Dr. B.R. Ambedkar National Institute of Technology Jalandhar

Jalandhar(144008), Punjab

Password Lock FSM

Submitted to:

Dr. Tarun Chaudhary

Assistant Professor, ECE

NITJ, Jalandhar

Submitted by:

Group 6

Aditya (25904138)

Mayank Verma (25904145)

Sarthak Singhal (25204131)

Karthikeyan (25904141)

Pradhum Kumar (25904147)

Abstract

This project demonstrates how FSMs can create robust security applications with features like:

- 4-digit password entry system
- Failed attempt tracking (3 attempts before lockout)
- Automatic timeout reset
- Timed unlock duration
- Password change capability

Helps understand how to architect a state machine that handles complex sequential logic and security protocols.

Password Lock FSM

- Password Lock FSM is a security mechanism that grants access to anything like device or may be doors/ lockers cars etc. only when the correct password is entered.
- Widely used in digital security systems like electronic safes, electronic door locks, and embedded devices.
- It ensures protection against unauthorized access.
- At the heart of these systems lies a simple yet powerful concept:

  Finite State Machine (FSM).

Finite State Machine

A Finite State Machine is a sequential logic model used to design systems that operate through a series of well-defined states. The system moves from one state to another based on input signals and predefined rules.

- FSM consists of a finite number of states.
- The system can be in only one state at a time.
- State transitions occur when specific inputs or conditions are met.
- Used to design control-oriented digital systems.

How FSM Works

01 Define states and transitions

02 Create a state transition table or diagram

03 Initialize the FSM

04 Process input event

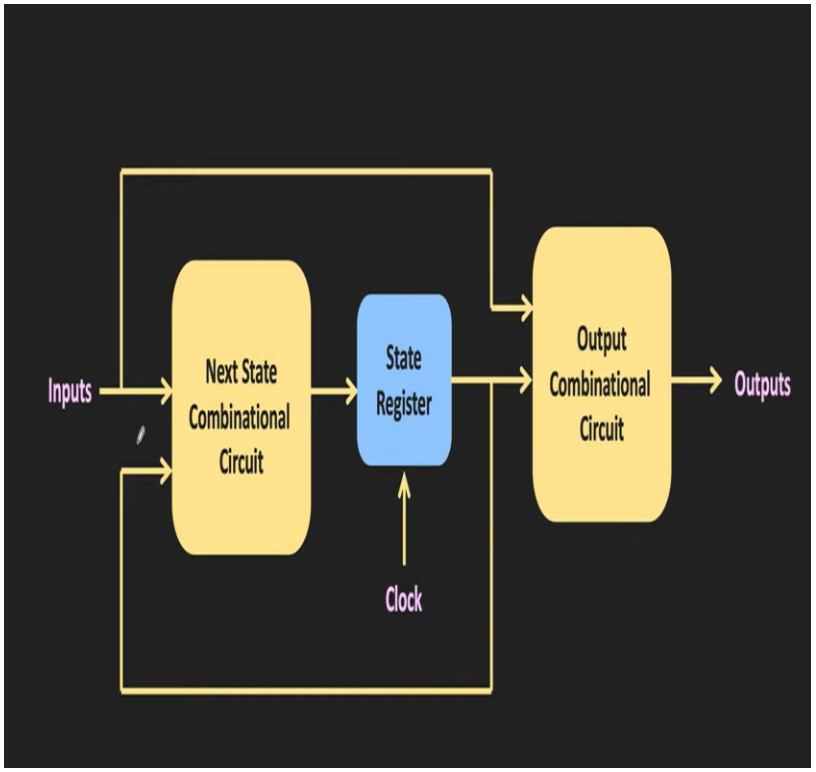
05 Check the current state and input event

