**Lebanese American University**

A close-up of a logo

AI-generated content may be incorrect.

***COE322 – Logic Design Lab***

***Final Project Spring 2025***

Ahmad Noura 202307409

05/13/2025

# **Table of** Content

[Table of Content 2](#_Toc198074275)

[Table of Figures 3](#_Toc198074276)

[Table of Tables 4](#_Toc198074277)

[Introduction 5](#_Toc198074278)

[Equipment needed 6](#_Toc198074279)

[Design and Analysis 7](#_Toc198074280)

[Key Design Advantages 9](#_Toc198074281)

[Design breakdown: 10](#_Toc198074282)

[Mind Map Explanation Logic-Controlled Board 24](#_Toc198074283)

[Power-On and Boot Phase 24](#_Toc198074284)

[Switch Polling and Tracking 24](#_Toc198074285)

[Selecting the Next Sequence 24](#_Toc198074286)

[Sequence Execution and LED Activation 25](#_Toc198074287)

[Reset and Timeout Handling 25](#_Toc198074288)

[Error Management 25](#_Toc198074289)

[Summary 25](#_Toc198074290)

[Conclusion 26](#_Toc198074291)

# Table of Figures

[Figure 1- the firure shows the design of the final circuit 10](#_Toc198074263)

[Figure 2- sequance detector 11](#_Toc198074264)

[Figure 3-Sequence detector simulation 12](#_Toc198074265)

[Figure 4- Sequence 1 design 13](#_Toc198074266)

[Figure 5-sequence 2 design 14](#_Toc198074267)

[Figure 6- Sequence 3 design 15](#_Toc198074268)

[Figure 7-Sequence 4 design 15](#_Toc198074269)

[Figure 8-Sequence 1 simulation 16](#_Toc198074270)

[Figure 9-Locked state simulation 18](#_Toc198074271)

[Figure 10-final circuit design (reuploaded) 21](#_Toc198074272)

[Figure 11-Finalized simualtion 22](#_Toc198074273)

[Figure 12-Mind Map of the project design 23](#_Toc198074274)

# Table of Tables

[Table 1-Next sequence based on the last turned off switch 8](#_Toc198074259)

[Table 2-identifying output connections 17](#_Toc198074260)

# Introduction

This project focuses on the design and implementation of a logic-controlled board system using logic design principles such as Boolean algebra, FSMs, and circuit minimization. The board changes LED activation sequences based on user interaction, specifically the last switch turned off. The design involves FSM planning, simulation using Quartus, hardware implementation, This Logic-Controlled Board was created as a project to bring together all the digital logic techniques learned in class. With four switches and four LEDs, the board reacts based on the last switch turned off, creating an interactive experience. Each switch is mapped to an LED by default, but sequence changes, locking mechanisms, and switch disabling features add layers of logic complexity.

# Equipment needed

1. Breadboard​
2. Logic Gates Ics

* D flip flops MUX AND OR NOT gates

1. Clock design
2. 4 toggle switches to provide inputs for the circuits.
3. Wires
4. Boss Wires
5. Switches: To provide input values​
6. LEDs
7. Altera Quartus II Software: For circuit design and simulation​

# Design and Analysis

The Logic-Controlled Board is built around a finite state machine (FSM) that manages dynamic LED sequencing based on switch interactions. The system consists of four switches and four LEDs, each initially assigned a fixed mapping. However, once all switches are turned off, the system enters a reset phase triggered by a timer, and the FSM updates the LED activation sequence based on which switch was turned off last. Each switch corresponds to a specific sequence that determines the order in which LEDs light up upon reactivation. This design required careful planning of state transitions, encoding of sequences, and logic minimization using Karnaugh maps. The functionality was first simulated using Quartus to verify behavior before implementing the design on hardware through modular circuit block

Inputs and Outputs:

**Inputs:**

* **4 toggle switches (SW1, SW2, SW3, SW4)** each mapped to one of four colored lamps (Red, Green, Blue, Yellow).
* The system also tracks the **last switch turned off**, stored internally as part of the state logic.
* **Timer input** triggered when all switches are OFF for 4 seconds (marks reset condition).

**Outputs:**

* **4 LEDs (L1–L4)** each corresponding to a lamp that lights up based on the current active sequence.
* **7-segment display** shows the number of the active sequence (1 to 4), useful in practice mode for tracking internal logic behavior.

FSM Characteristics

This system is implemented as a **Moore Machine**, where:

* The **output (LED behavior and 7-segment value)** depends only on the **current state**.
* Each state represents either:
  + A **boot/reset condition**
  + A **static sequence state (S1–S4)**
  + Or a **temporary mode like Locked or Disabled (capless trick in Sequence 3)**

The FSM handles user interaction, sequence updates, switch tracking, and LED response behavior.

States and Representations

Each **state** corresponds to a distinct behavioral phase of the system:

| FSM State | Description |
| --- | --- |
| Boot | Initial power-on check; enters normal or locked mode |
| Locked | SW2 was ON during power-up; disables dynamic logic |
| Sequence Detector | Identifies last switch turned off (after timeout) |
| Sequence 1 | Standard LED order: 1 → 2 → 3 → 4 |
| Sequence 2 | LED order: 2 → 3 → 4 → 1 |
| Sequence 3 | LED order: 3 → 4 → 1 → 2 |
| Sequence 4 | LED order: 4 → 3 → 2 → 1 |
|  | |  |  |  | | --- | --- | --- | | Switch | Binary | Triggers This Sequence | | SW1 | 00 | SEQ1 → 1 → 2 → 3 → 4 | | SW2 | 01 | SEQ2 → 2 → 3 → 4 → 1 | | SW3 | 10 | SEQ3 → 3 → 4 → 1 → 2 | | SW4 | 11 | SEQ4 → 4 → 3 → 2 → 1 | |

Table -Next sequence based on the last turned off switch

The last switch the was trurned off determines the next sequence of Leds which is encoded based on the table

The logic controlling transitions between states is based on:

* **Last switch turned OFF** → selects next sequence.
* **All switches OFF for 4 seconds** → triggers reset, reevaluates sequence.
* **Cap removed** (in Sequence 3) → temporarily disables the switch (special state).
* **Switch reinserted or timeout ends** → re-enables switch functionality.

➤ From Boot:

* If SW2 is ON → enter **Locked** state.
* Else → enter **Sequence Detector** to evaluate the next active sequence.

