**Lebanese American University**



***COE322 – Logic Design Lab***

***Logic Controlled Board Project***

Charbel Abi Saad | 202205909

Chiara Bassil | 202208400

Hassan Hamdan | 202205362

Zayd El Hajj | 202304488

**Table of Contents**

[**Abstract** 5](#_Toc198056390)

[**Introduction** 6](#_Toc198056391)

[**Components and Equipment Used** 7](#_Toc198056392)

[**Analysis** 8](#_Toc198056393)

[**Project Requirements** 9](#_Toc198056394)

[**Detailed Description** 9](#_Toc198056395)

[**Visual Indicators** 11](#_Toc198056396)

[**Paper Design** 12](#_Toc198056397)

[**State Diagrams, K-maps, and Equations** 12](#_Toc198056398)

[**NE 555- Research** 24](#_Toc198056399)

[**BCD Decoder – Research** 26](#_Toc198056400)

[**Quartus Design and Analysis** 28](#_Toc198056401)

[**Breadboard Design and Analysis** 38](#_Toc198056402)

[**Financial Study** 40](#_Toc198056403)

[**Delay Calculation** 41](#_Toc198056404)

[**Power Consumption Analysis** 42](#_Toc198056405)

[**Problems Faced During the Project** 43](#_Toc198056406)

[**Advantages of our Design** 44](#_Toc198056407)

[**Conclusion** 45](#_Toc198056408)

[**References** 46](#_Toc198056409)

**Tables of Figures**

[Figure 1- Desired Project 8](#_Toc198056410)

[Figure 2 - Astable mode 24](#_Toc198056411)

[Figure 3 - Timer IC Internal Circuit 25](#_Toc198056412)

[Figure 4 - 555 timer pin diagram 26](#_Toc198056413)

[Figure 5 - Pin, Names, and Purpose 26](#_Toc198056414)

[Figure 6 - BCD decoder pin configuration 27](#_Toc198056415)

[Figure 7 - Quartus design for the treeset 28](#_Toc198056416)

[Figure 8 - Quartus design for LED1 29](#_Toc198056417)

[Figure 9 - Quartus design for LED2 30](#_Toc198056418)

[Figure 10 - Quartus design for LED3 31](#_Toc198056419)

[Figure 11 - Quartus design for LED4 32](#_Toc198056420)

[Figure 12 - Quartus design for the sequence detector 33](#_Toc198056421)

[Figure 13 - Quartus design for the Flags 34](#_Toc198056422)

[Figure 14 - Quartus design for the Decoder 35](#_Toc198056423)

[Figure 15 - Quartus design for A 35](#_Toc198056424)

[Figure 16 – Quartus design for B 36](#_Toc198056425)

[Figure 17 – Quartus design for C 36](#_Toc198056426)

[Figure 18 - Quartus design for the 7-segment display 37](#_Toc198056427)

**Table of Tables**

[Table 1 - BOOT state transitions and outputs 12](#_Toc198056428)

[Table 2 - Locked state transitions and outputs 13](#_Toc198056429)

[Table 3 - Flag-Based transitions and outputs 14](#_Toc198056430)

[Table 4 - Sequence 1 Transitions and outputs 15](#_Toc198056431)

[Table 5 - Sequence 2 transitions and outputs 16](#_Toc198056432)

[Table 6 - Sequence 3 transitions and outputs 17](#_Toc198056433)

[Table 7 - Sequence 4 transitions and outputs 18](#_Toc198056434)

[Table 8 - Special trick transitions and outputs 19](#_Toc198056435)

[Table 9 - Treset Logic 20](#_Toc198056436)

[Table 10 - Deriving the equation for A 20](#_Toc198056437)

[Table 11 - Deriving the equation for B 20](#_Toc198056438)

[Table 12 - Display Decoder Logic 21](#_Toc198056439)

[Table 13 - K-map 1 22](#_Toc198056440)

[Table 14 - K-map 2 22](#_Toc198056441)

[Table 15 - K-map 4 22](#_Toc198056442)

[Table 16 - K-map 3 22](#_Toc198056443)

[Table 17 - K-map 5 23](#_Toc198056444)

[Table 18 Delay Time for Components 41](#_Toc198056445)

[Table - Components' current consumption 42](#_Toc198056446)

# **Abstract**

This project presents the design and implementation of a logic-controlled switching board that dynamically associates four user-operated switches with four colored lamps: red, green, blue, and yellow. Unlike static connections, the system adapts based on user interactions, specifically the last switch turned off, which governs the lamp activation sequence. The project also features a mechanism where removing a switch cap disables its functionality, adding an interactive component. The system relies on combinational logic, memory elements, and a finite state machine (FSM) to ensure adaptive, consistent behavior. Simulated using Altera Quartus II, the project demonstrates how digital logic principles can be applied to interactive real-world systems.

# **Introduction**

In this project, a logic-controlled board with four switches and four colored lamps demonstrates how intelligent behavior can emerge from structured logic and finite state machines (FSMs). Although each switch seems directly linked to a same-color lamp, internal logic determines lamp activation based on the last switch turned off. Removing a switch cap temporarily disables it, simulating real-time overrides. The system maintains accurate color-to-switch mapping, even if caps are rearranged. By integrating sequential logic, memory elements, and FSM design, this project bridges core digital logic concepts with real-world interactivity. The system was implemented and tested using Altera Quartus II.

