# DIGITAL SYSTEM DESIGN

# COURSE PROJECT REPORT

*Faculty Mentor :*
Prof.VINOD G

# DIGITAL LOCK

Bachelor of Technology
in
Electronics and Communication Engineering

SUBMITTED BY

**PADMAPRIYA J(NSS22EC090)**

**SUHAIR K(NSS22EC118)**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**
**NSS COLLEGE OF ENGINEERING,PALAKKAD**
**KERALA**

# Contents

# 1. INDRODUCTION

A Digital Lock with Pattern Detection is a secure access control system that verifies a specific binary sequence ("1101") before granting access. This project implements a non-overlapping Moore machine using Verilog HDL to create a reliable state machine that processes serial input bits and triggers an unlock signal only when the exact pattern is detected. The system consists of six distinct states (IDLE, S1, S11, S110, S1101, and UNLOCK) with clear transition logic, ensuring accurate pattern recognition while ignoring incorrect sequences. The design features asynchronous reset capability and processes one input bit per clock cycle, maintaining synchronization with the system clock. Implemented and simulated using Xilinx Vivado, the digital lock demonstrates correct functionality through comprehensive testbench verification, proving its suitability for FPGA-based security applications such as electronic door locks, safe mechanisms, and authentication systems. Keywords: Digital Lock, Moore Machine, Pattern Detection, Verilog, FSM, Security System.

# 2. STAE DIAGRAM

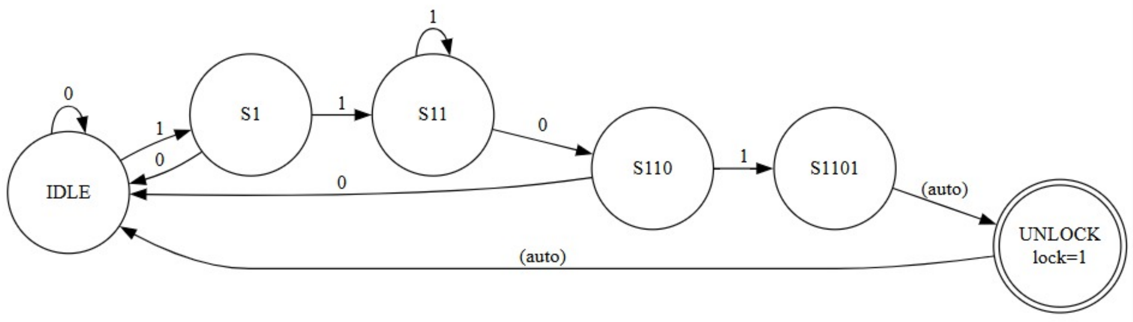


Figure 2.1: State diagram of Digital lock

# 3. CODE

## 3.1 main.code

The following Verilog code implements a *vending machine using verilog HDL*

```verilog
module digital_lock (
   input wire clk,         // Clock signal
   input wire reset,       // Asynchronous reset
   input wire in_bit,      // Serial input bit
   output reg lock         // Lock state
);

   // State encoding
   localparam IDLE  = 3'b000;
   localparam S1    = 3'b001;
   localparam S11   = 3'b010;
   localparam S110  = 3'b011;
   localparam S1101 = 3'b100;
   localparam UNLOCK = 3'b101;

   reg [2:0] current_state, next_state;

   // State transition logic
   always @(posedge clk or posedge reset) begin
       if (reset)
           current_state <= IDLE;
       else
           current_state <= next_state;
   end

   // Next state logic and output control
   always @(*) begin
       // Default values
       next_state = current_state;
       lock = 0;

       case (current_state)
           IDLE: begin
               if (in_bit)
                   next_state = S1;
               else
                   next_state = IDLE;
           end

           S1: begin
               if (in_bit)
                   next_state = S11;
```

```
                else
                    next_state = IDLE;
            end

            S11: begin
                if (!in_bit)
                    next_state = S110;
                else
                    next_state = S11;
            end

            S110: begin
                if (in_bit)
                    next_state = S1101;
                else
                    next_state = IDLE;
            end

            S1101: begin
                next_state = UNLOCK;
            end

            UNLOCK: begin
                lock = 1;  // Unlock when correct pattern is detected
                next_state = IDLE; // Return to IDLE after unlocking
            end

            default: next_state = IDLE;
        endcase
    end

endmodule
```

Listing 3.1: Verilog Code

## 3.2 Testbench.code

The following *Verilog testbench* verifies the functionality of the *vending machine using verilog HDL* module by simulating different scenarios.

```
`timescale 1ns / 1ps
module digital_lock_TB;

    reg clk, reset, in_bit;
    wire lock;

    // Instantiate the digital lock module
    digital_lock uut (
        .clk(clk),
        .reset(reset),
        .in_bit(in_bit),
        .lock(lock)
    );

    // Clock generation (10-time unit period)
```

```verilog
    always #5 clk = ~clk;

    initial begin
        // Initialize signals
        clk = 0;
        reset = 1;
        in_bit = 0;

        // Apply reset
        #10 reset = 0;

        // Test Case 1: Correct pattern "1101"
        #10 in_bit = 1;   // Input 1
        #10 in_bit = 1;   // Input 1
        #10 in_bit = 0;   // Input 0
        #10 in_bit = 1;   // Input 1
        #10 in_bit = 0;   // Keep in_bit LOW after pattern

        // Wait to observe lock activation
        #20;

        // Test Case 2: Incorrect pattern "101"
        reset = 1;  // Reset FSM
        #10 reset = 0;
        #10 in_bit = 1;   // Input 1
        #10 in_bit = 0;   // Input 0
        #10 in_bit = 1;   // Input 1
        #10 in_bit = 0;   // Keep in_bit LOW after pattern

        // Wait to observe that lock does not activate
        #20;

        // End simulation
        $stop;
    end

endmodule
```

