AMRITA VISHWA VIDYAPEETHAM

Department of ECE

19ECE383 : VLSI Design Lab

Project Report

Project Title: Secure Digital Voting System

Team Members

AM.EN.U4ECE22101 – ABHINAV K L
AM.EN.U4ECE22112 – ARDRA R NAIR
AM.EN.U4ECE22122 – HASINI N
AM.EN.U4ECE22133 – NIRANJAN
AM.EN.U4ECE22144 – TEENA S
AM.EN.U4ECE22155 – VARSHITH REDDY

Signature of faculty with Date

Secure Digital Voting System

Objective

The primary objective of this project is to design and implement a digital voting machine system using the Basys 3 FPGA board, which ensures secure, reliable, and user-friendly vote recording and display. The system aims to allow users to cast votes for three options—BJP, CONG, and NOTA—through push buttons, automatically increment the respective vote count, and provide real-time visual confirmation via LEDs and a seven-segment display.

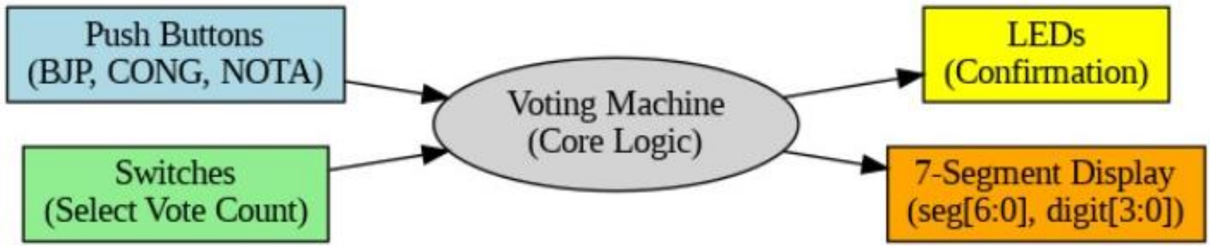
This project also aims to demonstrate the practical application of digital design concepts such as counters, multiplexers, registers, and display drivers using Verilog HDL. By dividing the system into modular components (voting logic, digit conversion, and display control), the project emphasizes modular hardware design, which enhances reusability, scalability, and debugging efficiency. Ultimately, the goal is to build a fully functional and interactive hardware-based voting system that simulates real-world electronic voting machines with a focus on accuracy, clarity, and simplicity.

Theory

This project is developed using the BASYS3 FPGA development board, which is based on the Xilinx Artix-7 FPGA. The board provides essential hardware features such as push buttons for vote input, LEDs for vote confirmation, and a seven-segment display for showing vote counts, making it ideal for implementing a digital voting machine.

The design and simulation are carried out using the Xilinx Vivado Design Suite. Vivado allows for writing Verilog code, performing synthesis, simulation, and programming the FPGA. It provides an efficient environment to test and implement the voting logic on the BASYS3 board for real-time operation.

Block Diagram



Description of Basys3 FPGA Board

The Basys 3 FPGA board is a beginner-friendly, yet powerful development platform created by Digilent, specifically designed for learning and prototyping digital logic systems. At its core is the Xilinx Artix-7 FPGA (XC7A35T-1CPG236C), a modern, low-power, high-performance chip that provides ample resources for a wide variety of digital applications. With 33,280 logic cells, numerous DSP slices, and abundant block RAM, it allows users to implement complex logic designs efficiently using Verilog or VHDL on the Vivado Design Suite provided by Xilinx.

One of the main advantages of the Basys 3 board is its rich set of onboard I/O components that enable interactive learning and experimentation. It includes 16 switches, 5 push buttons, 16 LEDs, and four 7-segment displays, which are perfect for testing logic gates, counters, state machines, and more. These features make it highly suitable for real-time user input and output, which is essential in projects like voting machines, digital clocks, or calculators. Additionally, the board has a 100 MHz onboard clock and supports custom clock generation using built-in PLLs through the Vivado software.

For communication and programming, the board comes with a USB-JTAG interface and a USB-UART bridge, allowing for seamless programming and serial communication with a PC. It also features 32 MB of non-volatile Quad-SPI flash memory, which stores the FPGA configuration and user data. Power can be supplied via micro-USB or an external source, giving users flexibility in deployment.

Another standout feature is the board's expandability. It includes four Pmod connectors, which enable users to connect a wide range of Digilent peripheral modules such as sensors, displays, or communication modules. This makes the Basys 3 not only suitable for educational use but also for building small-scale embedded systems and prototypes.