# **Components and Equipment Used**

In this project, we used the following components:

1. DIP switch 4 poles
2. IC 7408 2-input AND Gate
3. IC 7432 2-input OR Gate
4. IC 7486 2-input X-OR Gate
5. IC LM555 – Timer
6. IC 7404 Hex Schmitt Inverter
7. IC 7474 – Dual D Flip-Flop
8. Breadboard Double-sided PCB
9. IC socket
10. IC 7447 – 7-segment decoders
11. IC 74138 – 3 to 8 Line Decoder

# **Analysis**

To begin our project efficiently, we first reviewed the system’s objectives and planned how to approach the logic design. We met regularly to discuss dynamic switching behavior, understand how the system works, and identify the key logic components needed.

The following shows the final desired project:



Figure 1- Desired Project

The Logic-Controlled Board is an interactive system that controls four colored lamps (red, green, blue, yellow) through four corresponding switches. Initially, each switch directly controls its respective lamp (1-to-1 mapping), but the system's uniqueness lies in its dynamic behavior. After the last switch is turned off, the system reconfigures the lamp sequence based on which switch was turned off last. This creates a new, unpredictable sequence, engaging the user with each interaction.

The system also includes an advanced capless switch trick, where removing a switch’s cap temporarily disables its functionality, adding an extra layer of interactivity. Once the cap is replaced, the switch regains functionality, allowing the trick to be repeated. Additionally, the system has a locking mechanism to prevent dynamic behavior, where each switch is directly mapped to its corresponding lamp, ensuring no changes occur.

At the core of this project is a Finite State Machine (FSM) that controls the behavior of the system, managing the switch-lamp sequences, timers, and the capless trick. The FSM is designed to react to user inputs and drive the outputs like the LEDs and 7-segment display. This system showcases the power of digital logic circuits and sequential behavior while offering a fun and interactive way for users to explore how these concepts work.

In addition to its educational value, the principles behind this project can be applied to real-world systems like elevator controls, vending machines, and traffic light controllers, bridging the gap between theory and practical engineering applications.

## **Project Requirements**

To successfully build and implement the Logic-Controlled Board, the following requirements must be met. These specifications outline the functional and operational needs of the system, which will guide the design and implementation process.

1. **Dynamic Lamp-Switch Mapping**

* The system must dynamically alter the lamp activation sequence based on the last switch turned off.
* The initial configuration links each switch to its corresponding lamp, but after a reset, the system updates the sequence.

1. **Sequence Activation**

* The system must support four different lamp activation sequences (1 → 2 → 3 → 4, 2 → 3 → 4 → 1, 3 → 4 → 1 → 2, and 4 → 3 → 2 → 1).
* The sequence should be triggered by the last switch turned off and reset upon specific conditions.

1. **Capless Switch Trick**

* A feature where removing the cap of a switch disables its function. The switch can still be toggled, but the lamp will not light up until the cap is replaced.

1. **Locking Mechanism**

* The system should support a locked state where switches are directly mapped to their corresponding lamps (1 → 1, 2 → 2, etc.) to prevent dynamic behavior.
* A unique procedure is provided to lock or unlock the system via Switch 2.

1. **User-Friendly Display**

* A 7-segment display should indicate an active sequence in Practice Mode to assist users in tracking their interactions.

1. **FSM Design**

* The system should be controlled via an FSM that manages the state transitions based on inputs (switch toggling, timers) and outputs (LED indicators).

## **Detailed Description**

The detailed design of the Logic-Controlled Board revolves around controlling the behavior of the system through Finite State Machines and digital logic components. The FSM will govern how the switches interact with the lamps and how the system handles special conditions, such as the capless switch trick and the locking/unlocking mechanism. Additionally, the system will need to manage timers, particularly the reset timer, to track when switches are turned off and when the capless trick is in effect.

In terms of inputs, the system will monitor the state of the four switches, the reset timer, and any changes in the status of the switch caps. Based on these inputs, the FSM will determine which outputs to activate, including which lamps to turn on or off and whether the 7-segment display should be updated.

The system’s behavior will change dynamically based on user interactions, but it also includes mechanisms to maintain functionality under specific conditions. The reset condition ensures the system returns to its default state, while the locking mechanism offers a way to control the board's behavior and prevent the dynamic sequence from being detected.

To further explain:

1. **Lamp Activation Sequences**

By default, the lamp activation order is 1 → 2 → 3 → 4. However, once a switch is turned off and the system resets (after a timeout or reset condition), the sequence updates according to the last switch turned off:

* Sequence 1: If Switch 1 was last off: 1 → 2 → 3 → 4.
* Sequence 2: If Switch 2 was last off: 2 → 3 → 4 → 1.
* Sequence 3: If Switch 3 was last off: 3 → 4 → 1 → 2 (includes Special Trick #3).
* Sequence 4: If Switch 4 was last off: 4 → 3 → 2 → 1.

1. **Reset Conditions**

The system resets when:

* All switches are turned off.
* At least one switch has been learned.
* A 4-second timer expires.

