BHE3233 DIGITAL SYSTEM DESIGN

PROJECT RESULT YOUTUBE LINK

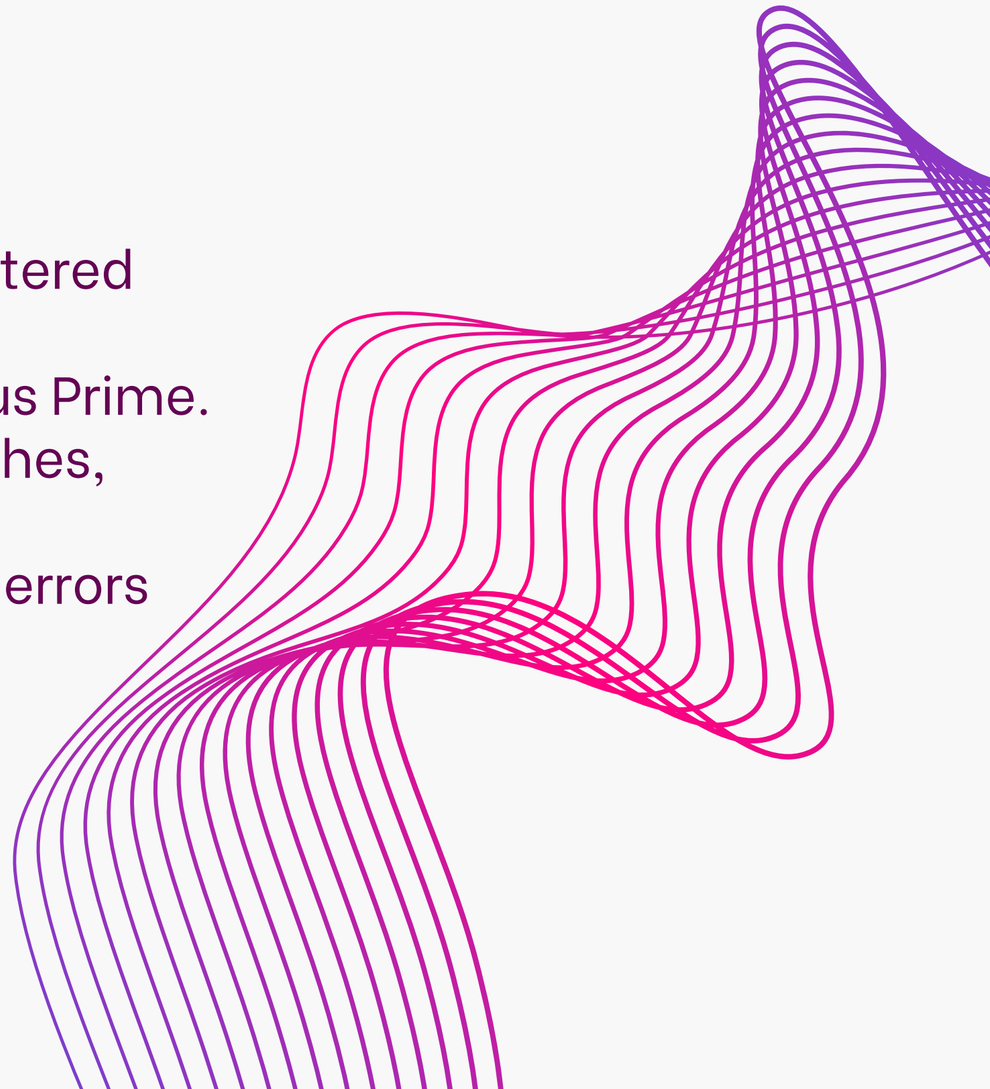
PASSWORD PROTECTED DIGITAL LOCK

JUNI INSYIRAH BINTI JAMRI HC22032
HANUM ANISAH BINTI AHMAD JANI HC22010

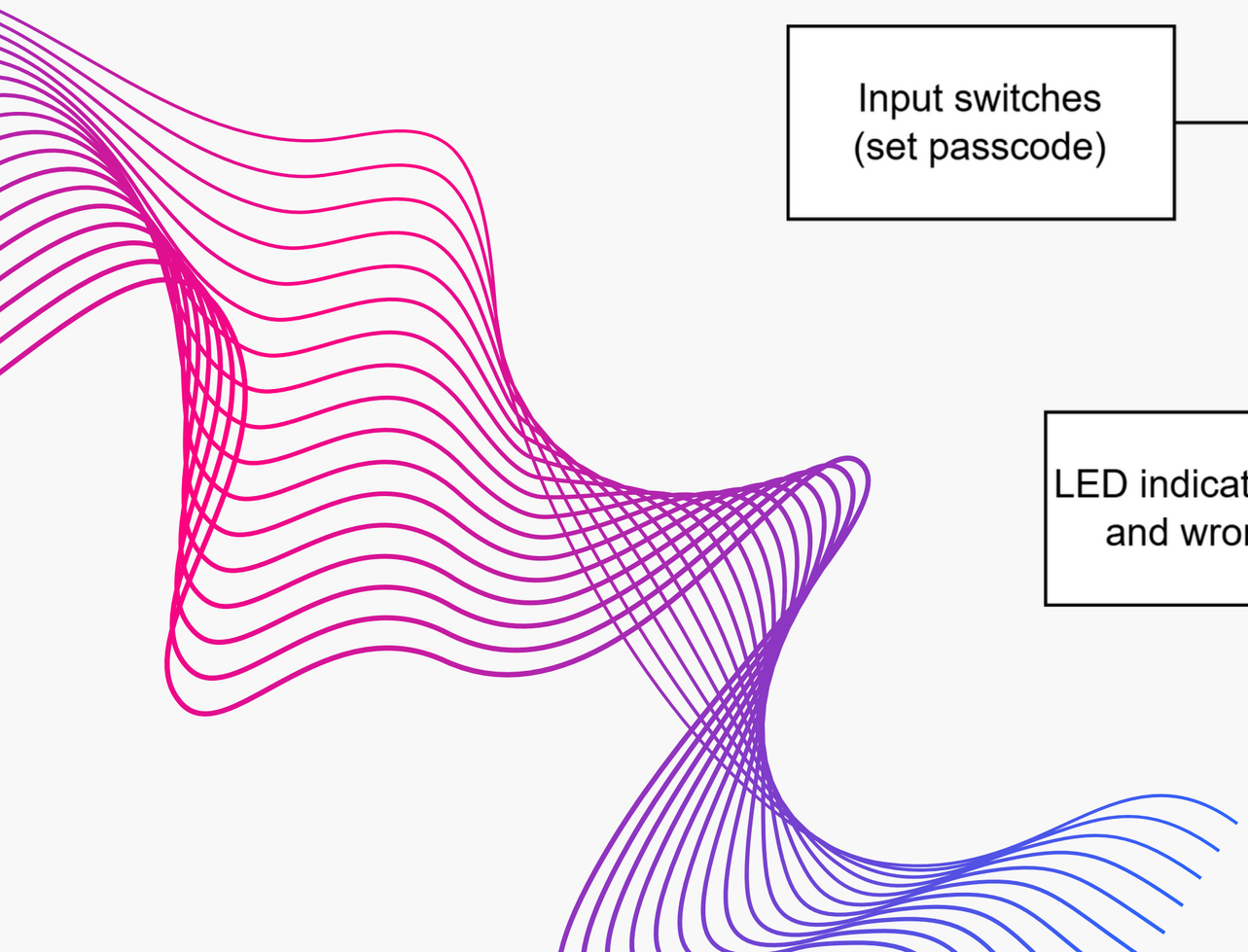
85/100

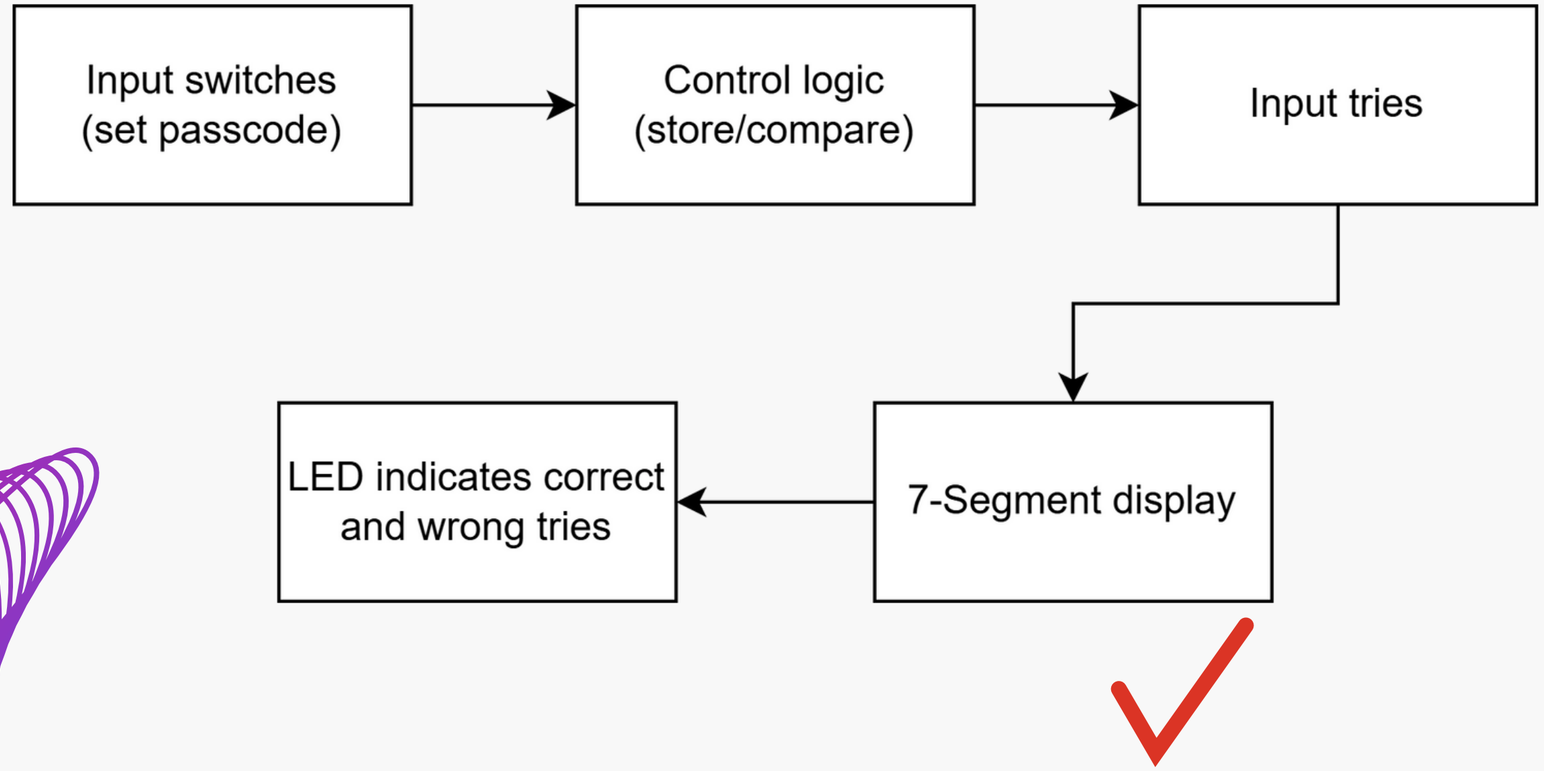
Objectives

- To design a digital lock using FSM that checks if the entered password matches the stored code.