Project Description

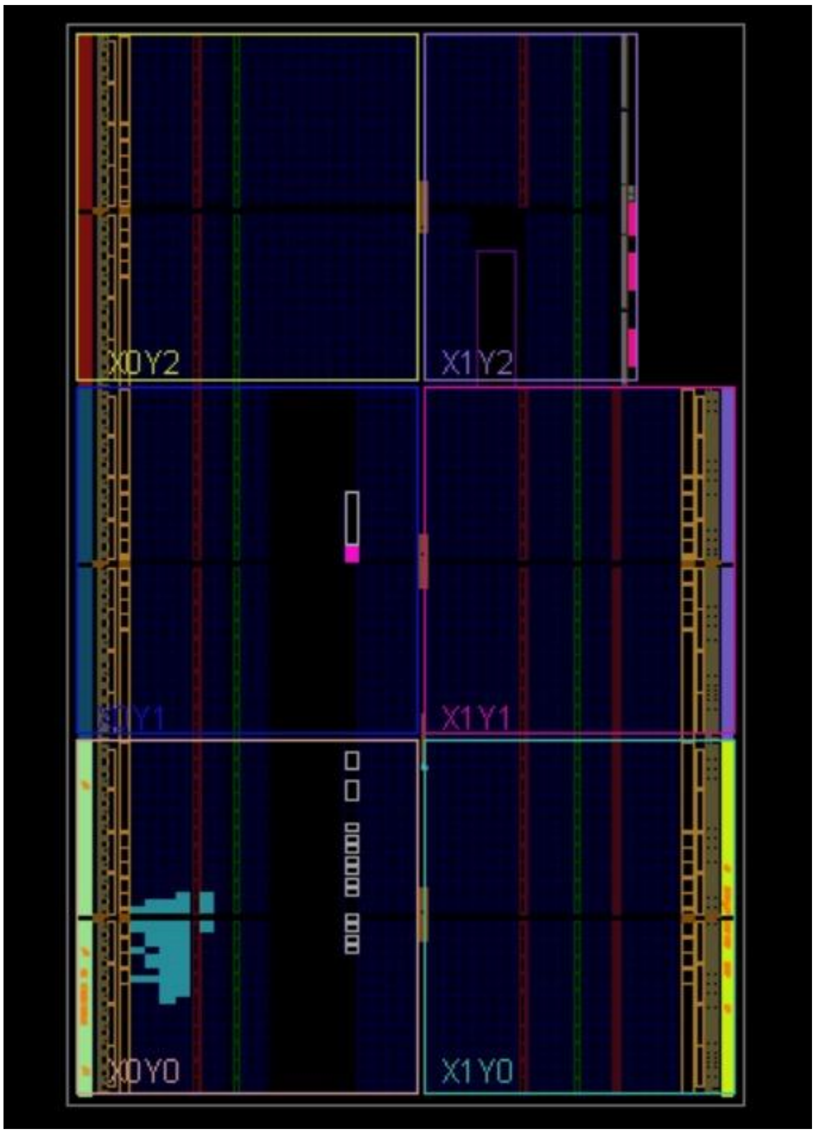
This project presents the design and implementation of a Digital Voting Machine System using digital logic modules to record, process, and display vote counts. The system is composed of three main modules: the Voting Machine Module, the Digits Module, and the Seven-Segment Display Module. Together, these modules enable secure vote casting, real-time count tracking, and dynamic visual representation of results on a digital display.

The Voting Machine Module forms the core of the system. It maintains separate registers to track vote counts for three voting options: BJP, CONG, and NOTA. Each option is associated with a push button; when pressed, the corresponding count is incremented, and a confirmation signal (e.g., BJP_CONF) is generated. This signal triggers an LED or similar indicator to provide instant visual feedback, confirming that the vote has been successfully registered. A system-wide reset function is available to clear all vote counts and restore the system to its initial state. Additionally, this module interacts with the Digits Module to determine which party's vote count is currently selected for display, based on control signals such as BJP_FIG, CONG_FIG, and NOTA_FIG.

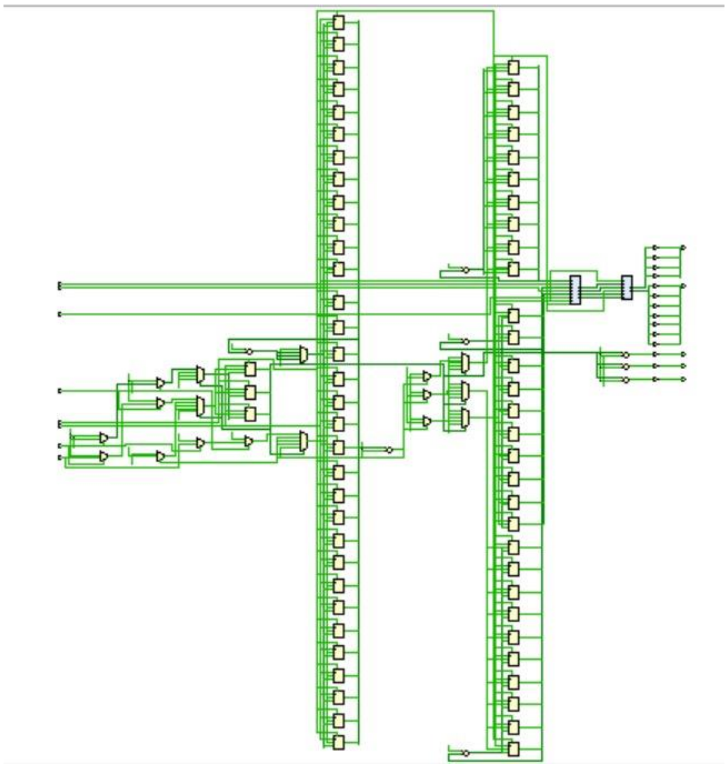
The Digits Module is responsible for converting the total vote count of the selected party into a four-digit format. It takes the vote counts of BJP, CONG, and NOTA as input and, depending on which figure select signal is active, isolates the corresponding count. This count is then broken down into thousands, hundreds, tens, and ones using arithmetic division and modulo operations. These individual digit outputs are passed to the Seven-Segment Display Module for visualization.