After reset, the system reconfigures the lamp sequence based on the last switch turned off.

1. **Special Trick**

In Sequence 3, a capless switch behavior is implemented:

* Step A: One of the active lamps is turned off, triggering a 4-second timer. During this period, the user must remove the cap from the corresponding switch.
* Step B: The capless switch can be toggled, but the LED won't light up unless the cap is placed back.
* Step C: After the 4-second timeout, if the switch remains OFF, it becomes reactivated when the cap is replaced.

1. **Locking and Unlocking Mechanism**

* Locking
* Remove the battery, turn on Switch 2, and reinsert the battery.
* The system enters the locked state, and no dynamic switching occurs
* Unlocking
* Restart the system with Switch 2 off.
* The system returns to normal operation with dynamic behavior.

1. **FSM Design**

The system is controlled by an FSM that handles various states such as:

* Boot: The initial state where the system checks if Switch 2 is ON to decide whether to enter the locked state.
* Locked: Prevents dynamic behavior by mapping switches directly to their respective LEDs.
* Sequence Detector: Determines which switch was turned off last and assigns the appropriate sequence.
* Sequence States: Each state corresponds to one of the four possible activation sequences.
* Switch Not Working: When a switch's cap is removed, this state ensures that the corresponding LED doesn’t light up.

## **Visual Indicators**

This section focuses on how the system communicates its status to users through visual feedback. The primary indicators include the four colored lamps and the 7-segment display. The LEDs represent the active lamp sequence based on the switch states, while the 7-segment display helps users track the current sequence number when the system is in practice mode.

* LED Indicators (L1–L4): Each switch will control an LED of a matching color (Red, Green, Blue, Yellow). The FSM determines which LEDs turn on or off, corresponding to the current lamp sequence.
* 7-Segment Display: This visual component will show the current active sequence number (1–4) during practice mode. The display will be updated according to which sequence is active and can be easily detached for performance mode, maintaining the illusion of intelligent behavior.

# **Paper Design**

This section outlines the detailed logic design and finite state machine (FSM) structure that governs the behavior of the logic-controlled board, including its operational flow and control logic.

## **State Diagrams, K-maps, and Equations**

The following table defines the logic and behavior of the system during the BOOT state, based on input switches and current state bits:

Table 1 - BOOT state transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BOOT | | | | | | | | | | | |
| State | **Inputs** | | | | **Next state** | | | **Outputs** | | | |
| ABC | SW1 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 000 | X | 0 | X | X | 0 | 1 | 0 | 0 | | | |
| X | 1 | X | X | 0 | 0 | 1 |

* **Boolean Equations (BOOT):**

1. **States:**

A = 0

B = SW2’ (A’B’C’)

C = SW2 (A’B’C’)

1. **LEDs:**

Led1 = 0

Led2 = 0

Led3 = 0

Led4 = 0

During the BOOT state, the transitions depend primarily on SW2 and the condition A’B’C’. The system initializes with all LEDs off. Depending on the input on SW2, the next state toggles between B and C being activated.

The FSM transitions into a locked state with predefined inputs. This section captures the lock state behavior.

Table 2 - Locked state transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Locked | | | | | | | | | | | |
| State | **Inputs** | | | | **Next state** | | | **Outputs** | | | |
| ABC | SW1 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 001 | 0 | 0 | 0 | 0 | 001 | | | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

* **Equations:**

1. **States:**

A = 0

B = 0

C = 1 (A’B’C)

1. **LEDs:**

Led1 = SW1 (A’B’C)

Led2 = SW2 (A’B’C)

Led3 = SW3 (A’B’C)

Led4 = SW4 (A’B’C)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | Flag | | Next state | | | Outputs | | | |
| ABC | F1 | F2 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 010 | 0 | 0 | 0 | 1 | 1 | 0 | | | |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

In the locked state, the FSM simply reflects the switch input on the LEDs, gated by the state condition A'B'C.

This section captures the state behavior influenced by Flag inputs F1 and F2

Table 3 - Flag-Based transitions and outputs

The switches are don’t cares.

* **Equations:**

1. **States:**

A = (F1+F2) (A’BC’)

B = ((F1.F2) + (F1’.F2’)) (A’BC’) or (F1 XNOR F2)(A’BC’)

C = F2’ (A’BC’)

1. **LEDs:**

Led1 = 0

Led2 = 0

Led3 = 0

Led4 = 0

Flag-based states incorporate conditional logic using F1 and F2. Output remains off but internal logic varies the state transition.

This sequence manages progressive states starting from ABC = 011. Each step is defined with respect to the Treset signal and switch inputs.