06 Look up and execute transition

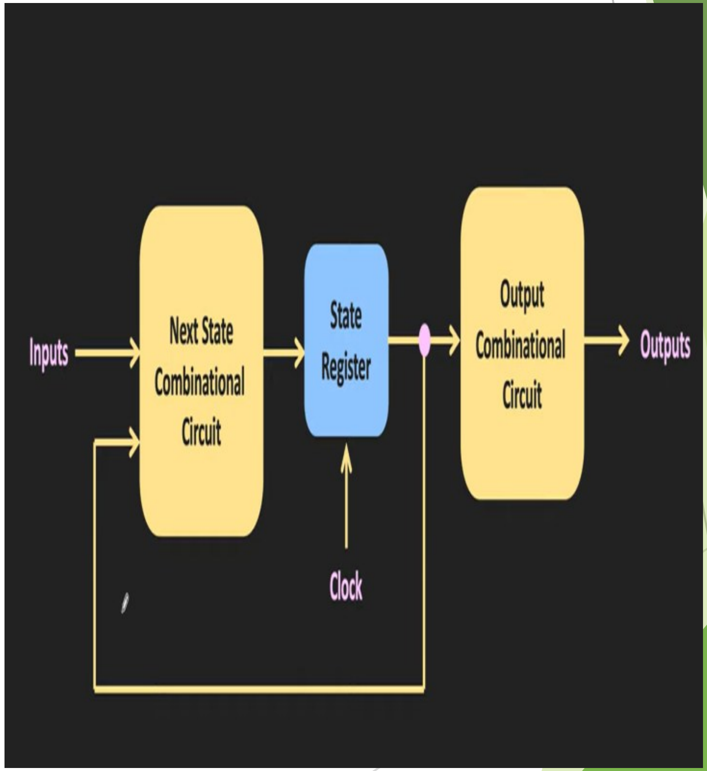
07 Update current state

Types of FSM

Mealy Machine


Inputs
Next State Combinational Circuit
State Register
Output Combinational Circuit
Outputs
Clock

Moore Machine


Inputs
Next State Combinational Circuit
State Register
Output Combinational Circuit
Outputs
Clock

Why use FSM for Password Lock

- FSM allows stepwise verification of input digits
- Ensures deterministic behavior
- Reliable for embedded security applications
- Easy to Handle Wrong Inputs.
- Supports Reset and Retry Logic
- Simplifies Debugging and Simulation

System Architecture

1. Input Interface

- Digit Input (4-bit)
- Enter Button
- Change Password Button

2. Edge Detection Module

- Detect a clean **rising pulse** from the enter button.

System Architecture

3. FSM Controller

- This is the **core of the system.**

States

- **IDLE**: Initial locked state, waiting for first digit
- **DIGIT1-DIGIT4**: Sequential digit entry states
- **CHECK_PASSWORD**: Validates entered password against stored password
- **UNLOCKED**: Door unlocked for 10 seconds
- **WRONG_PASSWORD**: Increments failed attempt counter
- **LOCKED_OUT**: System locked after 3 failed attempts
- **TIMEOUT**: 30-second penalty before returning to IDLE
- **CHANGE_PASSWORD**: Special mode for updating password

Responsibilities:

- Controls system behaviour.
- Moves sequentially from digit 1 → digit 4.
- Triggers password checking.
- Decides lock/unlock/error states.

System Architecture

4. Password Storage & Handling

a) Stored Password

b) Entered Password

5. Password Comparison Block

A parallel comparator checks the input and outputs

6. Attempt Counter Logic

Counts wrong attempts.

Max allowed attempts = 3.

7. Output Control Unit

Controls external indicators

Working

Top-level: An FSM implementing a 4-digit digital lock.
Inputs: clk, reset, digit_in (0–9), enter_btn, change_pwd_btn
Outputs: led (status), seg/an (7-seg), locked (1 = locked, 0= unlocked).

States: IDLE → DIGIT1 → DIGIT2 → DIGIT3 → DIGIT4 → CHECK_PASSWORD → UNLOCKED (or WRONG_PASSWORD / LOCKED_OUT).
DIGIT4 automatically proceeds to CHECK_PASSWORD.