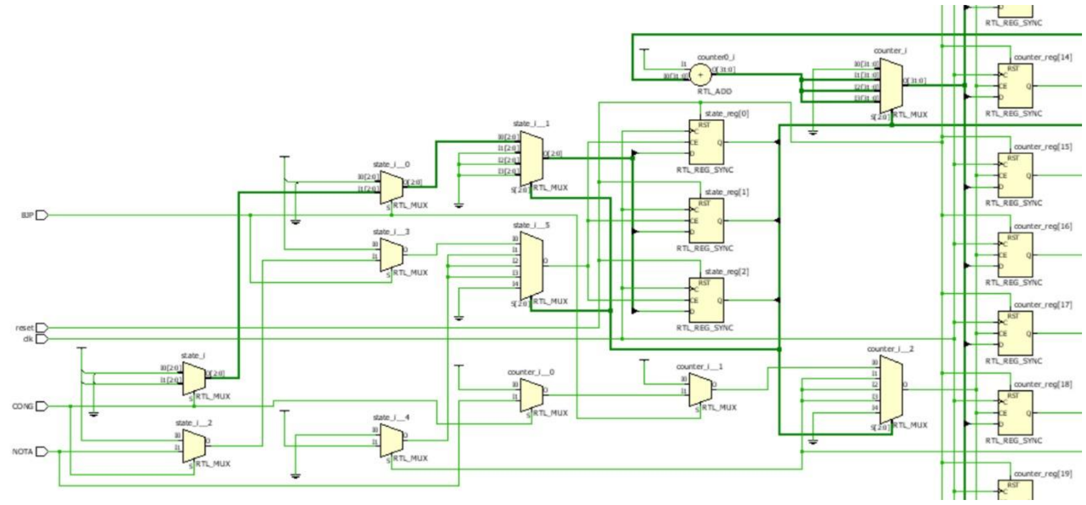
Finally, the Seven-Segment Display Module provides a visual representation of the vote count on a four-digit display. It uses a 2-bit counter (digit_select) to cycle rapidly through the thousands, hundreds, tens, and ones digits, enabling one digit at a time. This multiplexing creates the illusion of all digits being displayed simultaneously to the human eye. A timer (digit_timer) controls the refresh rate, ensuring that each digit is updated approximately every 1 ms. The digit values are decoded into corresponding seven-segment patterns using a case-based logic structure that maps binary numbers (0-9) to display configurations.