- To write and test the design using Verilog HDL in Quartus Prime.
- To implement the design on an FPGA board using switches, LEDs, and 7-segment displays.
- To identify and solve basic hardware issues like button errors and clock instability.



System Overview





PROJECT RESULT YOUTUBE LINK

What to expect?

System Features

1. Password input via DIP switches
2. Feedback via LEDs or 7-segment
3. Lock/unlock status

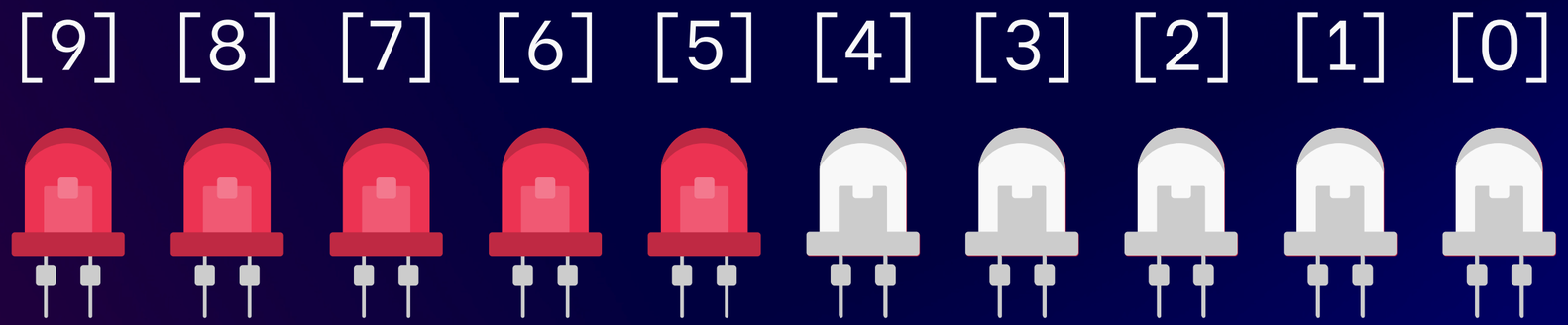
PROJECT RESULT YOUTUBE LINK

User Manual

1. Use SW[0] until SW[3] to input number
2. Use SW[4] to set it to password
3. Set SW[4] to 0 before you enter the password
4. After you enter new password, use SW[5] to check.