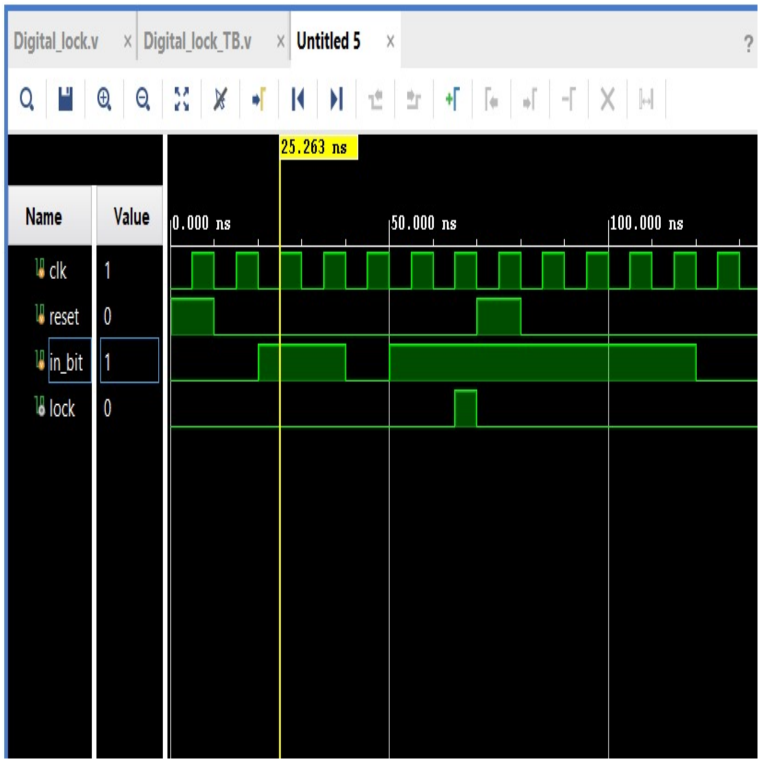
Listing 3.2: Testbench Code

# 4. WORKING

**Working of the Digital Lock System:**
The digital lock operates as a synchronous Moore finite state machine that detects the specific binary pattern "1101" in a serial bit stream. Starting from the IDLE state, the system progresses through successive states (S1 $\rightarrow$ S11 $\rightarrow$ S110 $\rightarrow$ S1101) only when each incoming bit matches the expected sequence. The state transitions occur at every rising clock edge, with the machine advancing to the next state for correct bits or resetting to IDLE for mismatches. Upon complete pattern recognition, it enters the UNLOCK state, activating the lock signal for one clock cycle before automatically returning to IDLE. This non-overlapping design ensures security by requiring fresh entry of the entire pattern for each unlock attempt. The system features six distinct states encoded using 3-bit binary representation for efficient hardware implementation. An asynchronous reset input provides immediate initialization capability. The Moore machine architecture guarantees stable outputs dependent only on the current state, eliminating glitches during transitions. The design processes one input bit per clock cycle, with combinational logic determining next states and sequential elements maintaining the current state. Testbench verification confirms correct operation by validating both successful unlocks with the "1101" pattern and rejection of incorrect sequences, demonstrating reliable pattern recognition and security against unauthorized access attempts.

## 5. SIMULATION RESULT

The Result shown below can be verified by comparing it with the timing diagram.



# 6. CONCLUSION

This project successfully designed and implemented a secure digital lock system using a non-overlapping Moore machine architecture in Verilog HDL. The system demonstrates precise detection of the "1101" binary pattern through a carefully structured six-state finite state machine, ensuring reliable operation and enhanced security. Key features include synchronous state transitions on clock edges, automatic reset on incorrect inputs, and stable output generation characteristic of Moore machines. The design was rigorously verified through comprehensive testbench simulations in Xilinx Vivado, confirming correct functionality for both valid and invalid input sequences.

The implementation showcases efficient hardware utilization with 3-bit state encoding while maintaining robust security through its non-overlapping pattern recognition approach. Practical applications include electronic door locks, safe mechanisms, and authentication systems. The project effectively bridges theoretical FSM concepts with practical digital design, highlighting the effectiveness of HDL-based development for sequential logic systems. Future enhancements could incorporate multiple authorized patterns, variable sequence lengths, or additional security layers like attempt limits. This work provides a solid foundation for developing more complex digital security solutions while demonstrating the practical implementation of fundamental digital design principles.

# 7. REFERENCES

- https://www.asic-world.com/verilog/index.html
- https://www.fpga4student.com/2017/02/finite-state-machine-on-fpga.html
- https://ieeexplore.ieee.org/document/1620780
- https://www.nandland.com/vhdl/modules/lock-in-fpga.html
- https://docs.xilinx.com/r/en-US/ug900-vivado-logic-simulation
- https://www.chipverify.com/verilog/verilog-testbench