➤ From Sequence Detector:

* If last switch off was SW1 → go to **Sequence 1**
* If SW2 → **Sequence 2**
* If SW3 → **Sequence 3**
* If SW4 → **Sequence 4**

➤ From Sequence States:

* Behavior depends on switch inputs and current sequence. For example:
  + In **Sequence 3**, if a lamp is turned OFF and the cap is removed, enter **Switch Disabled** temporarily.
  + In all sequences, system resets if all switches OFF for >4s.

➤ From Switch Disabled (in Sequence 3):

* If cap is restored or switch remains OFF for 4s → return to Sequence 3 state.

Output Behavior

* In every sequence, the **LEDs light up in a unique order**.
* Regardless of cap positions or physical switch swaps, **the system maintains correct LED logic**.
* The **7-segment display** shows the sequence number (1–4) only in **practice mode**, and is detachable in performance mode.

## Key Design Advantages

This project demonstrates several key strengths in both design structure and logical execution. The system supports **real-time switch tracking**, allowing it to update the LED sequence based on which switch was last turned off, without requiring any external reset. The logic is built around a **modular finite state machine (FSM)**, which simplifies the design and makes it easier to simulate, analyze, and expand if needed. Each part of the system is organized into separate functional blocks, including switch detection, sequence control, and output management. The Design is clean, scalable, and reinforces core logic design concepts like sequence control, state tracking, and modular implementation.

## Design breakdown:

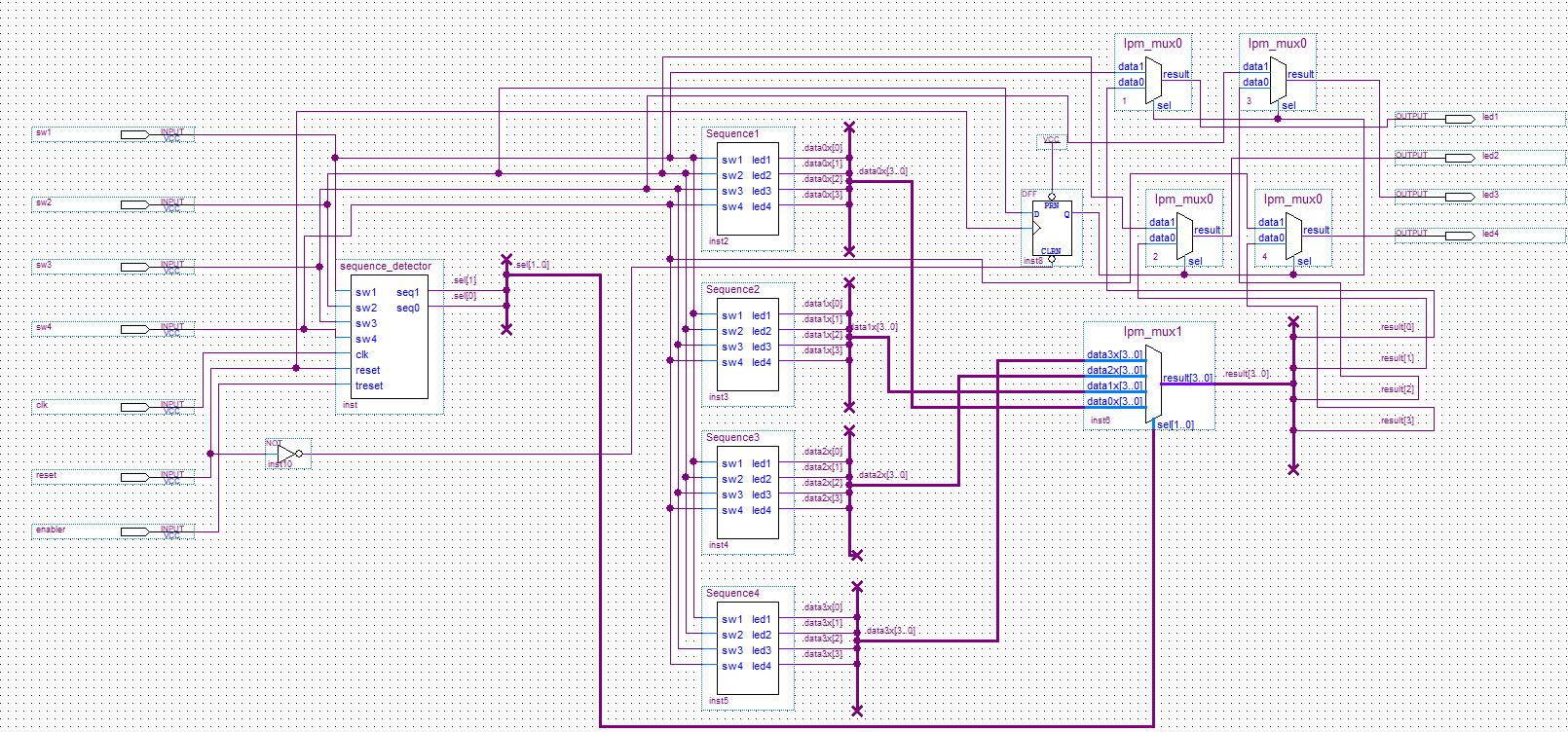


Figure - the firure shows the design of the final circuit

We will break down the final circuit and get into details while doing that. As observed there are 4 inputs (SW1, SW2, SW3, SW4) and a clock (clk) and reset and an enabler. All of these are first connected to a block which is called sequence detector.

* Objective of the sequence detector:

The purpose of this circuit is to implement a Sequence Detector capable of identifying the most recently turned-off switch among four inputs (SW1–SW4) and updating the system state accordingly. This forms the core of the dynamic behavior in the Logic-Controlled Board project, allowing flexible LED activation sequences based on user interaction.

