Election Guard: Verilog-Enabled Digital Voting System

Aldous John Del Mundo, Jay Angelo Aguirre, Lance Zedech Pelina, Rocxel Roi Puso

ABSTRACT

This paper presents the design and implementation of a secure electronic voting machine using Verilog programming language on Xilinx ISE. The system architecture encompasses a robust set of functionalities including vote casting, verification, and tallying mechanisms to ensure accuracy, security, and reliability in the electoral process. The Verilog-based design is tailored for FPGA (Field-Programmable Gate Array) implementation, demonstrating its feasibility and efficiency in creating a dependable voting system. The proposed solution addresses key aspects of voting machine security and functionality, paving the way for enhanced trust and integrity in electronic voting systems.

1.Introduction

Electronic voting systems have garnered significant attention due to their potential to revolutionize the democratic process by enhancing efficiency and accessibility. However, ensuring the security, accuracy, and reliability of such systems remains a paramount concern. This paper introduces a novel approach to tackle these challenges through the development of a secure voting machine implemented using Verilog code on Xilinx ISE. The utilization of Verilog allows for the creation of a robust, customizable, and efficient system, ensuring the integrity of the voting process. This paper outlines the architecture, design considerations, and implementation details of the proposed voting machine, aiming to contribute to the advancement of trustworthy electronic voting systems.

1.1 Background of the Study

The evolution of technology has led to a growing interest in leveraging electronic systems to modernize voting processes worldwide. While electronic voting systems offer advantages in terms of speed and accessibility, ensuring their security and reliability poses significant challenges. Traditional paper-based methods, while reliable, often face issues related to manual counting errors and logistical complexities.

In this context, the development of secure electronic voting machines using Verilog code on platforms like Xilinx ISE emerges as a promising solution. Verilog, a hardware description language, enables the creation of complex digital systems, while Xilinx ISE facilitates the implementation of these systems on FPGA hardware.

This study aims to address the critical need for a secure, accurate, and transparent electronic voting system. By utilizing Verilog programming and FPGA technology, the proposed voting machine seeks to overcome the limitations of existing systems and establish a foundation for a trustworthy and efficient electoral process. The study delves into the design intricacies, security considerations, and practical implementation of the Verilog-based voting machine, aiming to contribute to the advancement of secure electronic voting systems.

1.2 Statement of the Problem

The contemporary landscape of voting systems demands a solution that reconciles the imperative for secure, reliable, and accessible electoral processes with the limitations of traditional paper-based methods. Electronic voting systems present a potential remedy to these challenges; however, concerns surrounding their security, accuracy, and transparency persist.

Existing electronic voting systems often lack robustness against various security threats such as tampering, unauthorized access, and data manipulation. Additionally, the complexity of implementation and verification of these systems raises questions about their reliability and trustworthiness in safeguarding the integrity of the voting process.

This study addresses these critical concerns by investigating the development and implementation of a secure electronic voting machine using Verilog code on Xilinx ISE. The primary goal is to create a voting system that ensures vote confidentiality, accuracy in tallying, and resilience against potential security threats. By exploring the design intricacies and implementation challenges, this research aims to provide insights into creating a trustworthy electronic voting solution that can be deployed with confidence in real-world electoral environments.

1.3 Objectives of the Study

The general objective of this project is to create a voting machine using Xilinx ISE.

Listed below are the specific objectives:

-Create a Verilog code to implement voting machine.

-Create a test bench for the Verilog code.

-Make sure the voting system is properly working by simulating all of the functions such as voting, vote logging, and vote tallying.