Table 4 - Sequence 1 Transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sequence 1 | | | | | | | | | | | | |  |
| State | |  | **Inputs** | | | | **Next state** | | | **Outputs** | | | |  |
| ABC | | Treset | SW1 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 | Flag |
| 011 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |  |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 11 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 10 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |  |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 01 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |  |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |  |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 00 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |  |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |  |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |  |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |  |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |  |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |  |

* **Equations:**

1. **States:**

A = 0

B = 1 (A’BC)

C = Treset’ (A’BC)

1. **LEDs:**

Led1 = SW1 (A’BC)

Led2 = SW2 (A’BC)

Led3 = SW3 (A’BC)

Led4 = SW4 (A’BC)

1. **Flag:**

F1 = 1’2’3’4 + 1’2’34’

F2 = 1’2’3’4 + 1’23’4’

-Use a flipflop to store the values with an enabler of (12'3'4' + 1'23'4' + 1'2'34' + 1'2'3'4) (since they will disappear when you change the switches)

Table 5 - Sequence 2 transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sequence 2 | | | | | | | | | | | | |
| State | |  | **Inputs** | | | | **Next state** | | | **Outputs** | | | |
| ABC | | Treset | SW1 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 100 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

* **Equations:**

1. **States:**

A = Treset’(AB’C’)

B = Treset (AB’C’)

C = 0

1. **LEDs:**

Led1 = 4(AB’C’)

Led2 = 1(AB’C’)

Led3 = 2(AB’C’)

Led4 = 3(AB’C’)

Table 6 - Sequence 3 transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sequence 3 | | | | | | | | | | | | |
| State | |  | **Inputs** | | | | **Next state** | | | **Outputs** | | | |
| ABC | | Treset | SW11 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 101 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

* **Equations:**

1. **States:**

A = Treset’ (AB’C)

B = (Treset + 12 + 13 + 14 + 23 + 24 +34) (AB’C)

C = Treset’ (AB’C)

1. **LEDs:**

Led1 = 3(AB’C)

Led2 = 4(AB’C)

Led3 = 1(AB’C)

Led4 = 2(AB’C)

Table 7 - Sequence 4 transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sequence 4 | | | | | | | | | | | | |
| State | |  | **Inputs** | | | | **Next state** | | | **Outputs** | | | |
| ABC | | Treset | SW1 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 110 | | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

* **Equations:**

1. **States:**

A = Treset’ (ABC’)

B = 1 (ABC’)

C = 0

1. **LEDs:**

Led1 = 4(ABC’)

Led2 = 3(ABC’)

Led3 = 2(ABC’)

Led4 = 1(ABC’)

Table 8 - Special trick transitions and outputs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Special Trick | | | | | | | | | | | | |
| State | |  | **Inputs** | | | | **Next state** | | | **Outputs** | | | |
| ABC | | Treset | SW1 | SW2 | SW3 | SW4 | A | B | C | Led1 | Led2 | Led3 | Led4 |
| 111 | | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

* **Equations:**

1. **States:**

A = C = Treset’ (ABC)

B = 1 (ABC)

1. **LEDs:**

Led1 = 0

Led2 = 4(ABC)

Led3 = 1(ABC)

Led4 = 2(ABC)

**Treset Logic:**

Input I is = (1 + 2 + 3 + 4)' (A'B'C' + A'B'C + A'BC')'

Table 9 - Treset Logic

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Present State | | **Input** | **Next State** | | **Output** |
| A | B | I | A | B | Y |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

Table 10 - Deriving the equation for A

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BI**  **A** | 00 | 01 | 11 | 10 |
| **0** | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 1 | 0 |

We derive the equation for A:

Table 11 - Deriving the equation for B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BI**  **A** | 00 | 01 | 11 | 10 |
| **0** | 0 | 1 | 0 | 0 |
| **1** | 0 | 1 | 1 | 0 |

We derive the equation for B:

We derive the equation for Y:

Table 12 - Display Decoder Logic

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | Display | | | | | | |
| A | **B** | **C** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| 0 | 0 | 0 | X | X | X | X | X | X | X |
| 0 | 0 | 1 | X | X | X | X | X | X | X |
| 0 | 1 | 0 | X | X | X | X | X | X | X |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | X | X | X | X | X | X | X |

a=d

b=1

c=e’

Table 13 - K-map 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Output a | | | |
|  | **00** | **01** | **11** | **10** |
| **0**  **A** | x | x | 0 | x |
| **1**  **BC** | 1 | 1 | x | 0 |

From this Karnaugh Map, we get the following equation for Output a and d:

a= d = B’

b = 1

Table 14 - K-map 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Output c | | | |
|  | **00** | **01** | **11** | **10** |
| **0**  **A** | x | X | 1 | x |
| **1**  **BC** | 0 | 1 | X | 1 |

From this Karnaugh Map,

we get the following equation for Output c :

c =  B+C

Table 15 - K-map 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Output f | | | |
|  | **00** | **01** | **11** | **10** |
| **0** | x | x | 0 | x |
| **1** | 0 | 0 | X | 1 |

From this Karnaugh Map, we get the following equation for Output f:

f = BC’

**A**

**BC**

Table 16 - K-map 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Output e | | | |
|  | **00** | **01** | **11** | **10** |
| **0**  **A** | x | X | 0 | X |
| **1**  **BC** | 1 | 0 | **X** | **0** |

From this Karnaugh Map, we get the following equation for Output e:

e = (B+C)’ =B’C’

Table 17 - K-map 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Output g | | | |
|  | **00** | **01** | **11** | **10** |
| **0** | x | x | 0 | x |
| **1** | 1 | 1 | X | 1 |

From this Karnaugh Map, we get the following equation for Output g:

g = A

**A**

**BC**

## **NE 555- Research**

The NE555 timer is a widely used integrated circuit (IC) in electronics, renowned for its versatility in generating precise time delays and oscillations. Introduced by Signetics in 1972 and designed by Hans R. Camenzind, the NE555 has become a staple in both educational and industrial applications due to its reliability and simplicity.