PROJECT RESULT YOUTUBE LINK

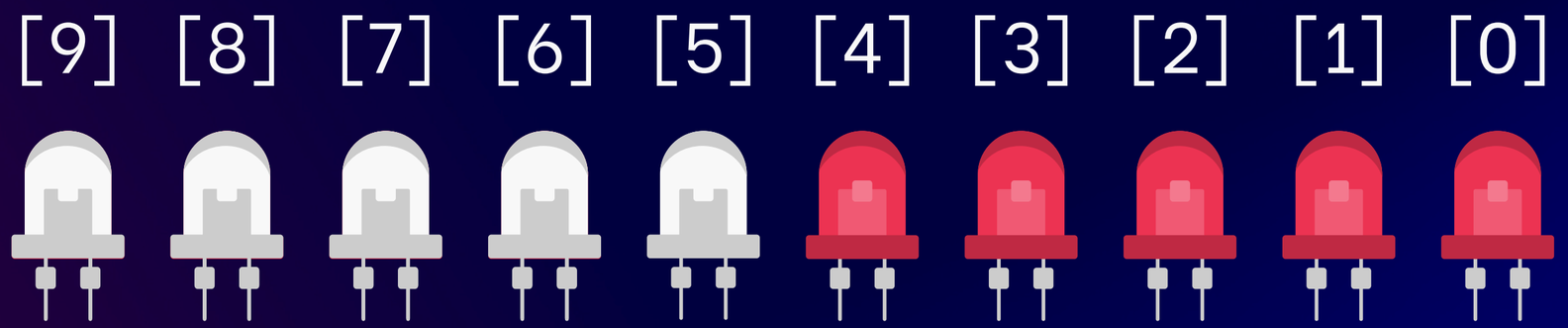
User Manual



PASSWORD IS FALSE

PROJECT RESULT YOUTUBE LINK

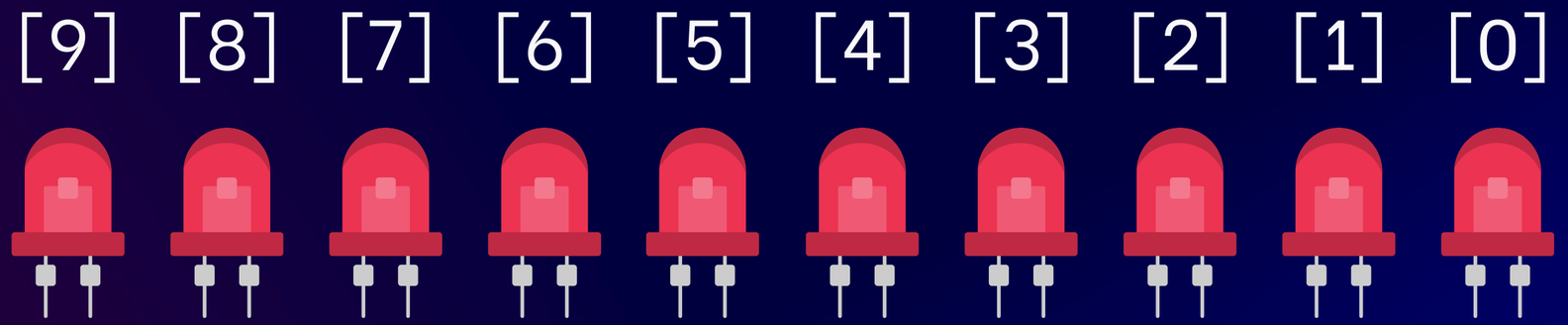
User Manual



PASSWORD IS TRUE

PROJECT RESULT YOUTUBE LINK

User Manual



TRIES = 0 | LOCKDOWN MODE

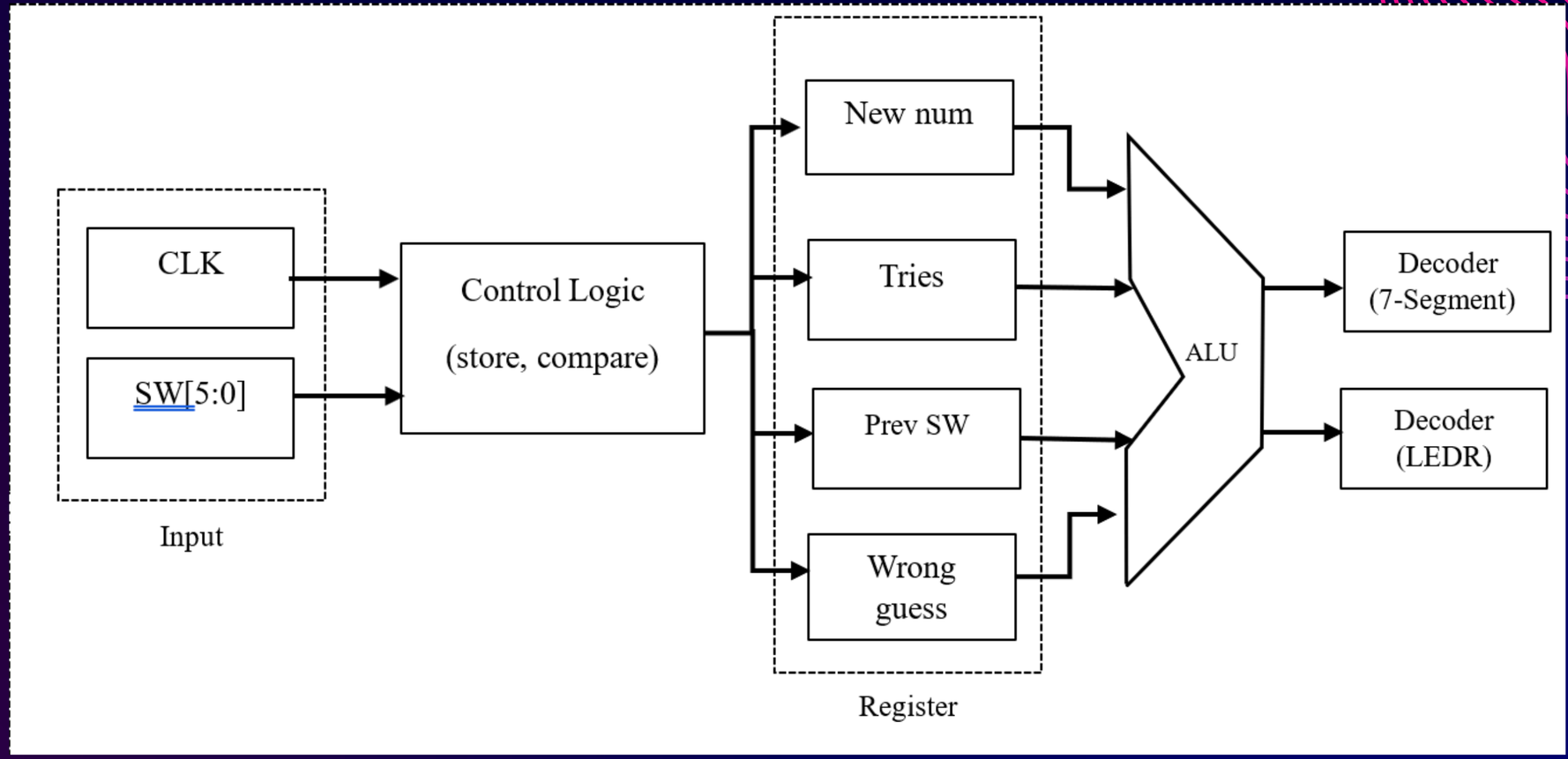