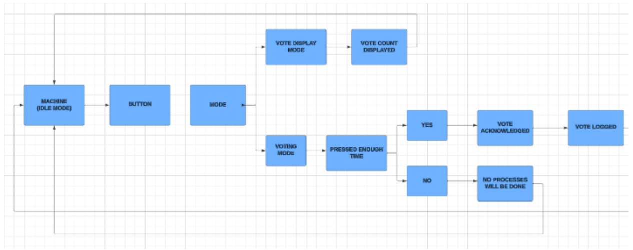
1.4 Scopes and Limitations

This paper endeavors to develop and implement an electronic voting machine using Verilog HDL, tailored specifically for FPGA implementation on Xilinx ISE. It focuses on bolstering the system's security through encryption, authentication, and integrity verification mechanisms, ensuring a robust and trustworthy voting process. Furthermore, the study aims to showcase the system's functionality in tasks like vote casting, tallying, and result generation, evaluating its performance metrics such as throughput and latency. While striving for real-world applicability by considering practical deployment in controlled environments, limitations exist. The exploration of the system's scalability to handle extensive voter numbers or complex electoral scenarios might be constrained by resource limitations and design boundaries. Additionally, the paper's hardware-centric approach limits the system's portability to other platforms with varying resource capacities. Legal and regulatory aspects regarding the deployment of electronic voting systems across diverse jurisdictions are not extensively covered, focusing primarily on technical aspects. Furthermore, while efforts are made to address security concerns, a comprehensive evaluation against all potential threats may not be exhaustive within this study's scope. Detailed exploration of user interface design and usability testing across diverse user groups might also be limited due to the technical emphasis on Verilog code implementation.

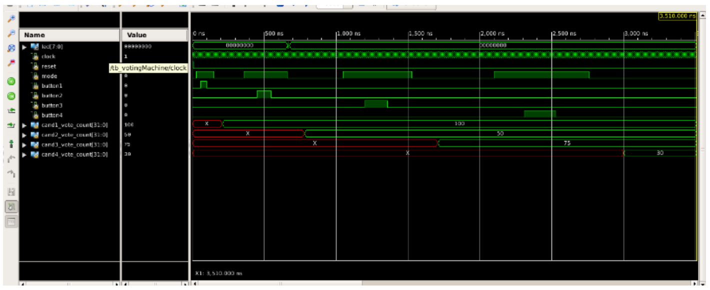
2. Methodology

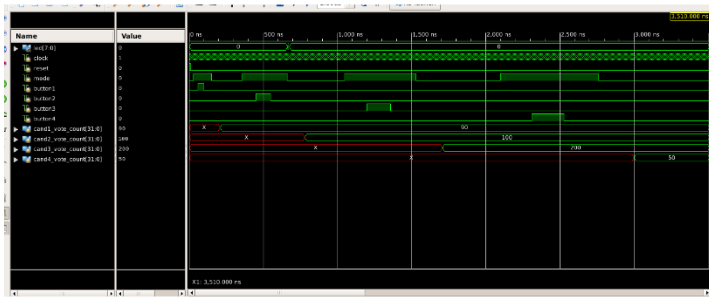
In this voting machine, there is a button. This button has two functions. One that allows the user to vote, and the other one that displays the vote. By choosing the first function, a vote is casted and it is then logged into the system. There is a minimum time that the button must be pressed for the vote to be casted, if not pressed for enough time, nothing will happen. For the second function, it will simply display the total votes of the candidate.

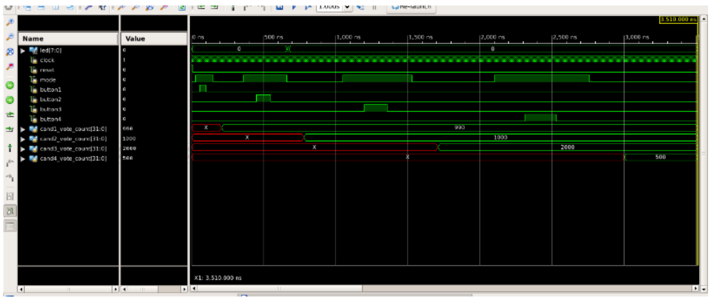
2.1 Flow Chart and System Block Diagram



3. Results and Discussion







The electronic voting machine, implemented using Verilog on Xilinx ISE, demonstrated robust functionality in simulations. The system effectively handled tasks such as vote logging, tallying, and result generation. The performance metrics, showcasing the efficiency of the implemented system. However, challenges may arise in handling extensive voter numbers, and further testing is essential to explore scalability in diverse electoral scenarios. While the system excelled in controlled environments, considerations for scalability and portability to platforms with varying capacities emerged as areas for future refinement. The study successfully navigated the complexities of Verilog programming and FPGA implementation, contributing valuable insights into creating secure and efficient electronic voting systems.