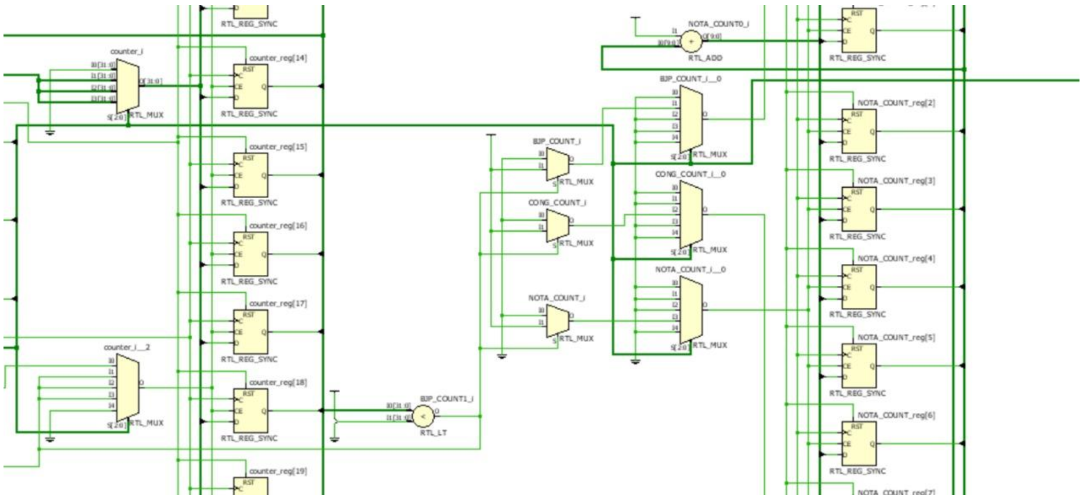
Implemented Design

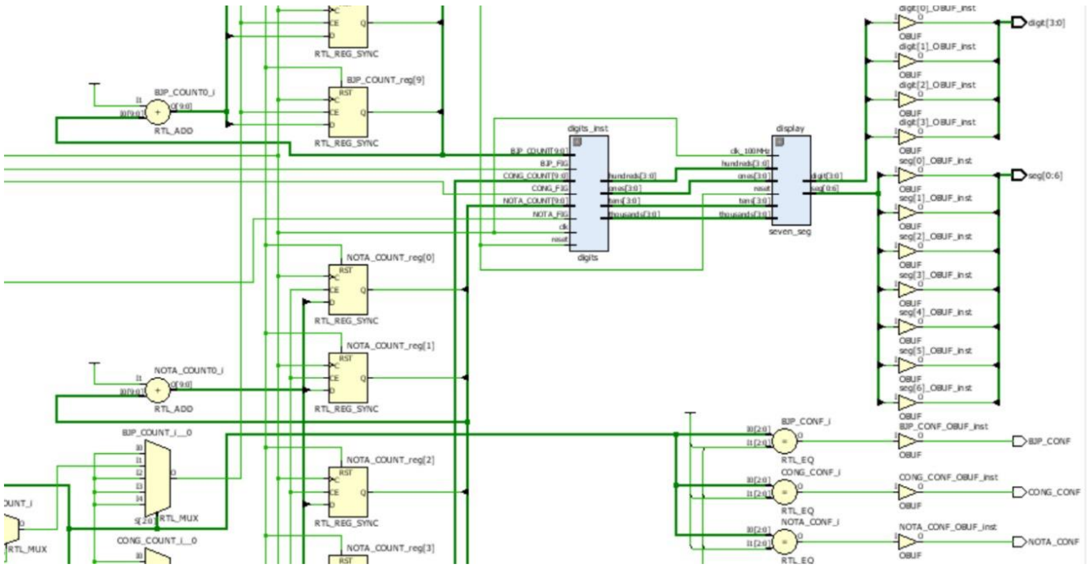


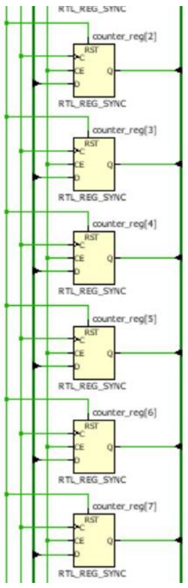
RTL Schematic Diagram

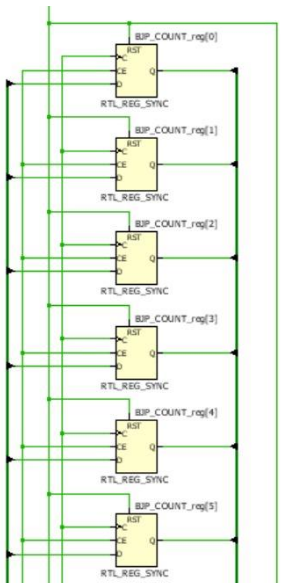


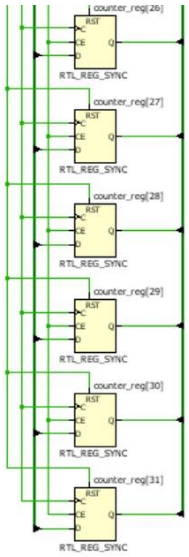


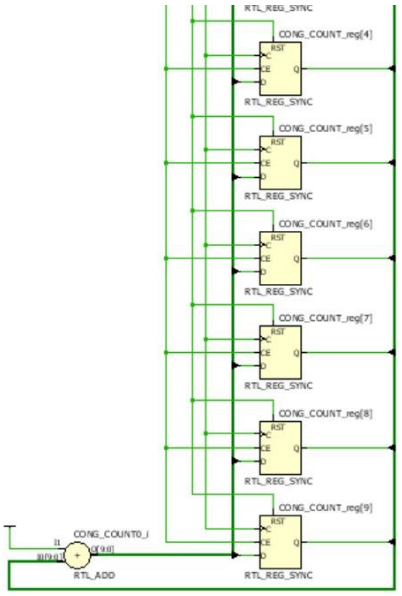



digits_inst
digits
display
seven_seg
BJP_COUNT0_i
NOTA_COUNT0_i
RTL_ADD
RTL_MUX
RTL_EQ
BJP_CONF
CONG_CONF
NOTA_CONF
digit[3:0]
seg[0:6]


counter_reg[2]
counter_reg[3]
counter_reg[4]
counter_reg[5]
counter_reg[6]
counter_reg[7]
RTL_REG_SYNC