PROJECT RESULT YOUTUBE LINK

Restriction

SW[4] is sensitive to set the last inputted value as password.

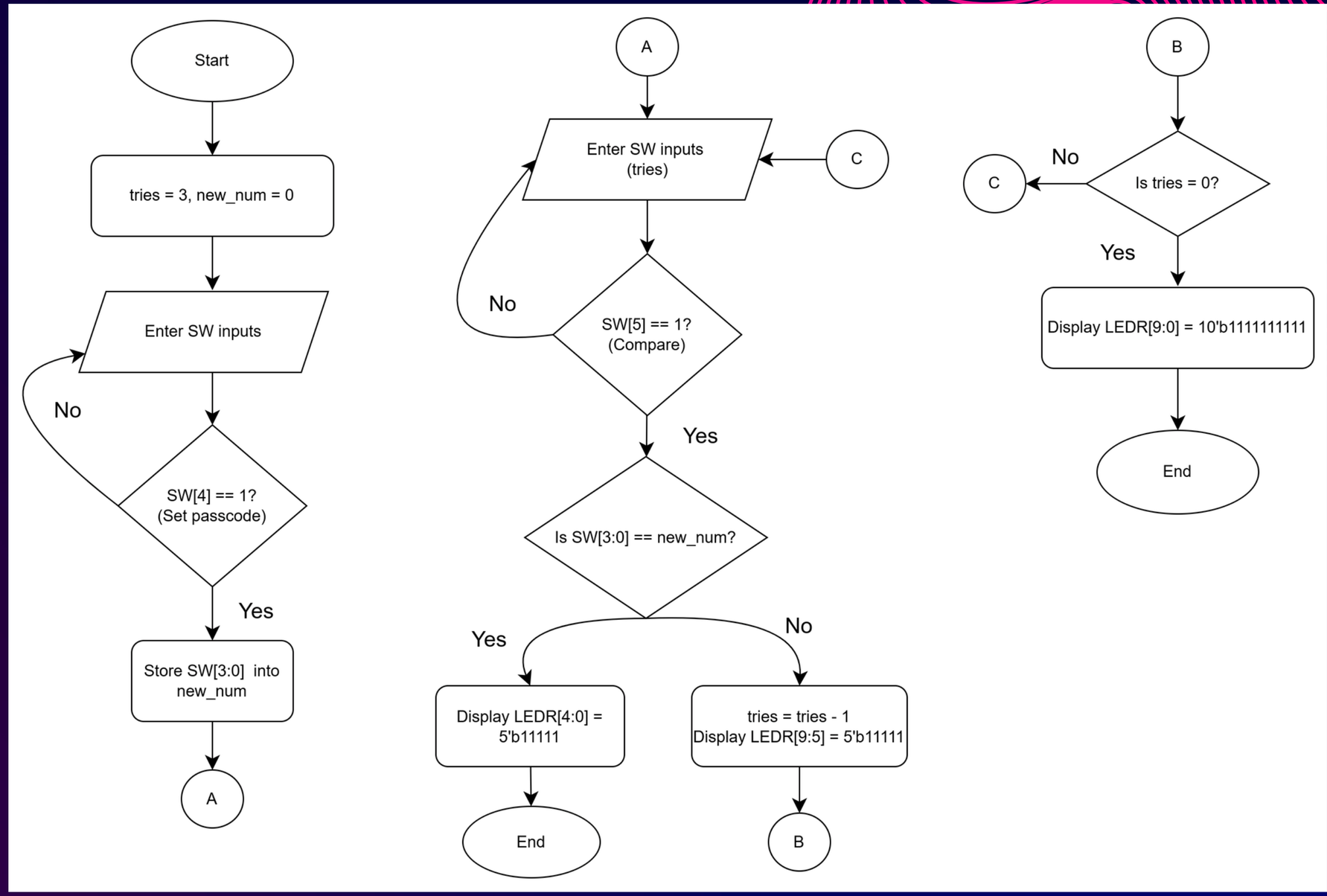
If we change the number input to zero before switch off the SW[4], the set pasword will be 00.

This is due to the structure of the coding.