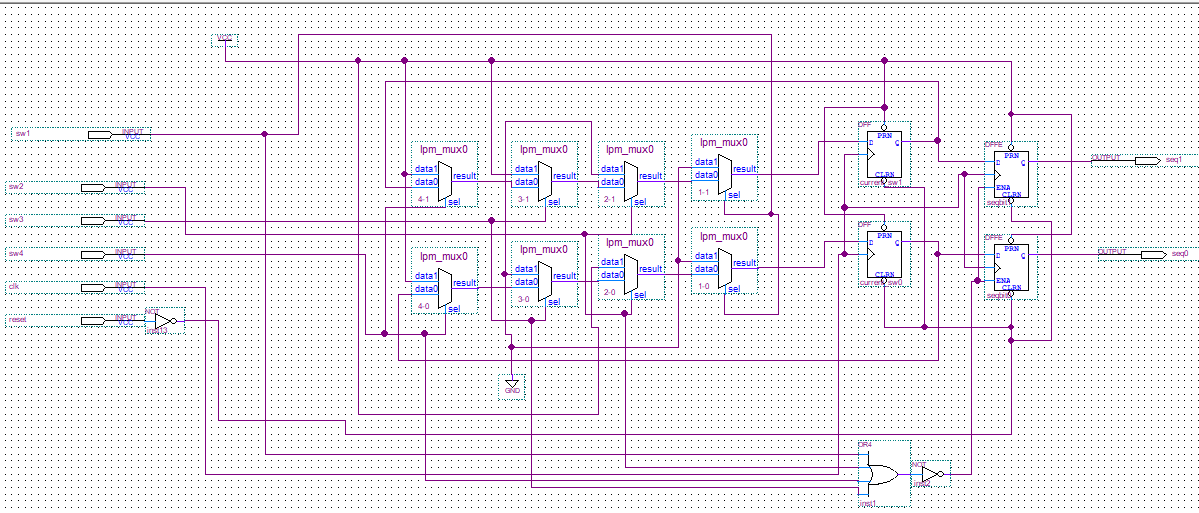


Figure - sequance detector

The figure above shows what is inside the sequence detector block.

* Circuit description:

SW1–SW4: The 4 primary input switches.

CLK: Clock input to drive state transitions.

RESET: Used to reset the system.

Clock (CLK): Synchronizes state transitions.

Reset: Clears all flip-flops for initial or reset conditions

Multiplexers: Each multiplexer takes two data inputs and one select line. Based on the value of the sel input, the appropriate data is passed to the result output. These MUXes are responsible for determining which switch data is passed to the state flip-flops during clock edges. This allows the circuit to remember the previous state, and determine which sequence (1 → 2 → 3 → 4, or any shifted version) should be active.

Flip-Flops: Each D flip-flop stores the result of a MUX, thus preserving the detected switch state across clock cycles. Flip-flops are edge-triggered and receive data inputs from MUXes and output to the next stage logic. The arrangement of 4 flip-flops here suggests a 2-bit or 4-bit state encoding system.

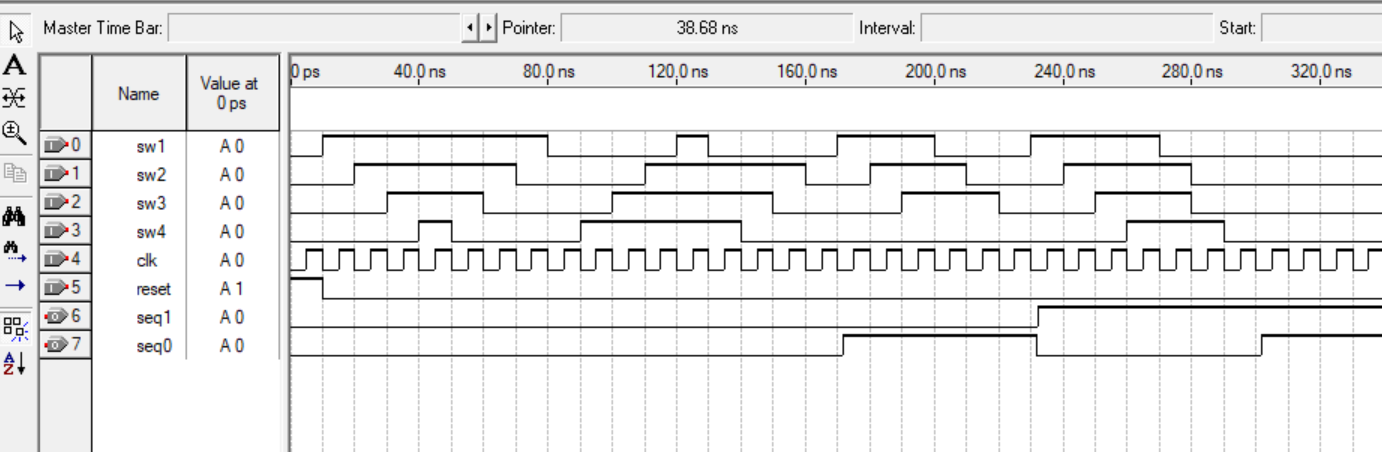


Figure -Sequence detector simulation

In this simulation, we can see the seq1 and seq0 lines changing in response to how the switches are turned off. These two signals together represent the selected sequence, and we see the values update correctly to 00, 01, 10, and 11 as expected. There's a small delay between the switch actions and the change in sequence, which in simulation is about 10 ns. This models the real-world 4-second waiting period used to confirm the switches are all off and detect the last one that was turned off. The delay ensures the system doesn’t switch too quickly and helps avoid false readings. Since this behavior is consistent across all four sequences, this simulation confirms the proper functioning of the Sequence Detector logic.

After implementing the sequence detector and running through the system's logic flow, we were able to accurately identify which specific sequence whether sequence one, two, three, or four was being detected. This outcome was determined by observing the state transitions which allowed us to classify 4 Sequences:

Sequence 1:

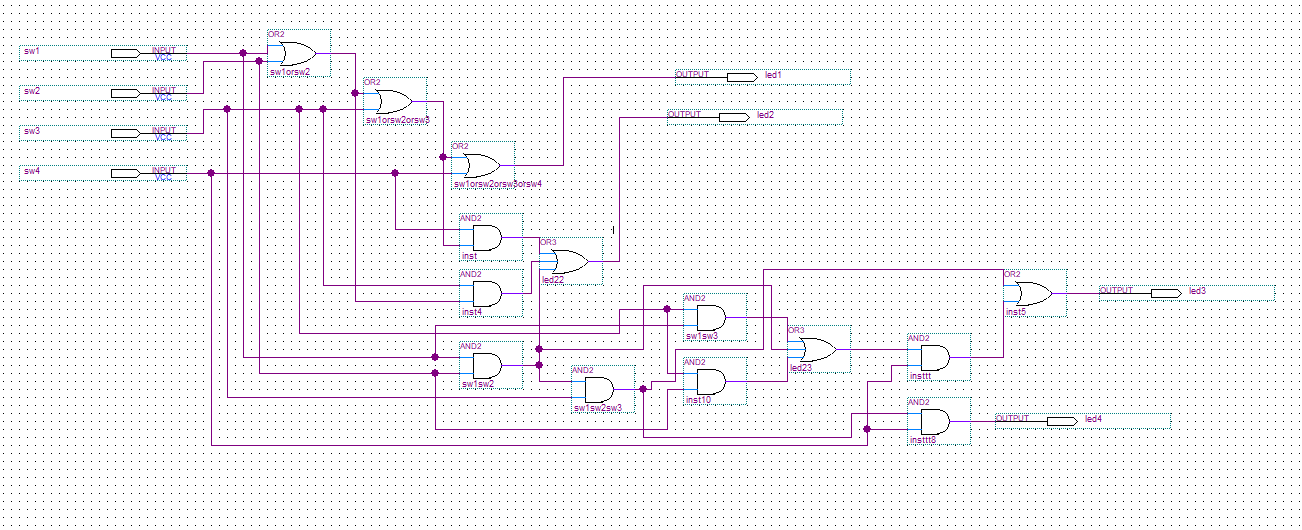


Figure - Sequence 1 design

The sequence detection circuit is composed of four switches (SW1 to SW4) and a combination of OR and AND gates used to control four output LEDs (LED1 to LED4). Each LED lights up only when a specific logic condition is met. **LED1** is controlled by a series of OR gates combining SW1 to SW4, and only lights up when a specific condition is met. **LED2** uses both OR and AND gates, requiring a combination of SW2 or SW3 along with SW3 and SW4 being ON**. LED3** checks different two-switch pairs using AND gates, combines them through an OR gate, and verifies one more condition before turning ON. **LED4** is the most restrictive, activating only when all four switches are ON together through a chain of AND gates.

Led1 = s1 + s2 + s3 + s4

Led2 = s4(s1+ s2 + s3) + s3(s1 + s2) + s1s2

Led3 = s4(s2s3 + s1s3 + s1s2) + s1s2s3

Led4 = s1s2s3s4

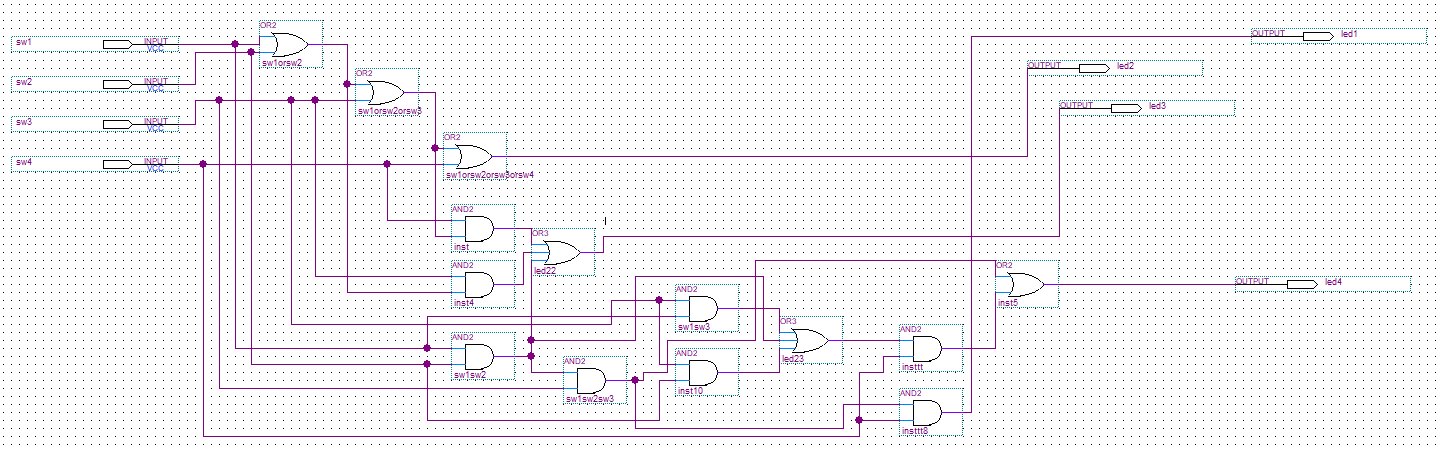
Sequence 2:

Figure -sequence 2 design

Led2 = s1 + s2 + s3 + s4

Led3 = s4(s1+ s2 + s3) + s3(s1 + s2) + s1s2

Led4 = s4(s2s3 + s1s3 + s1s2) + s1s2s3

Led1 = s1s2s3s4

Sequence 3:

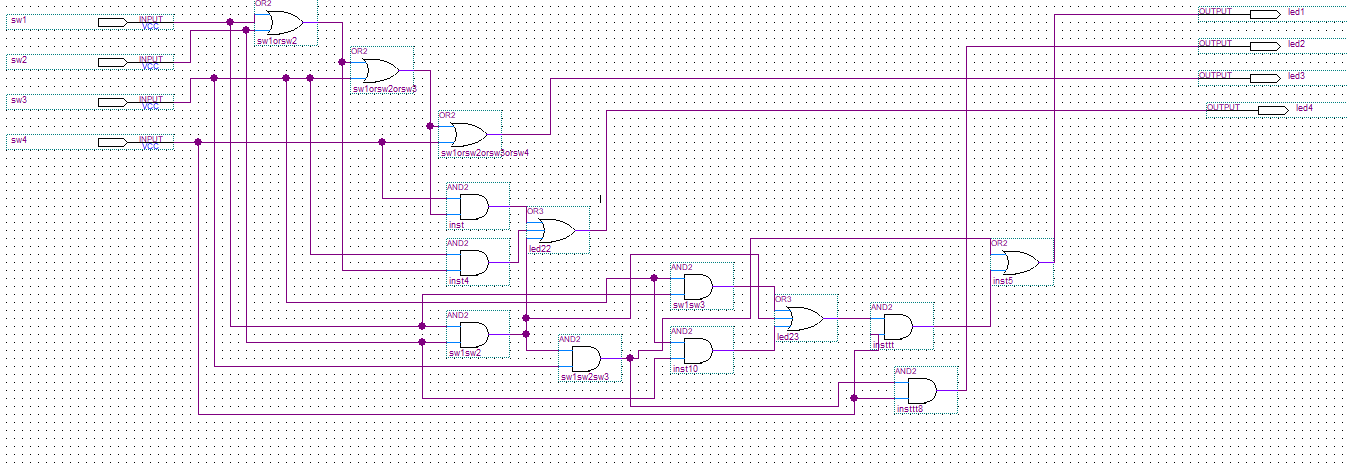


Figure - Sequence 3 design

Led3 = s1 + s2 + s3 + s4

Led4 = s4(s1+ s2 + s3) + s3(s1 + s2) + s1s2

Led1 = s4(s2s3 + s1s3 + s1s2) + s1s2s3

Led2 = s1s2s3s4

Sequence 4:

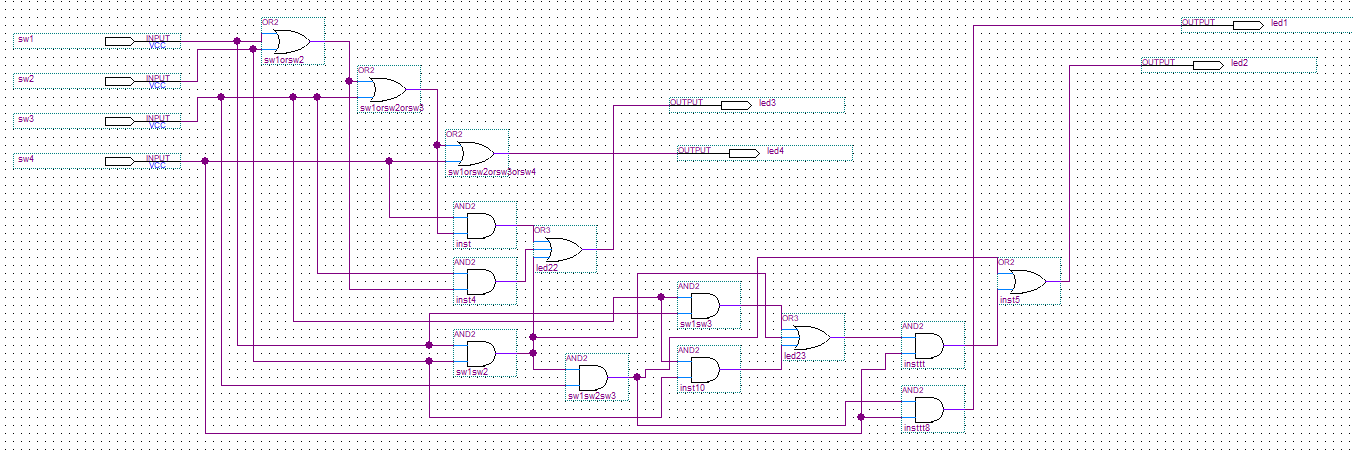


Figure -Sequence 4 design

Led4 = s1 + s2 + s3 + s4

Led3 = s4(s1+ s2 + s3) + s3(s1 + s2) + s1s2

Led2 = s4(s2s3 + s1s3 + s1s2) + s1s2s3

Led1 = s1s2s3s4

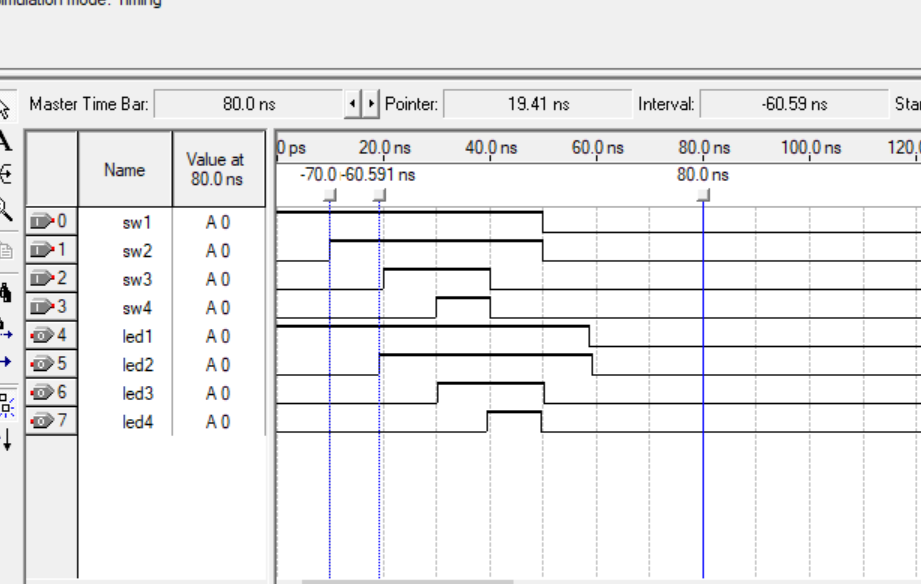


Figure -Sequence 1 simulation

This simulation clearly demonstrates how the system activates the correct led sequence based on the last switch turned off. In this case, sw1 was turned off last, triggering sequence 1, which lights the leds in the order: led1, led2, led3, and led4.

The same logic applies to the other sequences (2, 3, and 4) as well. Each sequence simply shifts the led activation pattern based on which switch was last to go low. Since the equations and logic structure are nearly identical across all sequences only the output mapping changes simulating each one separately would be repetitive. This waveform is representative of all sequences, as the core functionality and switching behavior remain the same. This simualtion verified our equations then sequence 2 3 and 4 follow the same equations but different mapping based on the order and equations specified above.

In the simulation, we notice a short delay between the point when all switches are turned off and when the leds begin their sequence. This delay simply represents the system taking time to process the input change and trigger the correct output. Since all four sequences use the same structure and logic, this delay appears in all of them in the same way. The only thing that changes between the sequences is the order of the led activations. So simulating one sequence is enough to understand the behavior of the others, and repeating the simulation for each one would be redundant.

**Locked state**:

The Locked state makes the system appear to behave in a simple 1 to 1 switch to LED mapping effectively disabling the dynamic sequence logic and hiding the trick from an observer. It’s triggered if the system starts while SW2 is ON.

1. Component Used
   * 4 LPM\_MUX0 modules (top right)
   * Each MUX connects:
     + data0 = output from the dynamic sequence module
     + data1 = direct switch input (SW1, SW2, SW3, SW4)
     + sel = control signal determining whether locked state is active
2. Selector Input
   * The sel input of each MUX is controlled by the locked state signal.
   * This signal is likely coming from the FSM inside the sequence detector block it becomes high (1) when:
     + The system is booted with SW2 ON, as described in your project manual.
3. MUX Behavior
   * If sel = 0 → Output = data0 (Normal FSM-based dynamic output)
   * If sel = 1 → Output = data1 (Direct switch-to-LED mapping)
4. Final Output
   * Each MUX's output is connected to one LED:

Table -identifying output connections

|  |
| --- |
| * + - MUX0 → LED1 |
| * + - MUX1 → LED2 |
| * + - MUX2 → LED3 |
| * + - MUX3 → LED4 |

This means:

* When Locked, flipping a switch directly turns on/off its matching LED.
* When Unlocked, the system uses the stored sequence info and state logic to control LEDs.

