result_ff = next_result_ff + 2'b01;
            end
        end
        // Update enable_prev for the next cycle
        enable_prev = enable;
        next_result_prev = next_result;
    end

    // Enable state toggle
endmodule
```

Test Bench

Test Bench Code

```
    `timescale 1ns / 1ps

module EVM_tb;
    // Define the inputs and outputs
    reg [3:0] v;
    reg enable;
    reg reset;
    reg clock;
    reg next_result;
    wire [3:0] to_display;
    wire [1:0] ldsv;
    wire [3:0] ldsc1;
    wire [1:0] ldsc2;
    wire overflow;
    // Instantiate the EVM module
    EVM EVM_inst (
        .v(v),
        .enable(enable),
        .reset(reset),
        .clock(clock),
        .next_result(next_result),
        .to_display(to_display),
        .ldsv(ldsv),
        .ldsc1(ldsc1),
        .ldsc2(ldsc2),
        .overflow(overflow)
    );

    // Clock generation
    always begin
        #5 clock = ~clock;
    end

    // Generate VCD file
    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(0, EVM_tb);
    end

    // Test sequence
    initial begin
        // Initialize inputs
        v = 4'b0000;
        enable = 1'b0;
        reset = 1'b0;
        next_result = 1'b0;
        clock = 1'b0;

        // Reset the system
        reset = 1'b1;
        #10 reset = 1'b0;

        // Enable voting
        enable = 1'b1;
        #10 enable = 1'b0;

        // Vote for candidate 1
        v = 4'b0001;
        #10 v = 4'b0000;

        // Vote for candidate 2
        v = 4'b0010;
        #10 v = 4'b0000;

        // Vote for candidate 3
        v = 4'b0100;
        #10 v = 4'b0000;

        // Vote for candidate 4
        v = 4'b1000;
        #20 v = 4'b0000;

      // Vote for candidate 1
        v = 4'b0001;
        #10 v = 4'b0000;

        // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;

    // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;

        // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
// Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
      // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
      // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
      // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
      // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
      // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;
      // Vote for candidate 2
        #10 v = 4'b0010;
        #10 v = 4'b0000;

       #40 enable=1'b1;
      #10 enable=1'b0;
        // Display results for candidate 1
        #10 next_result = 1'b1;
        #10 next_result = 1'b0;
            //display result for candidate 2

      #10 next_result = 1'b1;

        #10 next_result = 1'b0;
      //display result for candidate 3

        #10 next_result = 1'b1;
              #10 next_result = 1'b0;

      //display result for candidate 4
        #10 next_result = 1'b1;
                #10 next_result = 1'b0;
      // back to candidate 1
         #10 next_result = 1'b1;

        // End the simulation
        $finish;
    end
endmodule
```

Test Bench Waveforms

As can be seen form in the wavefrom and in accordance with our truth table, when enable is triggered for the first time the system goes into voting mode and votes are cased aaccording to 'v' which is in one hot notation. The voting count of each candidate increase correspondingly.
When Enable is triggered again the system goes into display mode and it shows the candidate number along with the number of votes they got during voting. When NextResult is triggered the we move on to the results for next candidate and cycle back onto 1st candidate again.

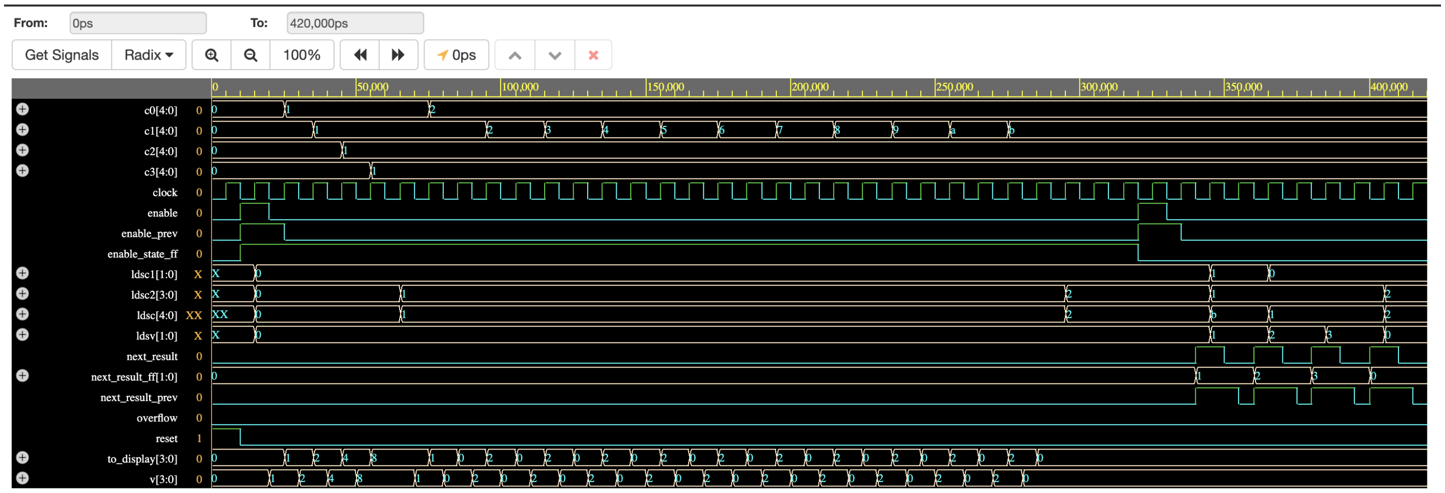

Figure 4: Waveforms for several signals

Future Prospects

• **Increasing Candidate Capacity:** The current implementation supports up to 4 candidates. Future iterations could explore expanding the system to accommodate a larger number of candidates, addressing the needs of more complex electoral processes.

• **Enhancing Security Measures:** To ensure the integrity and security of the voting process, additional security measures could be implemented, such as voter authentication, tamper-proof mechanisms, and encryption of data transmission.

• **Remote Voting Capability:** Integrating remote voting capabilities could greatly enhance the accessibility and convenience of the voting process. This could involve developing secure online voting platforms or integrating the CPLD-based system with existing electronic voting infrastructure.

• **User Interface Improvements:** While the current implementation utilizes switches and LEDs for input and output, future versions could explore more user-friendly interfaces, such as touch screens or graphical user interfaces, to improve the overall voting experience.

• **Data Analysis and Reporting:** Incorporating data analysis and reporting functionalities could provide valuable insights into voting patterns, voter turnout, and other relevant statistics. This could aid in the evaluation and improvement of the electoral process.