CLK
SW[5:0]
Input
Control Logic
(store, compare)
New num
Tries
Prev SW
Wrong
guess
Register
ALU
Decoder
(7-Segment)
Decoder
(LEDR)

Datapath

Flowchart


Start
tries = 3, new_num = 0
Enter SW inputs
No
SW[4] == 1?
(Set passcode)
Yes
Store SW[3:0] into
new_num
A
A
Enter SW inputs
(tries)
C
No
SW[5] == 1?
(Compare)
Yes
Is SW[3:0] == new_num?
Yes
No
Display LEDR[4:0] =
5'b11111
End
tries = tries - 1
Display LEDR[9:5] = 5'b11111
B
B
No
C
Is tries = 0?
Yes
Display LEDR[9:0] = 10'b1111111111
End

Coding

```
module PassDigLock (
    input             clk,
    input       [5:0] SW,       // SW[3:0] = input number, SW4 = store, SW5 = compare
    output      [6:0] HEX0,     // Ones digit display
    output      [6:0] HEX1,     // Tens digit display
    output      [6:0] HEX5,     // Tries display
    output reg  [9:0] LEDR      // LEDs
);
    reg [3:0] ones;
    reg [3:0] tens;
    reg [3:0] new_num = 4'b0000;
    reg [1:0] tries   = 2'd3;    // 0 to 3 tries
    reg       prev_sw = 1'b0;
    reg       wrong_guess = 1'b0;

    always @(*) begin
        LEDR = 10'b0;             // Clear LEDs

        if (SW[4] == 1'b1) begin
            // Store mode
            LEDR[4] = 1'b1;
        end
        else if (SW[5] == 1'b1) begin
            // Compare mode
            if (SW[3:0] == new_num) begin
                // Correct
                LEDR[4:0] = 5'b11111;
            end
            else if (tries == 2'd0) begin
                // Game over - all tries used
                LEDR = 10'b1111111111;
            end
            else if (wrong_guess) begin
                // Wrong guess indication
                LEDR[9:5] = 5'b11111;
            end
        end

        // Binary to BCD conversion for display
        if (SW[3:0] < 4'd10) begin
            tens = 4'd0;
            ones = SW[3:0];
        end else begin
            tens = 4'd1;
            ones = SW[3:0] - 4'd10;
        end
    end
```

```
    // Handle store and tries on clock edge
    always @(posedge clk) begin
        if (SW[4] == 1'b1) begin
            // Store mode
            new_num <= SW[3:0];
            tries <= 2'd3;
        end
        else if (SW[5] == 1'b1 && SW[3:0] != new_num && prev_sw == 1'b0) begin
            // Decrement tries on incorrect guess
            wrong_guess <= 1'b1;
            if (tries > 2'd0)
                tries <= tries - 2'd1;
        end
        prev_sw <= SW[5];
    end

    SevenSegDec decode_tens  (.bin(tens),  .seg(HEX1));
    SevenSegDec decode_ones  (.bin(ones),  .seg(HEX0));
    SevenSegDec decode_tries (.bin(tries), .seg(HEX5));
endmodule
```

Coding

```
module SevenSegDec (
    input [3:0] bin,
    output reg [6:0] seg
);
    always @(*) begin
        case (bin)
            4'd0: seg = 7'b1000000;
            4'd1: seg = 7'b1111001;
            4'd2: seg = 7'b0100100;
            4'd3: seg = 7'b0110000;
            4'd4: seg = 7'b0011001;
            4'd5: seg = 7'b0010010;
            4'd6: seg = 7'b0000010;
            4'd7: seg = 7'b1111000;
            4'd8: seg = 7'b0000000;
            4'd9: seg = 7'b0010000;
            default: seg = 7'b1111111; // blank
        endcase
    end
endmodule
```

Testbench

This testbench is designed to simulate the functionality of a password-protected digital clock implemented on an FPGA. It verifies key aspects of the design, including proper handling of user input through a keypad or switches, and secure access control via a preset password

```
`timescale 1ns / 1ps

module PassDigLock_tb;

    // Inputs
    reg clk;
    reg [5:0] SW;

    // Outputs
    wire [6:0] HEX0, HEX1, HEX5;
    wire [9:0] LEDR;

    // Instantiate the DUT (Device Under Test)
    PassDigLock uut (
        .clk(clk),
        .SW(SW),
        .HEX0(HEX0),
        .HEX1(HEX1),
        .HEX5(HEX5),
        .LEDR(LEDR)
    );

    // Clock generation
    initial clk = 0;
    always #5 clk = ~clk;  // 10 ns clock period

    // Task to simulate button press
    task press_store(input [3:0] val);
        begin
            SW[3:0] = val;  // Input number
            SW[4] = 1;       // Store mode
            SW[5] = 0;
            #10;
            SW[4] = 0;
            #10;
        end
    endtask
```

Testbench

This testbench is designed to simulate the functionality of a password-protected digital clock implemented on an FPGA. It verifies key aspects of the design, including proper handling of user input through a keypad or switches, and secure access control via a preset password

```
    task press_compare(input [3:0] val);
        begin
            SW[3:0] = val;  // Input number
            SW[4] = 0;
            SW[5] = 1;      // Compare mode
            #10;
            SW[5] = 0;
            #10;
        end
    endtask

    initial begin
        // Initial state
        SW = 6'b0;

        // Wait for global reset
        #20;

        // Store the number 6
        $display("Storing number 6...");
        press_store(4'd6);

        // Try correct guess
        $display("Trying correct guess 6...");
        press_compare(4'd6);

        // Try incorrect guess 3 times
        $display("Trying incorrect guess 4...");
        press_compare(4'd12);

        $display("Trying incorrect guess 5...");
        press_compare(4'd5);

        $display("Trying incorrect guess 7...");
        press_compare(4'd7);

        // One more incorrect guess after 3 tries used up
        $display("Trying incorrect guess 8 (should be game over)...");
        press_compare(4'd8);

        #100;

        $stop;
    end

endmodule
```

Simulation Waveform

It simulates both valid and invalid password attempts, ensuring the system only grants access upon correct authentication. The purpose of this simulation is to validate the integration of digital clock logic with user authentication, ensuring both accuracy and reliability before hardware deployment.