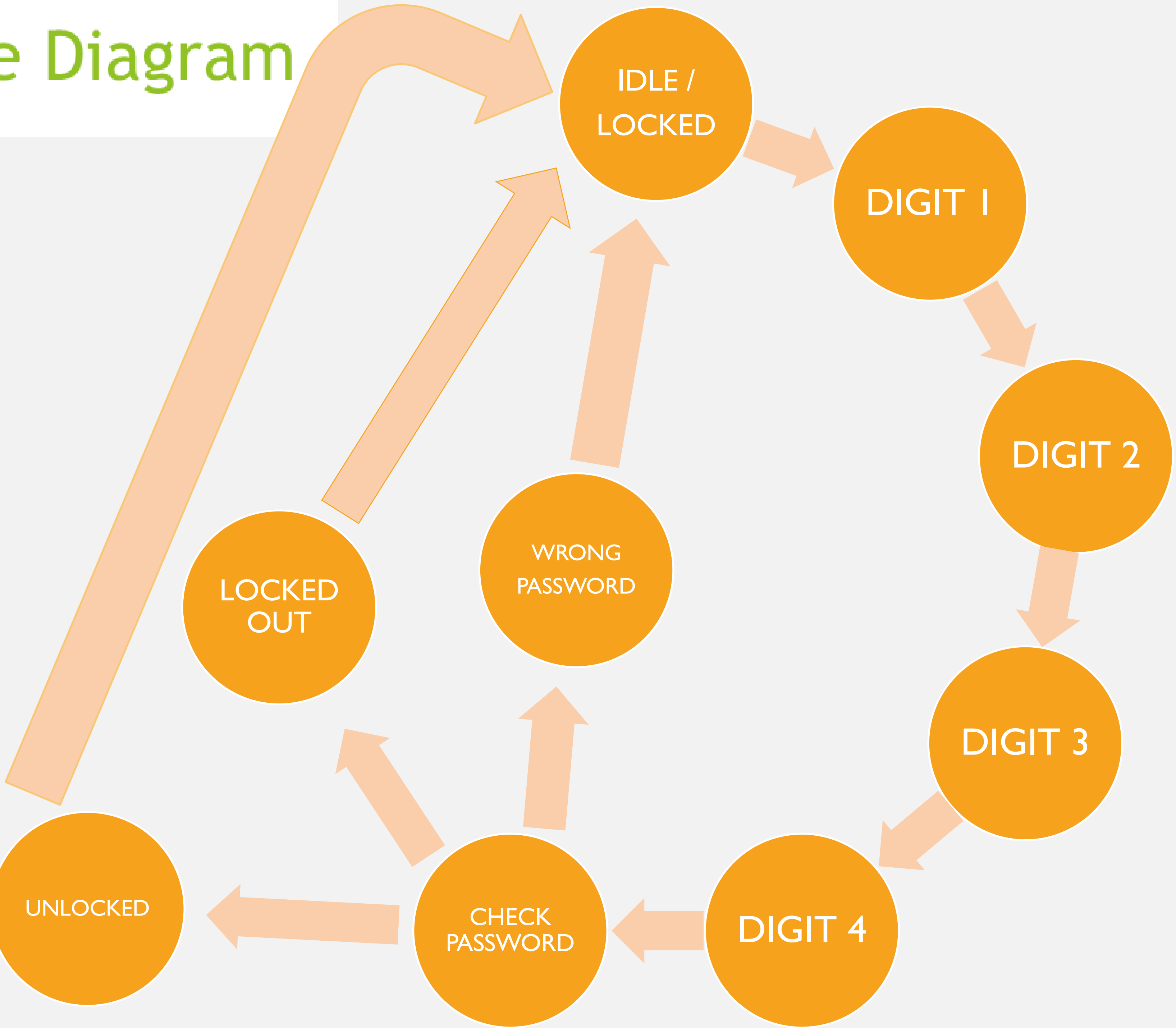
Password storage: password[0:3] initialized to 1,2,3,4. Entered digits saved into entered_digits[0:3] on rising edges of enter_btn when in the corresponding DIGIT state.

Button edge detect: two-flip flop sampling of enter_btn creates enter_btn_rising (rising edge detection).

Password check: combinational compare of all four entered digits to stored password.

Attempt counting: attempt_count is intended to track failed tries and move to LOCKED_OUT after 3 attempts.