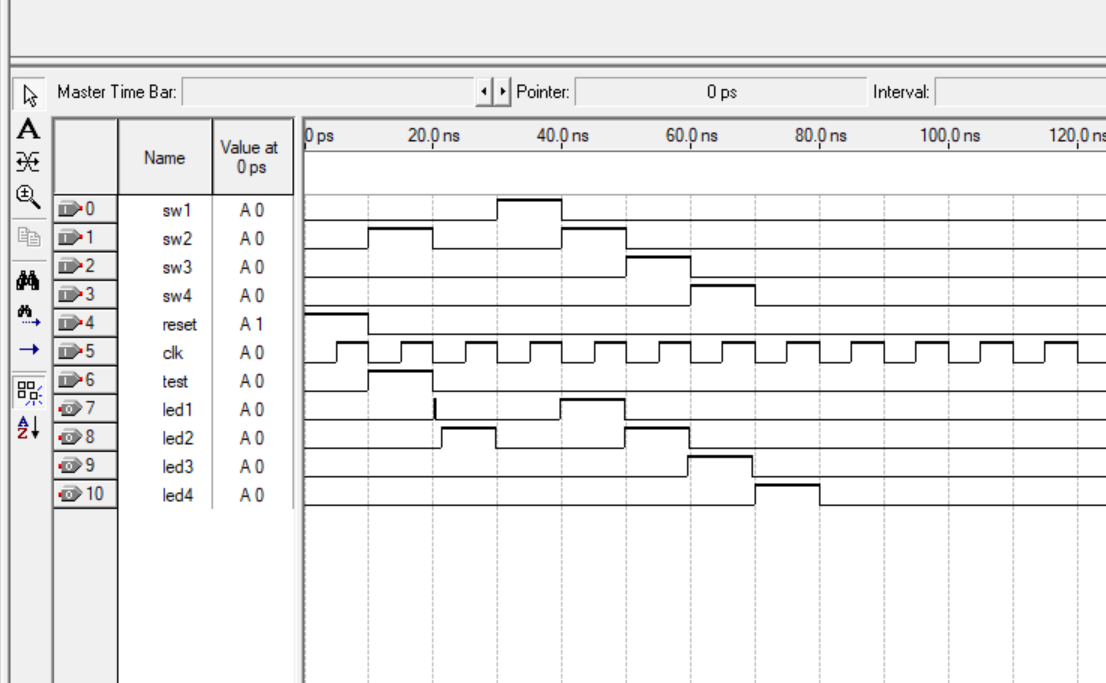


Figure -Locked state simulation

**Explanation for how we built the final circuit:**

The final circuit for the Logic-Controlled Board was designed to detect the last switch turned off, control a sequence of LEDs dynamically, and include a locked mode for presentation purposes. It was built in Altera Quartus II using a modular, FSM-based design. Below is a breakdown of the construction process.

Step 1: Designing the Sequence Detector (FSM Block)

We started by building a Finite State Machine (FSM) named sequence\_detector, which is the heart of the system.

* Inputs:
  + SW1–SW4: Switch inputs
  + CLK: Clock input
  + RESET: Clears the FSM and internal memory
  + ENABLE: Used for timing and special tricks
* Outputs:
  + seq0, seq1: 2-bit output representing the currently selected sequence (1 to 4)
  + locked\_state: Indicates whether the system is in locked mode

Purpose:

* Detect which switch was last turned off
* Hold this information using flip-flops
* Output the correct sequence number for LED control
* Trigger locked mode if SW2 is ON at startup

Step 2: Implementing Sequence Logic Blocks

We created four separate modules to represent the 4 LED sequences:

* Sequence1, Sequence2, Sequence3, Sequence4
* Each module takes the current state of switches and outputs led1, led2, led3, led4 according to a specific LED activation order:
  + Sequence 1: 1 → 2 → 3 → 4
  + Sequence 2: 2 → 3 → 4 → 1
  + Sequence 3: 3 → 4 → 1 → 2
  + Sequence 4: 4 → 3 → 2 → 1 (reverse order)

Each block contains logic to drive the correct LED ON/OFF depending on the sequence selected.

Step 3: Using a 4-to-1 Multiplexer to Select Active Sequence

To dynamically select which sequence is active, we used an lpm\_mux1 module:

* Inputs: The 4 sequence blocks’ outputs (bundled)
* Selector: seq1 and seq0 bits from the FSM
* Output: Combined LED control signals

Function:

* Only one sequence block’s output is active at a time
* Chosen based on which switch was last turned off

Step 4: Implementing the Locked Mode

In the top-right corner of the circuit, we added the locked state logic.

* Each final LED output is passed through a 2-to-1 MUX (lpm\_mux0)
* MUX inputs:
  + data0: Dynamic LED control (from selected sequence)
  + data1: Direct connection to switches (SW1 → LED1, etc.)
* Selector (sel): Controlled by locked\_state signal from the FSM

Behavior:

* If locked\_state = 0: System operates normally (dynamic sequences)
* If locked\_state = 1: Switches directly control LEDs (1-to-1 mapping)

This makes the circuit appear simple and non-dynamic when in locked mode — perfect for “performing the trick” without revealing the logic.

Step 5: Clock, Reset, and Timing Integration

* A clock (CLK) was connected to synchronize all sequential behavior.
* A reset input (RESET) was used to return the FSM and sequence logic to the initial state.
* A timing system (possibly external or handled via FSM) is used for:
  + 4-second delay when all switches are off
  + Special Trick #3 handling (cap removal logic)

**Functional operation:**

* **Initialization**: When the circuit is powered on or reset, the FSM enters the initial Boot or idle state. All flip-flops are cleared.
* **Switch Detection**: Each switch has a path leading to a MUX (multiplexer) and flip-flop structure. There are 4 main sets of MUX + Flip-Flop blocks, each representing a switch. Each MUX (lpm\_mux0) has: Two inputs (data0, data1), A selector (sel), One output (result). The output of each MUX is connected to a D flip-flop. If a switch is turned OFF, its signal becomes low (0). The selector logic detects this and updates the MUX output, which then gets clocked into the flip-flop.

This way, the flip-flop stores which switch was turned off last effectively encoding the most recent user action.