**Some of its features include:**

* Operating Voltage: 4.5V to 15V
* Output Current: Up to 200 mA
* Temperature Range: 0°C to +70°C for NE555; -55°C to +125°C for SE555
* Package Types: Available in 8-pin DIP and other configurations
* Internal Components: Comprises 25 transistors, 2 diodes, and 15 resistors

**Operating Modes**

1. **Astable Mode (Oscillator):**

* Generates a continuous square wave without external triggering.
* Applicants: LED flashers, tone generation, clock pulses.
* Frequency formula:
* Duty Cycle:

A diagram of a timer

AI-generated content may be incorrect.

Figure 2 - Astable mode

1. **Monostable Mode (one-shot pulse):**

* Produces a single output pulse in response to an external trigger.
* Applications: Timers, pulse generation, switch debouncing.
* Pulse Width Formula:

1. **Bistable Mode (Flip-Flop):**

* Operates as a flip-flop, toggling between two stable states.
* Applications: Toggle switches, memory storage elements.

**Common Applications**

* Pulse Width Modulation (PWM): Controlling motor speeds and LED brightness.
* Timers and Delays: Creating precise time delays in circuits.
* Oscillators: Generating clock signals for digital circuits.
* Tone Generation: Producing audio tones in alarms and sound effects.
* Sequential Timers: Managing sequences in automation systems.

**Internal Structure**

The NE555's internal architecture includes:

* Three resistors form a voltage divider.
* Two voltage comparators.
* An SR flip-flop.
* A discharge transistor.
* An output stage capable of sourcing or sinking current.

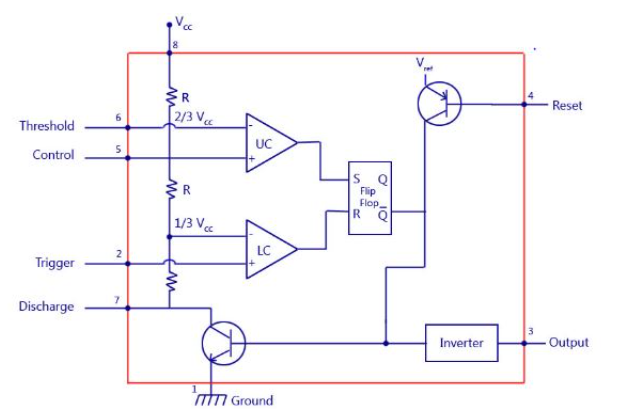


Figure 3 - Timer IC Internal Circuit

This configuration allows the NE555 to operate in various modes with minimal external components.

The following shows the pin configuration and purpose of each pin:

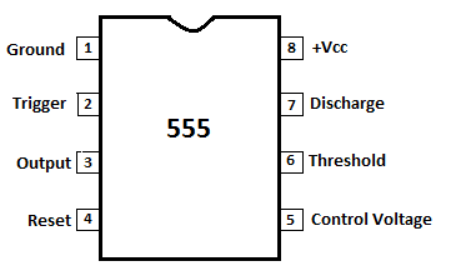


Figure 4 - 555 timer pin diagram

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5 - Pin, Names, and Purpose

## **BCD Decoder – Research**

A BCD to Decimal Decoder is a digital integrated circuit that converts a 4-bit binary-coded decimal (BCD) input into one of ten mutually exclusive outputs, corresponding to decimal digits 0 through 9.

**Key Features:**

* Input: 4-bit BCD code (inputs A, B, C, D).
* Output: 10 active-low outputs (Y0 to Y9), each representing a decimal digit.
* Invalid Input Handling: For BCD inputs representing decimal numbers 10 to 15, all outputs remain inactive (high).
* Output Configuration: Open-collector outputs allow wired-AND connections and drive higher voltage loads.
* Applications: Used in digital systems for decoding BCD inputs to drive displays, such as Nixie tubes or LED indicators

**Functional Description:**

When a valid BCD input (0000 to 1001) is applied:

* The corresponding output (Y0 to Y9) is driven low (active).
* All other outputs remain high (inactive).

For invalid BCD inputs (1010 to 1111):

* All outputs remain high, indicating no valid decimal digit.

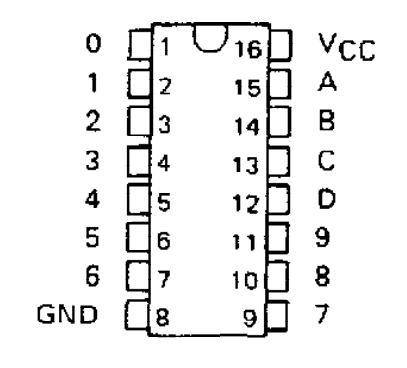


Figure 6 - BCD decoder pin configuration

# **Quartus Design and Analysis**

In this part, the logic-controlled board is implemented using Quartus software, focusing on digital logic development, simulation results, and a detailed system performance analysis.