State Diagram



Security Features

- **Attempt Limiting:** After 3 incorrect passwords, the system enters LOCKED_OUT state
- **Timeout Protection:** Automatic reset if user takes too long between digits
- **Timed Unlock:** Lock automatically re-engages after 10 seconds
- **Password Management:** Secure password change mode with verification

Code

```
module digital_lock (
    input wire clk,
    input wire reset,
    input wire [3:0] digit_in,        // Input digit 0-9
    input wire enter_btn,              // Button to
submit digit
    input wire change_pwd_btn,         // Button to
change password
    output reg [15:0] led,            // Status LEDs
    output reg [6:0] seg,             // 7-segment
display
    output reg [7:0] an,              // 7-segment
anodes
    output reg locked                 // Lock status
(1=locked, 0=unlocked)
);

    // STATE DEFINITIONS
 localparam [3:0] IDLE          = 4'd0;
 localparam [3:0] DIGIT1         = 4'd1;
 localparam [3:0] DIGIT2         = 4'd2;
 localparam [3:0] DIGIT3         = 4'd3;
 localparam [3:0] DIGIT4         = 4'd4;
 localparam [3:0] CHECK_PASSWORD = 4'd5;
 localparam [3:0] UNLOCKED       = 4'd6;
 localparam [3:0] WRONG_PASSWORD = 4'd7;
 localparam [3:0] LOCKED_OUT     = 4'd8;

 reg [3:0] current_state, next_state;



// PASSWORD STORAGE
 reg [3:0] password [0:3];
 reg [3:0] entered_digits [0:3];
 reg [1:0] attempt_count;
```

```
initial begin
    password[0] = 4'd1;
    password[1] = 4'd2;
    password[2] = 4'd3;
    password[3] = 4'd4;
    attempt_count = 2'd0;
end

// SIMPLE BUTTON EDGE DETECTION
reg enter_btn_r1, enter_btn_r2;
wire enter_btn_rising;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        enter_btn_r1 <= 1'b0;
        enter_btn_r2 <= 1'b0;
    end else begin
        enter_btn_r1 <= enter_btn;
        enter_btn_r2 <= enter_btn_r1;
    end
end

 assign enter_btn_rising = enter_btn_r1 &&
!enter_btn_r2;
// PASSWORD VERIFICATION
 wire password_correct;
 assign password_correct = (entered_digits[0] ==
password[0]) &&
                          (entered_digits[1] ==
password[1]) &&
                          (entered_digits[2] ==
password[2]) &&
                          (entered_digits[3] ==
password[3]);

 // STATE REGISTER
 always @(posedge clk or posedge reset) begin
    if (reset)
        current_state <= IDLE;
    else
        current_state <= next_state;
 end
```

Code

```
// NEXT STATE LOGIC - SIMPLIFIED (NO TIMEOUT)
 always @(*) begin
    next_state = current_state;

    case (current_state)
        IDLE: begin
            if (enter_btn_rising)
                next_state = DIGIT1;
        end

        DIGIT1: begin
            if (enter_btn_rising)
                next_state = DIGIT2;
        end

        DIGIT2: begin
            if (enter_btn_rising)
                next_state = DIGIT3;
        end

        DIGIT3: begin
            if (enter_btn_rising)
                next_state = DIGIT4;
        end

        DIGIT4: begin
            // Automatically go to CHECK_PASSWORD after
storing 4th digit
            // No need to wait for another button press
            next_state = CHECK_PASSWORD;
        end

CHECK_PASSWORD: begin
            if (password_correct)
                next_state = UNLOCKED;
            else if (attempt_count >= 2'd2)
                next_state = LOCKED_OUT;
            else
                next_state = WRONG_PASSWORD;
        end

UNLOCKED: begin
            // Stay unlocked (manual reset to test)
            next_state = UNLOCKED;
        end

        WRONG_PASSWORD: begin
            // Manual return to IDLE for now
            if (enter_btn_rising)
                next_state = IDLE;
        end

        LOCKED_OUT: begin
            // Manual return to IDLE for now
            if (enter_btn_rising)
                next_state = IDLE;
        end

        default: next_state = IDLE;
    endcase
 end

 // DIGIT STORAGE - SIMPLIFIED
 always @(posedge clk or posedge reset) begin
    if (reset) begin
        entered_digits[0] <= 4'd0;
        entered_digits[1] <= 4'd0;
        entered_digits[2] <= 4'd0;
        entered_digits[3] <= 4'd0;
    end else begin
        if (current_state == IDLE && enter_btn_rising) begin
            // When transitioning from IDLE to DIGIT1, store first digit
            entered_digits[0] <= digit_in;
        end
        else if (current_state == DIGIT1 && enter_btn_rising) begin
            // When transitioning from DIGIT1 to DIGIT2, store second digit
            entered_digits[1] <= digit_in;
        end
        else if (current_state == DIGIT2 && enter_btn_rising) begin
            // When transitioning from DIGIT2 to DIGIT3, store third digit
            entered_digits[2] <= digit_in;
        end
        else if (current_state == DIGIT3 && enter_btn_rising) begin
            // When transitioning from DIGIT3 to DIGIT4, store fourth digit
            entered_digits[3] <= digit_in;
        end
        else if (current_state == UNLOCKED || current_state ==
WRONG_PASSWORD) begin
            // Clear entered digits after checking
            entered_digits[0] <= 4'd0;
            entered_digits[1] <= 4'd0;
            entered_digits[2] <= 4'd0;
            entered_digits[3] <= 4'd0;
        end
    end
 end
```