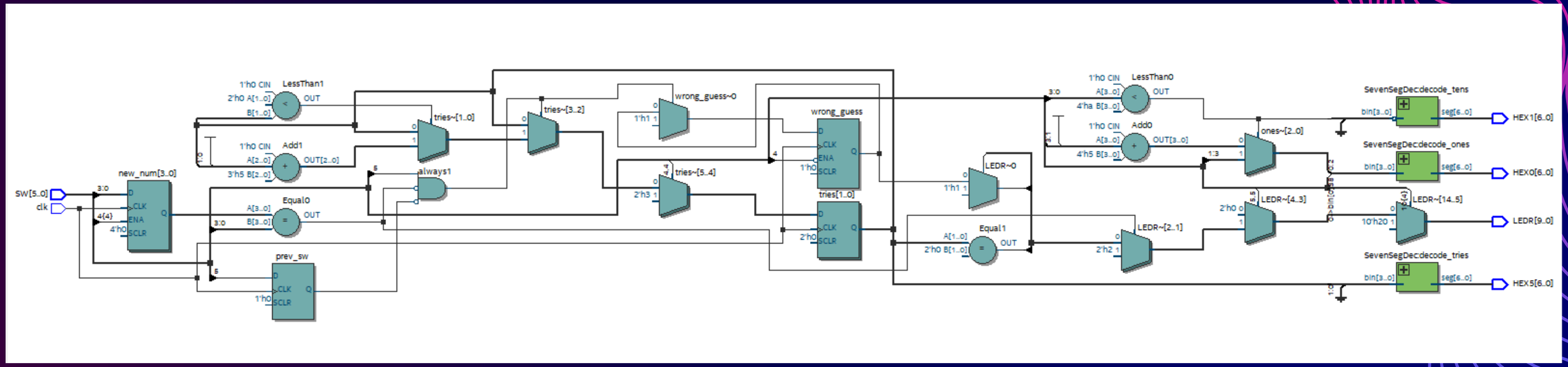
I/O Pin Assignments:

All Pins

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatior	I/O Standard	Reserved	urrent Streng	Slew Rate
HEX0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	2.5 V		12mA ...ault)	2 (default)
HEX0[5]	Output	PIN_D17	7	B7_N0	PIN_D17	2.5 V		12mA ...ault)	2 (default)
HEX0[4]	Output	PIN_E16	7	B7_N0	PIN_E16	2.5 V		12mA ...ault)	2 (default)
HEX0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	2.5 V		12mA ...ault)	2 (default)
HEX0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	2.5 V		12mA ...ault)	2 (default)
HEX0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	2.5 V		12mA ...ault)	2 (default)
HEX0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	2.5 V		12mA ...ault)	2 (default)
HEX1[6]	Output	PIN_B17	7	B7_N0	PIN_B17	2.5 V		12mA ...ault)	2 (default)
HEX1[5]	Output	PIN_A18	7	B7_N0	PIN_A18	2.5 V		12mA ...ault)	2 (default)
HEX1[4]	Output	PIN_A17	7	B7_N0	PIN_A17	2.5 V		12mA ...ault)	2 (default)
HEX1[3]	Output	PIN_B16	7	B7_N0	PIN_B16	2.5 V		12mA ...ault)	2 (default)
HEX1[2]	Output	PIN_E18	6	B6_N0	PIN_E18	2.5 V		12mA ...ault)	2 (default)
HEX1[1]	Output	PIN_D18	6	B6_N0	PIN_D18	2.5 V		12mA ...ault)	2 (default)
HEX1[0]	Output	PIN_C18	7	B7_N0	PIN_C18	2.5 V		12mA ...ault)	2 (default)
HEX5[6]	Output	PIN_N20	6	B6_N0	PIN_N20	2.5 V		12mA ...ault)	2 (default)
HEX5[5]	Output	PIN_N19	6	B6_N0	PIN_N19	2.5 V		12mA ...ault)	2 (default)
HEX5[4]	Output	PIN_M20	6	B6_N0	PIN_M20	2.5 V		12mA ...ault)	2 (default)
HEX5[3]	Output	PIN_N18	6	B6_N0	PIN_N18	2.5 V		12mA ...ault)	2 (default)
HEX5[2]	Output	PIN_L18	6	B6_N0	PIN_L18	2.5 V		12mA ...ault)	2 (default)
HEX5[1]	Output	PIN_K20	6	B6_N0	PIN_K20	2.5 V		12mA ...ault)	2 (default)
HEX5[0]	Output	PIN_J20	6	B6_N0	PIN_J20	2.5 V		12mA ...ault)	2 (default)
LEDR[9]	Output	PIN_B11	7	B7_N0	PIN_B11	2.5 V		12mA ...ault)	2 (default)
LEDR[8]	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		12mA ...ault)	2 (default)
LEDR[7]	Output	PIN_D14	7	B7_N0	PIN_D14	2.5 V		12mA ...ault)	2 (default)
LEDR[6]	Output	PIN_E14	7	B7_N0	PIN_E14	2.5 V		12mA ...ault)	2 (default)
LEDR[5]	Output	PIN_C13	7	B7_N0	PIN_C13	2.5 V		12mA ...ault)	2 (default)
LEDR[4]	Output	PIN_D13	7	B7_N0	PIN_D13	2.5 V		12mA ...ault)	2 (default)
LEDR[3]	Output	PIN_B10	7	B7_N0	PIN_B10	2.5 V		12mA ...ault)	2 (default)
LEDR[2]	Output	PIN_A10	7	B7_N0	PIN_A10	2.5 V		12mA ...ault)	2 (default)
LEDR[1]	Output	PIN_A9	7	B7_N0	PIN_A9	2.5 V		12mA ...ault)	2 (default)
LEDR[0]	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA ...ault)	2 (default)
SW[5]	Input	PIN_B12	7	B7_N0	PIN_B12	2.5 V		12mA ...ault)	
SW[4]	Input	PIN_A12	7	B7_N0	PIN_A12	2.5 V		12mA ...ault)	
SW[3]	Input	PIN_C12	7	B7_N0	PIN_C12	2.5 V		12mA ...ault)	
SW[2]	Input	PIN_D12	7	B7_N0	PIN_D12	2.5 V		12mA ...ault)	
SW[1]	Input	PIN_C11	7	B7_N0	PIN_C11	2.5 V		12mA ...ault)	
SW[0]	Input	PIN_C10	7	B7_N0	PIN_C10	2.5 V		12mA ...ault)	
clk	Input	PIN_P11	3	B3_N0	PIN_P11	2.5 V		12mA ...ault)	
<<new node>>									