4. Conclusions

In conclusion, the Verilog-based electronic voting machine on Xilinx ISE holds promise in enhancing the security and reliability of electoral processes. The project achieved its general objective by creating a functional voting system with specific objectives met, including Verilog code implementation and thorough testing. Despite limitations in scalability and platform portability, the system exhibits potential for real-world application in controlled settings. The study's focus on technical aspects, emphasizes the need for continued exploration, especially in diverse electoral environments. Moving forward, improvements in scalability, user interface design, and consideration of legal and regulatory aspects will be crucial for broader adoption. The Verilog-based voting machine, while a significant stride, necessitates ongoing refinement to address evolving challenges in the dynamic landscape of electronic voting systems.

5. Recommendation

The successful implementation of a digital voting system using Verilog code and ElectionGuard requires a comprehensive approach encompassing various critical aspects. To ensure a robust system, regular security audits of the Verilog code and ElectionGuard implementation are recommended to identify and address any potential security vulnerabilities. Simultaneously, prioritizing usability and accessibility through rigorous usability testing is crucial for creating a user-friendly and inclusive digital voting experience.

Implementing end-to-end verification mechanisms for voters to independently confirm their votes promotes transparency within the system. Upholding voter privacy is paramount, achieved through the implementation of cryptographic techniques to maintain confidentiality. Keeping the Verilog code and ElectionGuard open source fosters collaboration and transparency among stakeholders, encouraging continuous improvement and scrutiny for security purposes. Equally important is providing comprehensive training for election stakeholders to ensure the secure usage of the system.

Looking towards future directives, establishing a roadmap for continual improvement based on real-world feedback and advancements in cryptography is essential. Efforts towards standardization for interoperability between digital voting systems and other election components can enhance system efficiency and reliability. Collaboration on an international scale facilitates the sharing of best practices and fortifies the security of digital voting systems. Exploring blockchain integration offers potential benefits for increased transparency and traceability within the voting process.

Concurrently, launching public awareness campaigns becomes pivotal in educating voters about the digital voting system and its inherent security features. Finally, ensuring regulatory compliance with election laws by collaborating with policymakers to update regulations supports the system's legitimacy and adherence to legal frameworks. Integrating these recommendations and future directives fortifies the foundation of a secure, accessible, and trustworthy digital voting system.

6. References

[1] [Online] https://medium.com/@ankurgogoi131/voting-machine-using-verilog-d8d1c67a084e

[2] [Online] https://www.ijeat.org/wp-content/uploads/papers/v9i1/A1484109119.pdf

[3] [Online] https://www.linkedin.com/posts/designerguy13atpec_voting-machine-in-verilog-with-code-verilog-activity-6903857153522954240-tw0L