Code

```
// ATTEMPT COUNTER
 always @(posedge clk or posedge reset) begin
     if (reset) begin
         attempt_count <= 2'd0;
     end else begin
         if (current_state == UNLOCKED)
             attempt_count <= 2'd0;
         else if (current_state == WRONG_PASSWORD)
             attempt_count <= attempt_count + 1;
     end
 end

 // OUTPUT LOGIC
 always @(posedge clk or posedge reset) begin
     if (reset) begin
         locked <= 1'b1;
         led <= 16'h0000;
     end else begin
         case (current_state)
             IDLE: begin
                 locked <= 1'b1;
                 led <= 16'h0001;  // State 0
             end

             DIGIT1: begin
                 locked <= 1'b1;
                 led <= 16'h0002;  // State 1
             end

             DIGIT2: begin
                 locked <= 1'b1;
                 led <= 16'h0004;  // State 2
             end
DIGIT3: begin
                 locked <= 1'b1;
                 led <= 16'h0008;  // State 3
             end
DIGIT4: begin
                 locked <= 1'b1;
                 led <= 16'h0010;  // State 4
             end

             CHECK_PASSWORD: begin
                 locked <= 1'b1;
                 led <= 16'h0020;  // State 5
             end
UNLOCKED: begin
                 locked <= 1'b0;
                 led <= 16'hFFFF;  // All on = unlocked!
             end

             WRONG_PASSWORD: begin
                 locked <= 1'b1;
                 led <= 16'h0040;  // State 7
             end

             LOCKED_OUT: begin
                 locked <= 1'b1;
                 led <= 16'hAAAA;  // Alternating
             end

             default: begin
                 locked <= 1'b1;
                 led <= 16'h0000;
             end
         endcase
     end
 end
```

7-SEGMENT DISPLAY

```
reg [15:0] refresh_counter = 0;
always @(posedge clk) begin

refresh_counter <= refresh_counter + 1;
end

wire [1:0] mux_select =
refresh_counter[15:14];
function [6:0] encode7;
    input [3:0] num;
    case(num)
        4'd0: encode7 = 7'b1000000;
        4'd1: encode7 = 7'b1111001;
        4'd2: encode7 = 7'b0100100;
        4'd3: encode7 = 7'b0110000;
        4'd4: encode7 = 7'b0011001;
        4'd5: encode7 = 7'b0010010;
        4'd6: encode7 = 7'b0000010;
        4'd7: encode7 = 7'b1111000;
        4'd8: encode7 = 7'b0000000;
        4'd9: encode7 = 7'b0010000;
        default: encode7 = 7'b1111111;
    endcase
endfunction
```

```
always @(*) begin
    case(mux_select)

        2'b00: begin
            an = 8'b11111110;     // Digit 0 ON
            seg = encode7(entered_digits[0]);
        end

        2'b01: begin
            an = 8'b11111101;     // Digit 1 ON
            seg = encode7(entered_digits[1]);
        end

        2'b10: begin
            an = 8'b11111011;     // Digit 2 ON
            seg = encode7(entered_digits[2]);
        end

        2'b11: begin
            an = 8'b11110111;     // Digit 3 ON
            seg = encode7(entered_digits[3]);
        end
    endcase
end
endmodule
```