BJP_COUNT_reg[0]
BJP_COUNT_reg[1]
BJP_COUNT_reg[2]
BJP_COUNT_reg[3]
BJP_COUNT_reg[4]
BJP_COUNT_reg[5]
RTL_REG_SYNC


counter_reg[26]
counter_reg[27]
counter_reg[28]
counter_reg[29]
counter_reg[30]
counter_reg[31]
RTL_REG_SYNC


CONG_COUNT_reg[4]
CONG_COUNT_reg[5]
CONG_COUNT_reg[6]
CONG_COUNT_reg[7]
CONG_COUNT_reg[8]
CONG_COUNT_reg[9]
CONG_COUNT0_i
RTL_ADD
RTL_REG_SYNC

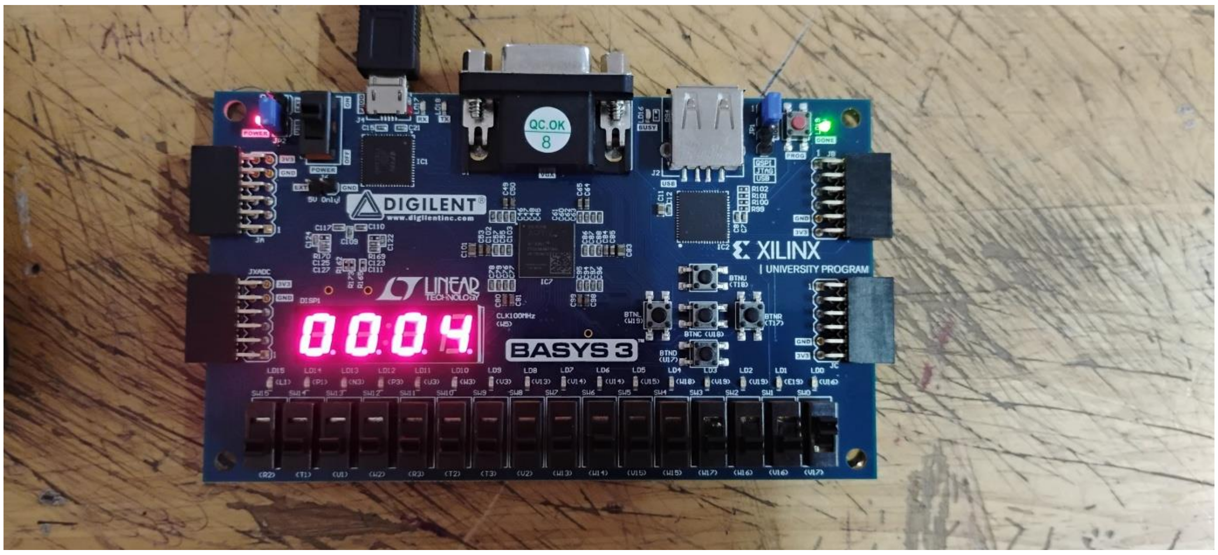
Result

The successful implementation of the Secure Digital Voting System on the Basys 3 FPGA board demonstrated the effective working of all three core modules—Voting Machine, Digits, and Seven-Segment Display. The system accurately recorded votes for the three options (BJP, CONG, and NOTA) through the push buttons and incremented the respective counters. Each vote triggered a corresponding confirmation signal that activated an LED, giving immediate feedback to the voter that their selection was registered.

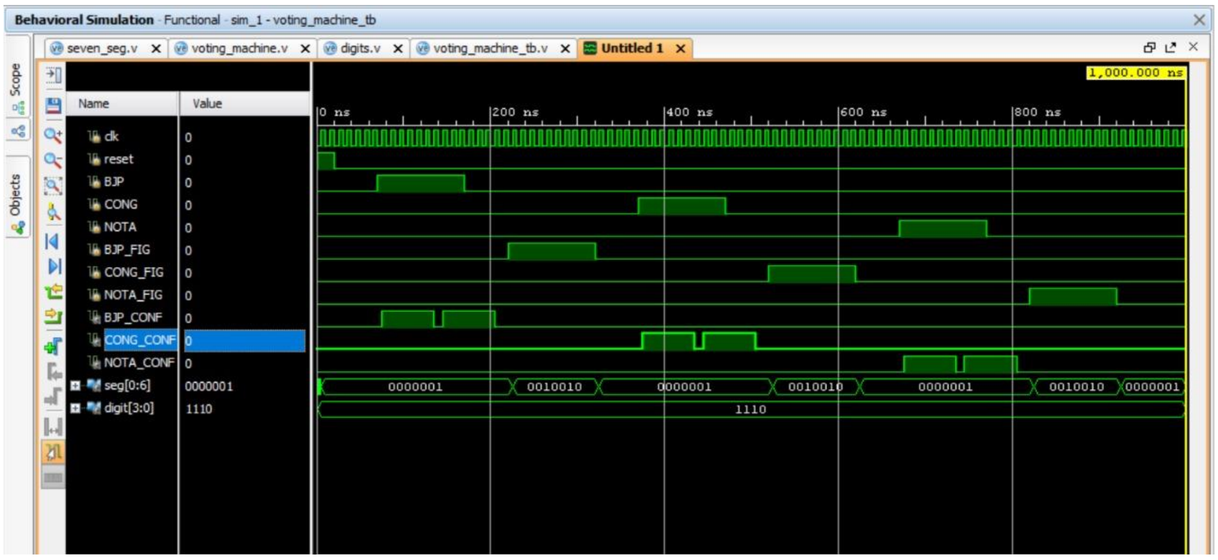
Upon selecting a party using the figure select buttons (BJP_FIG, CONG_FIG, or NOTA_FIG), the current vote count was correctly broken down into thousands, hundreds, tens, and ones by the Digits Module. These values were then continuously refreshed and displayed on the four-digit seven-segment display using a digit multiplexing technique controlled by the digit timer and digit selector.