The Quartus simulation for **Treset** was conducted to verify that the system correctly resets to its initial state under the defined conditions.

A white sheet of paper with purple lines

AI-generated content may be incorrect.

Figure 7 - Quartus design for the treeset

In the following, the Quartus image for **LED1** was simulated to ensure it activates accurately based on the logic conditions tied to the FSM state and inputs.

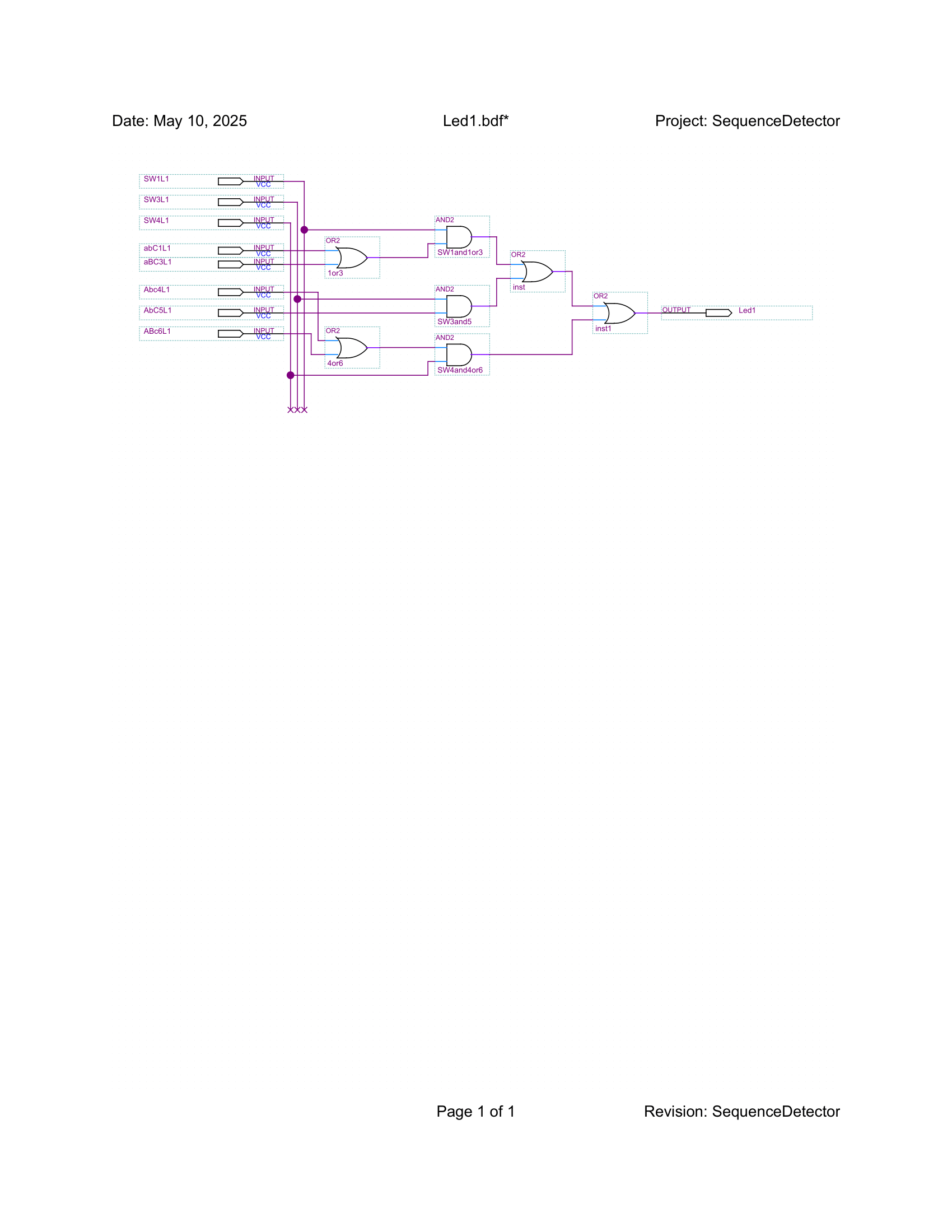


Figure 8 - Quartus design for LED1

In the following, the Quartus design and simulation for **LED2** confirm its proper behavior in response to specific state transitions and input combinations.