RTL Viewer


SW[5..0]
clk
new_num[3..0]
LessThan1
Add1
Equal0
prev_sw
tries~[1..0]
tries~[3..2]
always1
wrong_guess~0
tries~[5..4]
wrong_guess
tries[1..0]
LEDR~0
Equal1
LessThan0
Add0
ones~[2..0]
LEDR~[2..1]
LEDR~[4..3]
LEDR~[14..5]
SevenSegDec:decode_tens
SevenSegDec:decode_ones
SevenSegDec:decode_tries
HEX1[6..0]
HEX0[6..0]
LEDR[9..0]
HEX5[6..0]

Performance Evaluation

Item	Observation
FPGA logic utilization	< 1 %
Total combinational functions	23
Total register	8
Maximum clock frequency	629.72 MHz
Critical path delay	-0.588 ns

Compilation Report

Flow Summary

<<Filter>>

Flow Status	Successful - Sun Jun 22 01:23:37 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	NumberHandler
Top-level Entity Name	SwitchToSevenSeg
Family	MAX 10
Device	10M50DAF484C6GES
Timing Models	Preliminary
Total logic elements	27 / 49,760 (< 1 %)
Total registers	8
Total pins	38 / 360 (11 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Performance vs Complexity – Trade-Off Analysis:

Performance Highlights

Very Low Resource Utilisation:

- Implication: The design is extremely lightweight and should operate efficiently with very low power and minimal propagation delay, offering high speed in practical terms.

No use of multipliers, memory blocks, or PLLs:

- This contributes to low latency, and the system likely performs only basic logic functions (e.g., simple control, encoding, or switching).

Complexity Trade-offs

The design doesn’t use any advanced hardware features such as:

- Embedded multipliers (0/288)
- Memory bits (0/1,677,312)
- PLLs (0/4)
- ADC blocks (0/2)
-

Implication:

- Not scalable for signal processing, real-time data acquisition, or complex algorithms without redesign.

Complexity Trade-offs

Pin Usage (11%):

- While only 27 logic elements are used, 38 pins are already occupied. This suggests the design might involve multiple inputs/outputs (e.g., switches, displays).
- Could become a bottleneck in future expansions without pin-sharing or I/O multiplexing.

Conclusion & Reflection

In this project, we successfully developed a working system that performs the intended basic functionality. However, several hardware limitations were encountered—most notably, the malfunctioning of the push button and the instability of the clock cycle. These issues hindered our ability to implement the design in a proper FSM (Finite State Machine) style. Additionally, Quartus Prime's lack of support for the enum construct further restricted our ability to structure the code according to standard FSM practices.

Despite achieving a functional output, the system remains unstable. With more time and thorough debugging, this instability could potentially be resolved. However, due to time constraints, we were unable to carry out further refinement. Overall, while the project can be considered a partial success, future work is needed to improve system reliability and fully realise the intended FSM-based design.

THANK YOU

DE10-LITE FOR INTEL FPGA UNIVERSITY PROGRAM

BHE3233 DIGITAL SYSTEM DESIGN

JUNI INSYIRAH BINTI JAMRI HC22032
HANUM ANISAH BINTI AHMAD JANI HC22010