Testbench

```
module digital_lock_tb;

    reg clk;
    reg reset;
    reg [3:0] digit_in;
    reg enter_btn;
    reg change_pwd_btn;
    wire [15:0] led;
    wire [6:0] seg;
    wire [7:0] an;
    wire locked;

     // Instantiate digital lock
      digital_lock uut (
          .clk(clk),
          .reset(reset),
          .digit_in(digit_in),
          .enter_btn(enter_btn),

.change_pwd_btn(change_pwd_btn),
          .led(led),
          .seg(seg),
          .an(an),
          .locked(locked)
      );

      // Clock - 100MHz
      initial begin
          clk = 0;
          forever #5 clk = ~clk;
      end
```

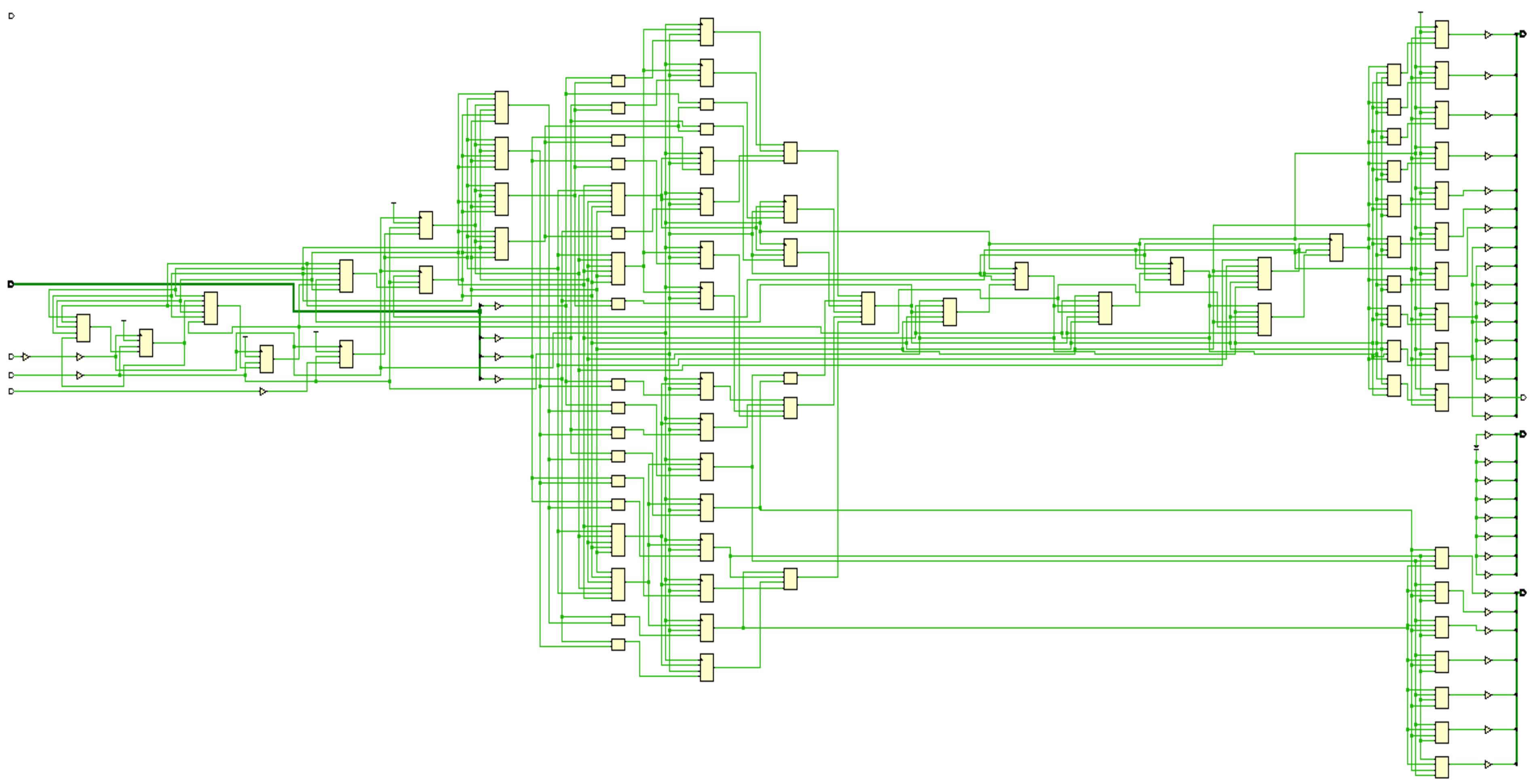
```
// Test
    initial begin

$display("\n===================================
=======");
        $display("  DIGITAL LOCK DEBUG
SIMULATION");

$display("===================================
=====");
        $display("Password: 1-2-3-4");
        $display("NO TIMEOUT - Pure state
machine test\n");

        // Init
        reset = 1;
        digit_in = 4'd0;
        enter_btn = 0;
        change_pwd_btn = 0;
        #100;

        reset = 0;
        $display("%0t: Reset released", $time);
        #100;

        // TEST: Enter password digit by digit
        $display("\n--- Entering Digit 1 ---");
        digit_in = 4'd1;
        #50;
        $display("%0t: digit_in = %0d", $time,
digit_in);
        enter_btn = 1;
        #20;
        $display("%0t: enter_btn = 1", $time);
        #100;
        enter_btn = 0;
        $display("%0t: enter_btn = 0", $time);
        #200;
        $display("%0t: State=%0d, LED=%h,
entered[0]=%0d",
               $time, uut.current_state, led,
uut.entered_digits[0]);

        $display("\n--- Entering Digit 2 ---");
        digit_in = 4'd2;
```