Simulation results (waveforms) verified the correct incrementing of vote counts, proper signal assertion, and digit extraction. The physical deployment on the Basys 3 board confirmed the real-time interaction between hardware inputs (push buttons) and outputs (LEDs and seven-segment displays), showcasing a fully functional and user-interactive voting system.

Basys3 FPGA Board Implementation



Waveform Obtained



Conclusion

The implementation of the Secure Digital Voting System using the Basys 3 FPGA board successfully demonstrated the application of digital design principles in building a reliable, secure, and user-friendly hardware-based voting machine. By integrating modules for vote recording, digit conversion, and seven-segment display control, the system effectively handled real-time vote inputs, confirmed selections through visual indicators, and displayed the results in a clear, multiplexed format.

Through the use of Verilog HDL and the Vivado Design Suite, the project enabled hands-on experience with RTL design, synthesis, simulation, and hardware testing. Each component functioned as intended, with seamless coordination between the modules and accurate output display. The modular approach of the system ensured clarity, scalability, and ease of debugging, making it suitable for academic learning and prototype-level deployment.

Overall, the project achieved its goals of combining theoretical knowledge with practical FPGA implementation to design a secure digital voting system. It reinforced key concepts in VLSI design and digital logic while showcasing the capabilities of the Basys 3 board as a platform for developing real-world embedded applications.

References

1. Kumar, Mritunjay, et al. "FPGA Based Voting Machine." Kilby 100 (2023): 7th.
2. Sriraam, V. S., V. R. Raghavi, and S. R. Ramesh. "Implementation of Electronic Voting Machine in FPGA." 2024 International Conference on Computer, Electronics, Electrical Engineering & their Applications (IC2E3). IEEE, 2024.
3. https://github.com/dhanush-271/Voting_Machine_Basys3/blob/main/README.md

Acknowledgement

We would like to express our sincere gratitude to Dr. Sreenidhi P R, Mrs. Anjana J G and other lab staffs for their invaluable guidance, support, and encouragement throughout the development of this project. Their expertise and insights were crucial in refining the concept and ensuring its successful implementation. We also extend our thanks to our peers for their constructive suggestions and collaborative efforts, which greatly enhanced the quality of the work. Finally, we are deeply grateful to our institution for providing the necessary resources and a conducive environment for research and innovation.