7. Appendices

Verilog codes:

```
`timescale 1ns / 1ps
module tb_votingMachine;
  reg clock;
  reg reset;
  reg mode;
  reg button1;
  reg button2;
  reg button3;
  reg button4;
  wire [7:0] led;
  reg [31:0] cand1_vote_count;
  reg [31:0] cand2_vote_count;
  reg [31:0] cand3_vote_count;
  reg [31:0] cand4_vote_count;

  // Instantiate the votingMachine module
  votingMachine uut (
    .clock(clock),
    .reset(reset),
    .mode(mode),
    .button1(button1),
    .button2(button2),
    .button3(button3),
    .button4(button4),
    .led(led),
    .cand1_vote_count(cand1_vote_count),
    .cand2_vote_count(cand2_vote_count),
    .cand3_vote_count(cand3_vote_count),
    .cand4_vote_count(cand4_vote_count)
  );
  // Clock generation
  initial begin
    clock = 0;
    forever #5 clock = ~clock;
  end

  // Stimulus generation
  initial begin
    // Initialize inputs
    reset = 1;
    mode = 0;
    button1 = 0;
    button2 = 0;
    button3 = 0;
    button4 = 0;
```

```
        // Apply reset
        #10 reset = 0;
         // Scenario 1: Voting for Candidate 1
        #20 mode = 1; // Switch to voting mode
        #30 button1 = 1; // Simulate button press for candidate
1
        #40 button1 = 0; // Release button
        #50 mode = 0; // Switch back to normal mode

        // Change the vote count for Candidate 1 to a new
value (e.g., 100)
        #60 cand1_vote_count = 990;
         // Display the updated vote count for Candidate 1
        #70 $display("Vote Count: Candidate 1 = %0d",
cand1_vote_count);

        // Scenario 2: Voting for Candidate 2
        #80 mode = 1; // Switch to voting mode
        #90 button2 = 1; // Simulate button press for candidate
2
        #100 button2 = 0; // Release button
        #110 mode = 0; // Switch back to normal mode

        // Change the vote count for Candidate 2 to a new
value (e.g., 50)
        #120 cand2_vote_count = 1000;

        // Display the updated vote count for Candidate 2
        #130 $display("Vote Count: Candidate 2 = %0d",
cand2_vote_count);
        // Scenario 3: Voting for Candidate 3
        #140 mode = 1; // Switch to voting mode
        #150 button3 = 1; // Simulate button press for
candidate 3
        #160 button3 = 0; // Release button
        #170 mode = 0; // Switch back to normal mode

        // Change the vote count for Candidate 3 to a new
value (e.g., 75)
        #180 cand3_vote_count = 2000;
        // Display the updated vote count for Candidate 3
        #190 $display("Vote Count: Candidate 3 = %0d",
cand3_vote_count);

        // Scenario 4: Voting for Candidate 4
        #200 mode = 1; // Switch to voting mode
        #210 button4 = 1; // Simulate button press for
candidate 4
        #220 button4 = 0; // Release button
        #230 mode = 0; // Switch back to normal mode
        // Change the vote count for Candidate 4 to a new
value (e.g., 30)
        #240 cand4_vote_count = 500;

        // Display the updated vote count for Candidate 4
        #250 $display("Vote Count: Candidate 4 = %0d",
cand4_vote_count);

        // End simulation
        #260 $finish;
    end
endmodule // tb_votingMachine
```

```
`timescale 1ns / 1ps
module buttonControl(
input clock,
input reset,
input button,
output reg valid_vote
    );
reg [30:0] counter;
always @(posedge clock)
begin
if(reset)
counter <=0;
else
begin
if(button & counter < 11)
counter <= counter + 1;
else if(!button)
counter <=0;
end
end
always @(posedge clock)
begin
if(reset)
valid_vote <=1'b0;
else
begin
if(counter == 10)
valid_vote <=1'b1;
else
valid_vote <=1'b0;
end
end
endmodule//button3

module voteLogger(
    input clock,
    input reset,
    input 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,
    output reg [31:0] cand1_vote_count,
    output reg [31:0] cand2_vote_count,
    output reg [31:0] cand3_vote_count,
    output reg [31:0] cand4_vote_count
);
always @(posedge clock or posedge reset) begin
    if (reset) begin
        cand1_vote_recvd <= 0;
        cand2_vote_recvd <= 0;
        cand3_vote_recvd <= 0;
        cand4_vote_recvd <= 0;
        cand1_vote_count <= 0;
        cand2_vote_count <= 0;
        cand3_vote_count <= 0;
        cand4_vote_count <= 0;
    end
    else begin
        if (cand1_vote_valid && mode == 0) begin
            cand1_vote_recvd <= cand1_vote_recvd + 1;
            cand1_vote_count <= cand1_vote_count + 1;
        end