Testbench

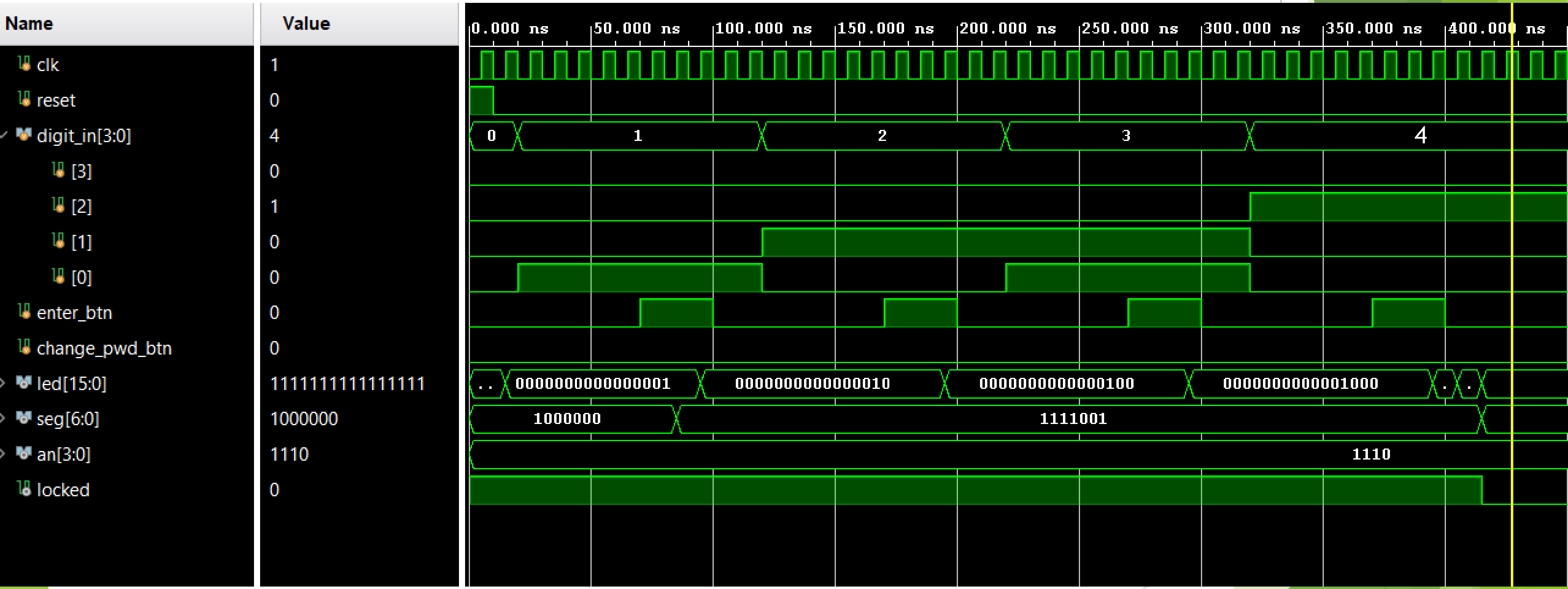
```
#50;
        $display("%0t: digit_in = %0d", $time,
digit_in);
        enter_btn = 1;
        #20;
        $display("%0t: enter_btn = 1", $time);
        #100;
        enter_btn = 0;
        $display("%0t: enter_btn = 0", $time);
        #200;
        $display("%0t: State=%0d, LED=%h,
entered[1]=%0d",
                 $time, uut.current_state, led,
uut.entered_digits[1]);

        $display("\n--- Entering Digit 3 ---");
        digit_in = 4'd3;
        #50;
        $display("%0t: digit_in = %0d", $time,
digit_in);
        enter_btn = 1;
        #20;
        $display("%0t: enter_btn = 1", $time);
        #100;
        enter_btn = 0;
        $display("%0t: enter_btn = 0", $time);
        #200;
        $display("%0t: State=%0d, LED=%h,
entered[2]=%0d",
                 $time, uut.current_state, led,
uut.entered_digits[2]);

        $display("\n--- Entering Digit 4 ---");
        digit_in = 4'd4;
        #50;
        $display("%0t: digit_in = %0d", $time,
digit_in);
        enter_btn = 1;
        #20;
        $display("%0t: enter_btn = 1", $time);
        #100;
        enter_btn = 0;
        $display("%0t: enter_btn = 0", $time);
        #200;
        $display("%0t: State=%0d, LED=%h,
entered[3]=%0d",
                 $time, uut.current_state, led,
uut.entered_digits[3]);
```

```
#500;

$display("\n========================================
==");
        $display("FINAL RESULTS:");
        $display("  State = %0d (should be
6=UNLOCKED)", uut.current_state);
        $display("  Locked = %b (should be 0)", locked);
        $display("  LED = %h (should be FFFF)", led);
        $display("  Password = %0d-%0d-%0d-%0d",