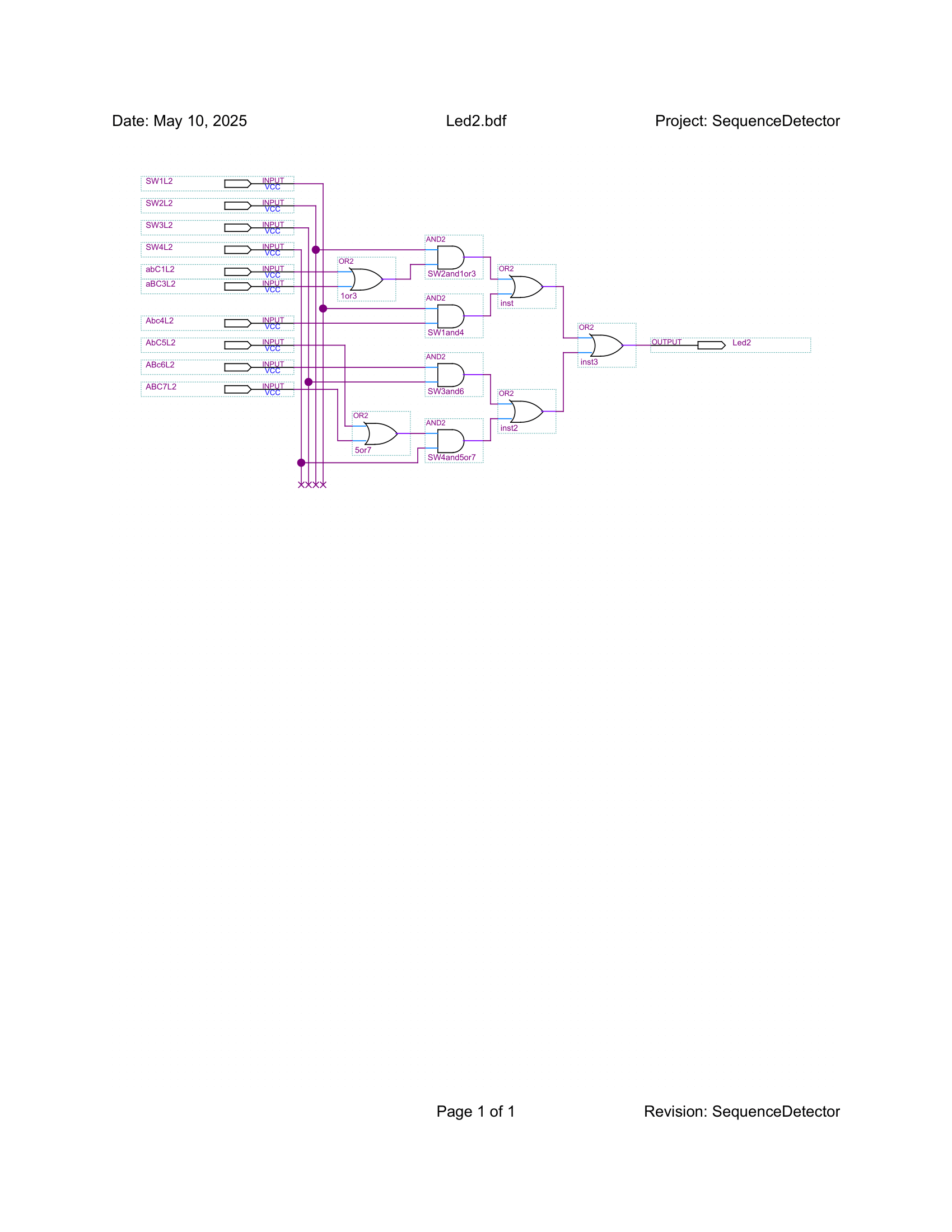


Figure 9 - Quartus design for LED2

In the following, the Quartus output for **LED3** was verified to activate during its corresponding logical state within the FSM.

A diagram of a circuit

AI-generated content may be incorrect.

Figure 10 - Quartus design for LED3

In the following, the Quartus waveform for **LED4** was generated to validate its function according to the assigned Boolean expression.

A diagram of a computer circuit

AI-generated content may be incorrect.

Figure 11 - Quartus design for LED4

In the following, the **sequence detector** logic was simulated in Quartus to confirm that it correctly identifies the predefined sequence of inputs.