Appendix: Code with Comments

- **Voting Machine Design Code**

```
module voting_machine(
   input clk, reset,                   // Clock and asynchronous reset
   input BJP, CONG, NOTA,              // Push button inputs for BJP, CONG, and
NOTA
   input BJP_FIG, CONG_FIG, NOTA_FIG,    // Signals to select which party's count
to display
   output BJP_CONF, CONG_CONF, NOTA_CONF,// Confirmation signals (e.g.,
LED indicators)
   output [0:6] seg,                   // Seven segment display segments (active low)
   output [3:0] digit                  // Digit enable signals for 4-digit display
);
   parameter CONFIRM_TIME = 50_000_000;  // Time duration to show confirmation
(~1s @50MHz clock)

   reg [9:0] BJP_COUNT=0, NOTA_COUNT=0, CONG_COUNT=0; // Vote counters
for each option
   wire [3:0] ones,tens,hundreds,thousands;          // Individual digit outputs for display

   // Instantiation of digit converter module
   digits digits_inst(
      clk, reset,
      BJP_FIG, CONG_FIG, NOTA_FIG,        // Figure select inputs
      BJP_COUNT, NOTA_COUNT, CONG_COUNT,   // Vote counts as inputs
      ones, tens, hundreds, thousands      // Digit outputs
   );

   // Instantiation of seven-segment display driver
   seven_seg display(
      clk, reset,
      ones, tens, hundreds, thousands,     // Digits to be displayed
      seg, digit                          // Seven-segment control outputs
   );

   reg [31:0] counter = 0;    // Timer counter for confirmation period
   reg [2:0] state = 0;       // FSM state variable: 0=Idle, 1=BJP, 2=CONG, 3=NOTA
```

```
// Assign confirmation signals based on current state
assign BJP_CONF  = (state == 1);  // BJP button pressed
assign CONG_CONF = (state == 2);  // CONG button pressed
assign NOTA_CONF = (state == 3);  // NOTA button pressed

// Main FSM logic
always @(posedge clk) begin
    if(reset) begin
        // Reset all counters and state machine
        BJP_COUNT <= 0;
        CONG_COUNT <= 0;
        NOTA_COUNT <= 0;
        counter <= 0;
        state <= 0;
    end else begin
        case(state)
            0: begin  // Idle state: wait for any button press
                if(BJP) begin
                    counter <= 0;
                    state <= 1;  // Transition to BJP vote state
                end else if(CONG) begin
                    counter <= 0;
                    state <= 2;  // Transition to CONG vote state
                end else if(NOTA) begin
                    counter <= 0;
                    state <= 3;  // Transition to NOTA vote state
                end
            end

            1: begin  // BJP confirmation state
                if(counter < CONFIRM_TIME) begin
                    counter <= counter + 1;  // Wait until confirmation time completes
                end else begin
                    BJP_COUNT <= BJP_COUNT + 1; // Register the vote
                    state <= 0;                 // Return to Idle
                end
            end
            2: begin  // CONG confirmation state
              if(counter < CONFIRM_TIME) begin
                counter <= counter + 1;  // Wait for confirmation duration
              end else begin
                CONG_COUNT <= CONG_COUNT + 1; // Register CONG vote
                state <= 0;                 // Return to Idle
              end
            end

            3: begin  // NOTA confirmation state
              if(counter < CONFIRM_TIME) begin
                counter <= counter + 1;  // Wait for confirmation duration
              end else begin
                NOTA_COUNT <= NOTA_COUNT + 1; // Register NOTA vote
                state <= 0;                 // Return to Idle
              end
            end
          endcase
        end
      end
    endmodule
```

- **Digits Design Code**

```
module digits(
  input clk, reset,                        // Clock and reset signals
  input BJP_FIG, CONG_FIG, NOTA_FIG,            // Figure select signals to
choose which count to display
  input [9:0] BJP_COUNT, NOTA_COUNT, CONG_COUNT,  // 10-bit vote
counts for each option
  output reg [3:0] ones, tens, hundreds, thousands // Outputs for individual
digits
);

  reg [9:0] temp_count;  // Temporary register to hold the selected count

  // Select the count value based on active figure select signal
  always @(posedge clk) begin
    if(reset) begin
      temp_count <= 0;  // Reset temp_count on reset
```

```
        end else begin
            case({BJP_FIG, CONG_FIG, NOTA_FIG})  // Priority encoder based
on select inputs
                3'b100: temp_count <= BJP_COUNT;      // Select BJP count if
BJP_FIG is high
                3'b010: temp_count <= CONG_COUNT;     // Select CONG count if
CONG_FIG is high
                3'b001: temp_count <= NOTA_COUNT;     // Select NOTA count if
NOTA_FIG is high
                default: temp_count <= 0;             // If none or multiple selected,
default to 0
            endcase
        end
    end

    // Combinational logic to split temp_count into individual digits
    always @* begin
        ones = temp_count % 10;                        // Extract ones place
        tens = (temp_count / 10) % 10;                 // Extract tens place
        hundreds = (temp_count / 100) % 10;            // Extract hundreds place
        thousands = (temp_count / 1000) % 10;          // Extract thousands place
    end
endmodule
```