                 uut.password[0], uut.password[1],
                 uut.password[2], uut.password[3]);
        $display("  Entered  = %0d-%0d-%0d-%0d",
                 uut.entered_digits[0],
uut.entered_digits[1],
                 uut.entered_digits[2],
uut.entered_digits[3]);

        if (locked == 0) begin
            $display("\n*** SUCCESS! Lock is UNLOCKED!
***");
        end else begin
            $display("\n*** FAILED! Lock is still locked
***");
            $display("Debug info:");
            $display("  enter_btn_r1 = %b",
uut.enter_btn_r1);
            $display("  enter_btn_r2 = %b",
uut.enter_btn_r2);
            $display("  enter_btn_rising = %b",
uut.enter_btn_rising);
        end

$display("========================================
=\n");

        #10000;
        $finish;
    end

    // Monitor every state change
    always @(uut.current_state) begin
        $display("%0t: >>> STATE CHANGED to %0d <<<",
$time, uut.current_state);
    end

    // Monitor edge detection
    always @(posedge clk) begin
        if (uut.enter_btn_rising)
            $display("%0t: !!! BUTTON EDGE DETECTED
!!!", $time);
    end
endmodule
```

Schematic



Simulation Results



Name	Value
clk	1
reset	0
digit_in[3:0]	4
[3]	0
[2]	1
[1]	0
[0]	0
enter_btn	0
change_pwd_btn	0
led[15:0]	1111111111111111
seg[6:0]	1000000
an[3:0]	1110
locked	0

Real Life Applications

- This FSM design pattern is used in:
- **Electronic Door Locks**: Hotel rooms, office buildings
- **Safe Systems**: Bank vaults, home safes
- **Vehicle Immobilizers**: Keyless entry systems
- **ATM Machines**: PIN verification systems
- **Access Control**: Server rooms, data centres

Thank You