* **MUX Logic Selection**: The MUXes dynamically choose whether to load a new input value or retain the old state, based on switch changes and the current FSM state.
* **Sequence Update**: Based on which switch was turned off last, the FSM transitions to a state representing one of the four sequences. The correct LED activation pattern (e.g., 2→3→4→1) will be enabled.
* **State Holding**: Flip-flops maintain the detected state until a new switch event is registered or the system is reset.
* **Clock Dependency**: All state transitions and flip-flop updates are synchronized with the clock signal, ensuring reliable sequential behavior.

Now that we’ve broken down each part of the system in detail from switch detection to sequence logic and outputs we’re bringing back the full circuit image to show how everything fits together. You can now identify each block we explained and see how they’re all connected in the final design.

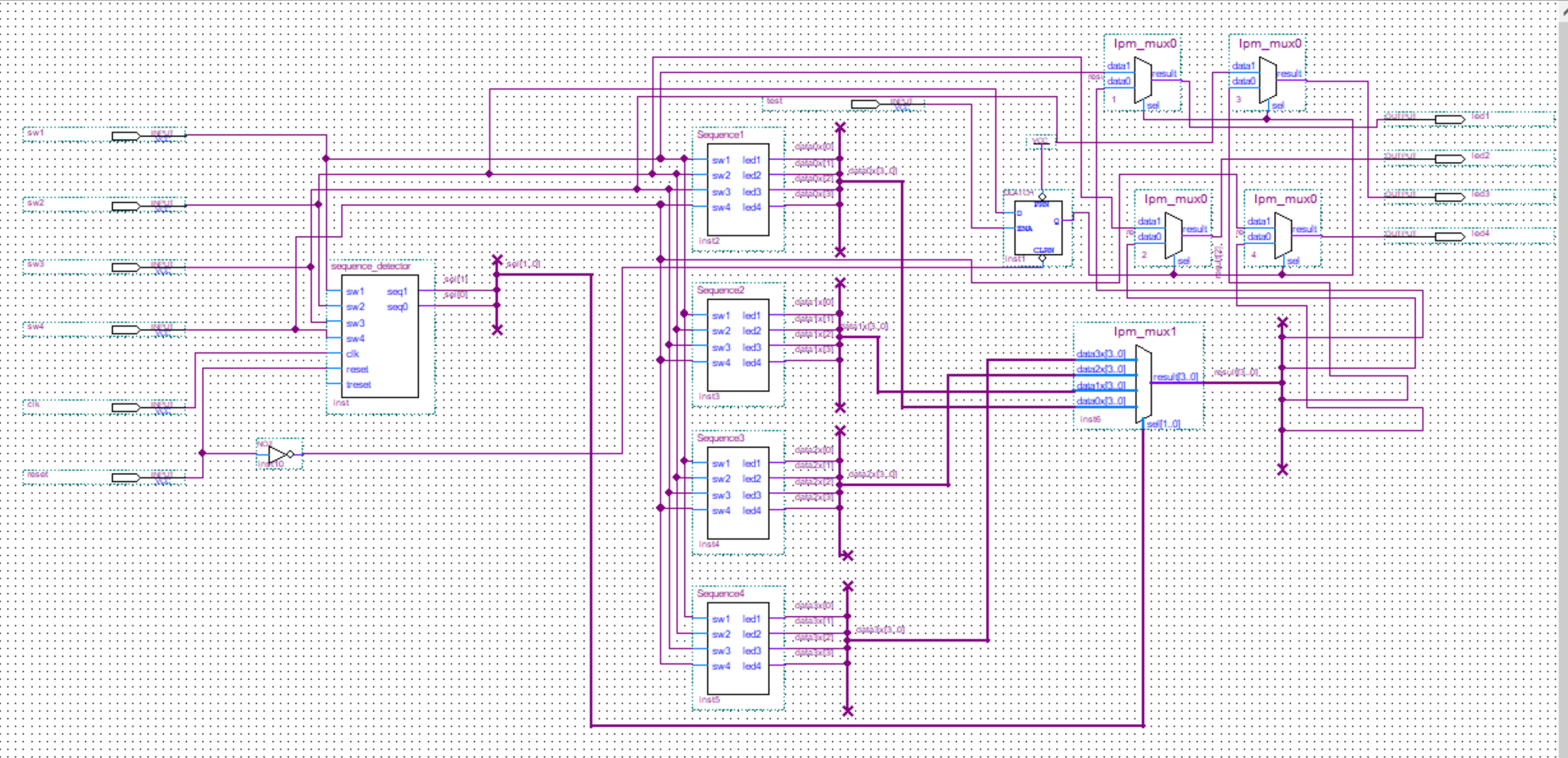


Figure -final circuit design (reuploaded)