- **Seven Segment Design Code**

```
`timescale 1ns / 1ps

module seven_seg(
    input clk_100MHz,                          // 100 MHz system clock input
    input reset,                               // Asynchronous reset signal
    input [3:0] ones, tens, hundreds, thousands,// 4-bit digit inputs for each place
    output reg [0:6] seg,                      // 7-segment segment control outputs
(active low for common anode)
    output reg [3:0] digit                     // Digit select outputs (active low for
common anode)
);

    // 7-segment display patterns for common anode configuration
    parameter ZERO  = 7'b0000001;  // Display '0'
    parameter ONE   = 7'b1001111;  // Display '1'
parameter TWO   = 7'b0010010;  // Display '2'
parameter THREE = 7'b0000110;  // Display '3'
parameter FOUR  = 7'b1001100;  // Display '4'
parameter FIVE  = 7'b0100100;  // Display '5'
parameter SIX   = 7'b0100000;  // Display '6'
parameter SEVEN = 7'b0001111;  // Display '7'
parameter EIGHT = 7'b0000000;  // Display '8'
parameter NINE  = 7'b0000100;  // Display '9'

reg [1:0] digit_select;       // 2-bit counter to select which digit is active
reg [16:0] digit_timer;       // Timer to control refresh rate (counts to ~1ms)

// Timer and digit selector logic
always @(posedge clk_100MHz or posedge reset) begin
    if(reset) begin
        digit_select <= 0;     // Reset digit selector
        digit_timer <= 0;      // Reset timer
    end else begin
        if(digit_timer == 99_999) begin // ~1 ms at 100 MHz
            digit_timer <= 0;           // Reset timer
            digit_select <= digit_select + 1; // Move to next digit
        end else begin
            digit_timer <= digit_timer + 1;   // Increment timer
        end
    end
end

// Activate one digit at a time using active-low logic
always @* begin
    case(digit_select)
        2'b00: digit = 4'b1110;  // Enable ones place (digit 0)
        2'b01: digit = 4'b1101;  // Enable tens place (digit 1)
        2'b10: digit = 4'b1011;  // Enable hundreds place (digit 2)
        2'b11: digit = 4'b0111;  // Enable thousands place (digit 3)
    endcase
end

// Assign segment pattern for the active digit
always @* begin
```

```
      case(digit_select)
         2'b00: seg = get_seg(ones);      // Display ones place digit
         2'b01: seg = get_seg(tens);      // Display tens place digit
         2'b10: seg = get_seg(hundreds);  // Display hundreds place digit
         2'b11: seg = get_seg(thousands); // Display thousands place digit
      endcase
   end

   // Function to return 7-segment pattern for a given number (0-9)
   function [6:0] get_seg(input [3:0] num);
      case(num)
         0: get_seg = ZERO;
         1: get_seg = ONE;
         2: get_seg = TWO;
         3: get_seg = THREE;
         4: get_seg = FOUR;
         5: get_seg = FIVE;
         6: get_seg = SIX;
         7: get_seg = SEVEN;
         8: get_seg = EIGHT;
         9: get_seg = NINE;
         default: get_seg = 7'b1111111; // Blank display for invalid input
      endcase
   endfunction

endmodule
```

- **Voting Machine Testbench Code**

```
`timescale 1ns / 1ps

module voting_machine_tb;

   // Clock and control signal declarations
   reg clk = 0;                        // System clock initialized to 0
   reg reset;                          // Reset signal
   reg BJP, CONG, NOTA;                // Push button inputs for voting
   reg BJP_FIG, CONG_FIG, NOTA_FIG;    // Signals to select party vote
count for display
```

```
   wire BJP_CONF, CONG_CONF, NOTA_CONF;// Confirmation outputs for
each party
   wire [0:6] seg;                    // Seven-segment segment outputs
   wire [3:0] digit;                  // Active digit control outputs

   // Instantiate the voting machine with a reduced CONFIRM_TIME for fast
simulation
   voting_machine #(.CONFIRM_TIME(5)) uut (  // Only 5 cycles needed to
confirm vote
       .clk(clk),
       .reset(reset),
       .BJP(BJP),
       .CONG(CONG),
       .NOTA(NOTA),
       .BJP_FIG(BJP_FIG),
       .CONG_FIG(CONG_FIG),
       .NOTA_FIG(NOTA_FIG),
       .BJP_CONF(BJP_CONF),
       .CONG_CONF(CONG_CONF),
       .NOTA_CONF(NOTA_CONF),
       .seg(seg),
       .digit(digit)
   );


   // Generate a 100MHz clock: toggles every 5 ns
   always #5 clk = ~clk;

   initial begin
       // Initialize all signals
       reset = 1;                     // Assert reset
       BJP = 0; CONG = 0; NOTA = 0;    // No buttons pressed initially
       BJP_FIG = 0; CONG_FIG = 0; NOTA_FIG = 0;  // No party selected for
display

       #20 reset = 0;                 // Deassert reset after 20 ns

       // ------------------- Test 1: BJP Vote -------------------
       #50 BJP = 1;                   // Simulate pressing BJP button
       #100 BJP = 0;                  // Release button after 100 ns
```

```
        #50 BJP_FIG = 1;               // Select BJP figure for display
        #100 BJP_FIG = 0;              // Deactivate display selection

        // ------------------- Test 2: CONG Vote ------------------
        #50 CONG = 1;                  // Simulate pressing CONG button
        #100 CONG = 0;                 // Release button
        #50 CONG_FIG = 1;              // Select CONG figure for display
        #100 CONG_FIG = 0;             // Deactivate display selection

        // ------------------- Test 3: NOTA Vote ------------------
        #50 NOTA = 1;                  // Simulate pressing NOTA button
        #100 NOTA = 0;                 // Release button
        #50 NOTA_FIG = 1;              // Select NOTA figure for display
        #100 NOTA_FIG = 0;             // Deactivate display selection

        #200 $finish;                  // End simulation after final delay
    end
endmodule
```