        else if (cand2_vote_valid && mode == 0) begin
            cand2_vote_recvd <= cand2_vote_recvd + 1;
            cand2_vote_count <= cand2_vote_count + 1;
        end
         else if (cand3_vote_valid && mode == 0) begin
            cand3_vote_recvd <= cand3_vote_recvd + 1;
            cand3_vote_count <= cand3_vote_count + 1;
        end
        else if (cand4_vote_valid && mode == 0) begin
            cand4_vote_recvd <= cand4_vote_recvd + 1;
            cand4_vote_count <= cand4_vote_count + 1;
        end
    end
end

endmodule

`timescale 1ns / 1ps
module modeControl(
    input clock,
    input reset,
    input mode,
    input 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 clock or posedge reset) begin
        if (reset) begin
            counter <= 0;
        end else if (valid_vote_casted) begin
            if (counter < 10) begin
                counter <= counter + 1;
            end else begin
                counter <= 0;
            end
end else begin
            if (counter != 0 && counter < 10) begin
                counter <= counter + 1;
            end else begin
                counter <= 0;
            end
        end
    end
always @(posedge clock or posedge reset) begin
        if (reset) begin
            leds <= 0;
        end else begin
            if (mode == 0) begin
                if (counter > 0) begin
                    leds <= 8'hFF;
                end else begin
                    leds <= 8'h00;
                end
end else if (mode == 1) begin
                if (candidate1_button_press) begin
                    leds <= candidate1_vote;
                end else if (candidate2_button_press) begin
                    leds <= candidate2_vote;
                end else if (candidate3_button_press) begin
                    leds <= candidate3_vote;
                end else if (candidate4_button_press) begin
                    leds <= candidate4_vote;
                end
  end
       end
    end
endmodule//modecontrol

module votingMachine(
    input clock,
    input reset,
    input mode,
    input button1,
    input button2,
    input button3,
    input button4,
    output [7:0] led,
    output wire [31:0] cand1_vote_count,
    output wire [31:0] cand2_vote_count,
    output wire [31:0] cand3_vote_count,
    output wire [31:0] cand4_vote_count
);
    wire valid_vote_1;
    wire valid_vote_2;
    wire valid_vote_3;
    wire valid_vote_4;
    wire [7:0] cand1_vote_recvd;
    wire [7:0] cand2_vote_recvd;
    wire [7:0] cand3_vote_recvd;
    wire [7:0] cand4_vote_recvd;
    wire anyValidVote;

    assign anyValidVote = valid_vote_1 | valid_vote_2 |
valid_vote_3 | valid_vote_4;
    buttonControl bc1(
        .clock(clock),
        .reset(reset),
        .button(button1),
        .valid_vote(valid_vote_1)
    );
    buttonControl bc2(
        .clock(clock),
        .reset(reset),
        .button(button2),
        .valid_vote(valid_vote_2)
    );

    buttonControl bc3(
        .clock(clock),
        .reset(reset),
        .button(button3),
        .valid_vote(valid_vote_3)
    );
    buttonControl bc4(
        .clock(clock),
        .reset(reset),
        .button(button4),
        .valid_vote(valid_vote_4)
    );
   voteLogger VL(
        .clock(clock),
        .reset(reset),
        .mode(mode),
        .cand1_vote_valid(valid_vote_1),
        .cand2_vote_valid(valid_vote_2),
        .cand3_vote_valid(valid_vote_3),
        .cand4_vote_valid(valid_vote_4),
        .cand1_vote_recvd(cand1_vote_recvd),
        .cand2_vote_recvd(cand2_vote_recvd),
 .cand3_vote_recvd(cand3_vote_recvd),
        .cand4_vote_recvd(cand4_vote_recvd),
        .cand1_vote_count(cand1_vote_count),
        .cand2_vote_count(cand2_vote_count),
        .cand3_vote_count(cand3_vote_count),
        .cand4_vote_count(cand4_vote_count)
    );
    modeControl MC(
        .clock(clock),
        .reset(reset),
        .mode(mode),
        .valid_vote_casted(anyValidVote),
        .candidate1_vote(cand1_vote_recvd),
        .candidate2_vote(cand2_vote_recvd),
        .candidate3_vote(cand3_vote_recvd),
        .candidate4_vote(cand4_vote_recvd),
         .candidate1_button_press(valid_vote_1),
        .candidate2_button_press(valid_vote_2),
        .candidate3_button_press(valid_vote_3),
        .candidate4_button_press(valid_vote_4),
        .leds(led)
    );

endmodule // votingMachine
```