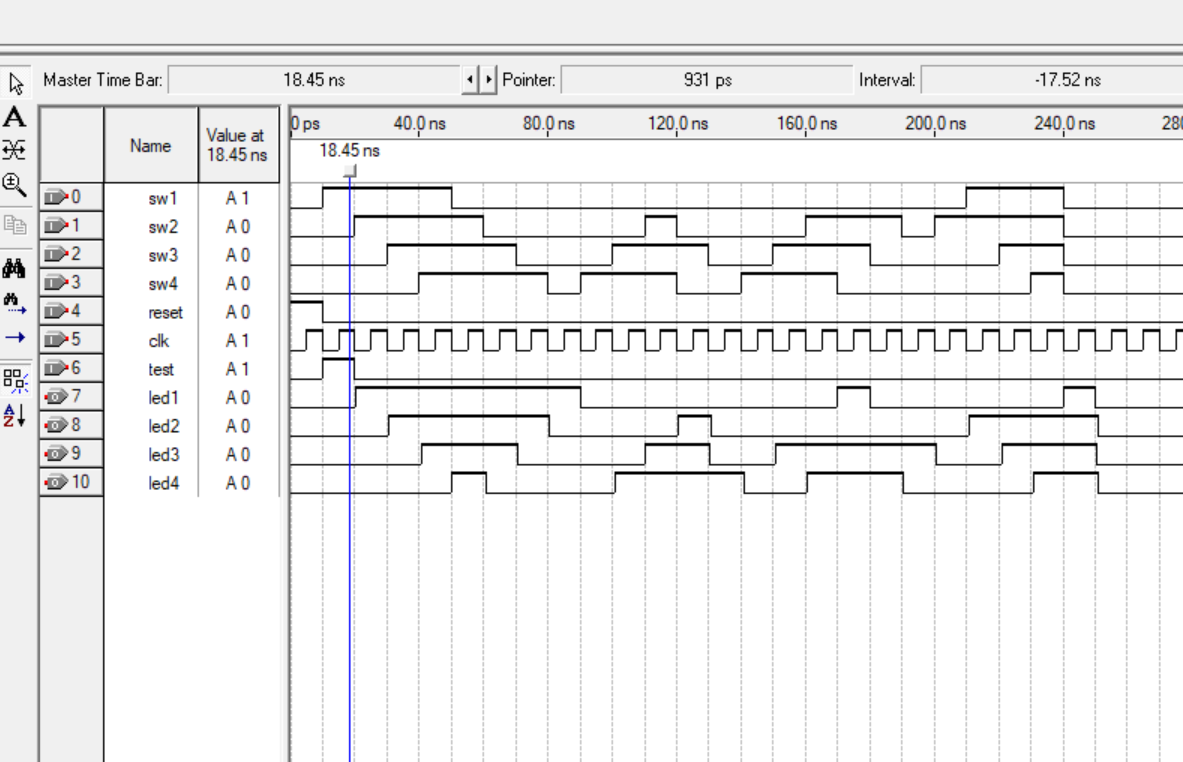


Figure -Finalized simualtion

In this simulation, we observe the complete behavior of the fully integrated logic-controlled board. The top four signals (sw1 to sw4) represent the switches being toggled at different moments across the timeline. We can clearly see that each is being toggled one after the other, with brief delays between each. This simulates a real user sequentially turning off the switches. The clk line is a consistent square wave, serving as the timing base for the logic and synchronizing state transitions. The reset signal starts high and then goes low, which allows the system to begin operating.

Immediately after all switches are off, there's a short delay before the led outputs (led1 to led4) begin to change. This delay is expected and represents the system’s internal processing time checking that all inputs are off, detecting the last switch that was turned off, and determining the appropriate sequence to activate. This brief pause (roughly 10–20 ns) in simulation is equivalent to the longer 4-second timeout in hardware that would be implemented using a timer circuit.

As the leds begin toggling, we observe a clear and structured activation pattern. Each led goes high at a different point in time, one after another, and then returns low. This confirms that the system correctly moved into one of the defined sequences (sequence 1, 2, 3, or 4), and executed it according to the timing logic programmed in that sequence module. The led transitions are synchronized with the clock, showing that the outputs are being driven based on proper timing rather than random toggles.

Although the waveform does not include the internal sequence code outputs (like seq1 and seq0), the ordered led behavior is enough to confirm that a valid sequence was selected and executed. Since the structure is the same across all four sequences and only the order of leds changes, the observed behavior validates the entire system pipeline from switch detection to sequence activation to led output.



Figure -Mind Map of the project design

# Mind Map Explanation Logic-Controlled Board

The mind map presented in this project represents the complete behavioral flow of the Logic-Controlled Board. It visualizes how the system operates step by step, starting from the moment it is powered on, all the way through normal use, error handling, and sequence selection. Each block or node in the mind map corresponds to a logic decision, a transition between states, or a condition being checked by the finite state machine (FSM) that governs the system.

## **Power-On and Boot Phase**

When the system is powered on, it immediately enters the 'BOOT' state. At this stage, it checks if switch SW2 is ON. If it is, the system enters 'LOCKED' mode, which disables all dynamic behavior and directly maps each switch to its corresponding LED (1-1, 2-2, 3-3, 4-4). If SW2 is not ON, the system proceeds into 'NORMAL\_OP' mode. This represents the standard operation mode of the Logic-Controlled Board where sequences, logic, and tricks are enabled.

## **Switch Polling and Tracking**

In normal operation, the system continuously scans and polls all switches through the 'POLL\_ALL' and 'SCAN\_SWITCHES' states. If any switch is OFF, the system monitors further changes. Once all switches are OFF, it begins a 4-second timeout. During this period, the FSM identifies which switch was turned OFF last — referred to as 'last\_sw'. This information is stored and later used to determine which LED activation sequence to follow.

## **Selecting the Next Sequence**

After the timeout completes and all switches remain OFF, the FSM enters the 'SELECT\_SEQUENCE' state. Based on the value of 'last\_sw', it maps the system to one of four predefined sequences:  
- SW1 (00) → Sequence 1  
- SW2 (01) → Sequence 2  
- SW3 (10) → Sequence 3  
- SW4 (11) → Sequence 4  
Each sequence defines a different order for the LED activation. Sequence 1 follows LED order 1-2-3-4, while Sequence 4 reverses the order to 4-3-2-1.

## **Sequence Execution and LED Activation**

Once a sequence is selected, the corresponding logic activates the LEDs in a fixed order defined by that sequence. The logic ensures that LEDs turn on one at a time in a visually recognizable order. This creates an interactive illusion that the switches and LEDs are smartly connected, while the underlying logic is handled by the FSM.

## **Reset and Timeout Handling**

If no input is detected for an extended period, such as 4 or 5 seconds, the system automatically enters a reset state. The FSM uses this condition to reevaluate the last switch interaction and update the active sequence accordingly. This ensures the system is always ready for the next interaction and does not get stuck in an undefined state.

## **Error Management**

The FSM includes built-in handling for signal bounce and input noise. States such as 'DEBOUNCE', 'VERIFY\_OFF', and 'CONFIRMED\_OFF' validate input stability. Additionally, conditions like 'FALSE\_ALARM', 'INVALID SW', and 'ERROR' are used to catch abnormal behaviors such as multiple switches turning off simultaneously or faulty toggles. These safeguards keep the system stable and accurate.

## **Summary**

Overall, the mind map outlines the complete flow of the Logic-Controlled Board's operation, from initialization to sequence execution and recovery. The FSM ensures each part of the system runs reliably, following a clear logic based on user input. The use of modular states allows for clean implementation and future expandability, making the design both educational and practical.

# Conclusion

This project provided a strong opportunity to apply key concepts from logic design in a hands-on and meaningful way. The Logic-Controlled Board was able to respond accurately to user inputs by detecting the last switch turned off and updating the LED sequence accordingly. Its behavior was driven by a well-structured finite state machine that ensured consistent and predictable transitions between states. The system was designed to be clear, modular, and easy to follow. Beyond functionality, the project helped reinforce our understanding of how theoretical logic can be translated into practical digital systems. It highlighted the importance of thoughtful design, input handling, and state control in building reliable interactive circuits.