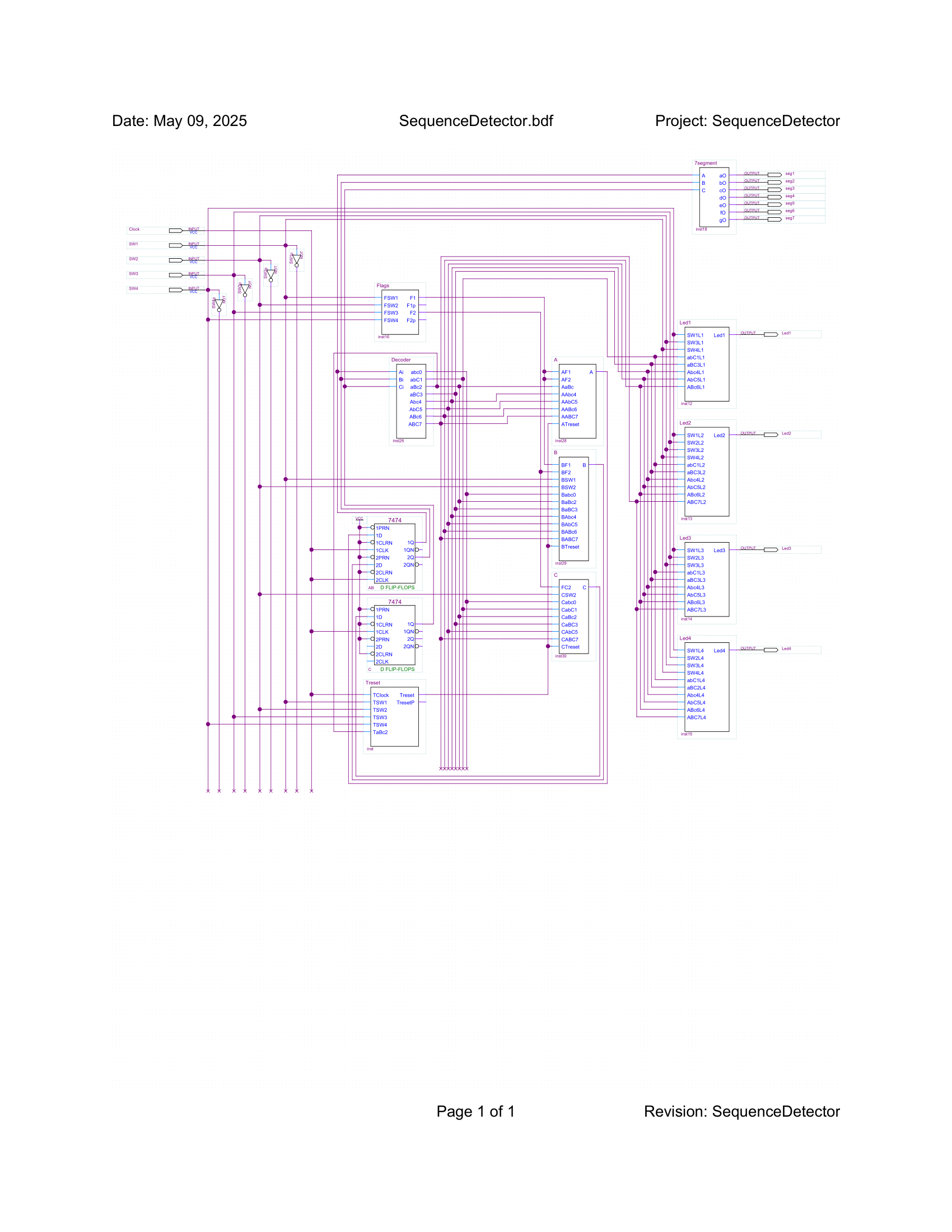


Figure 12 - Quartus design for the sequence detector

In the following, the Quartus simulation for **flags F1 and F2** demonstrates how these signals respond to specific switch combinations during Sequence 1.

A diagram of a circuit

AI-generated content may be incorrect.

Figure 13 - Quartus design for the Flags

In the following, the Quartus implementation of the **BCD decoder** was simulated to verify that each valid BCD input activates the corresponding output line.

A diagram of a computer circuit

AI-generated content may be incorrect.

Figure 14 - Quartus design for the Decoder

In the following, the state bits **A, B, and C** were simulated in Quartus to confirm correct state transitions and proper flip-flop behavior.

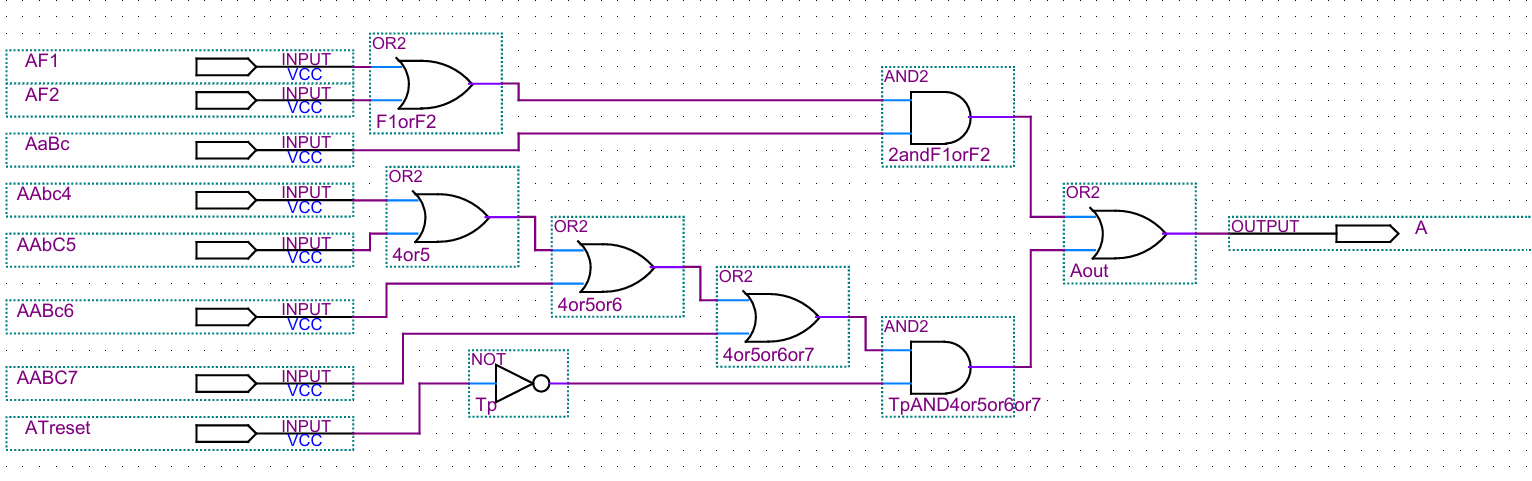


Figure 15 - Quartus design for A

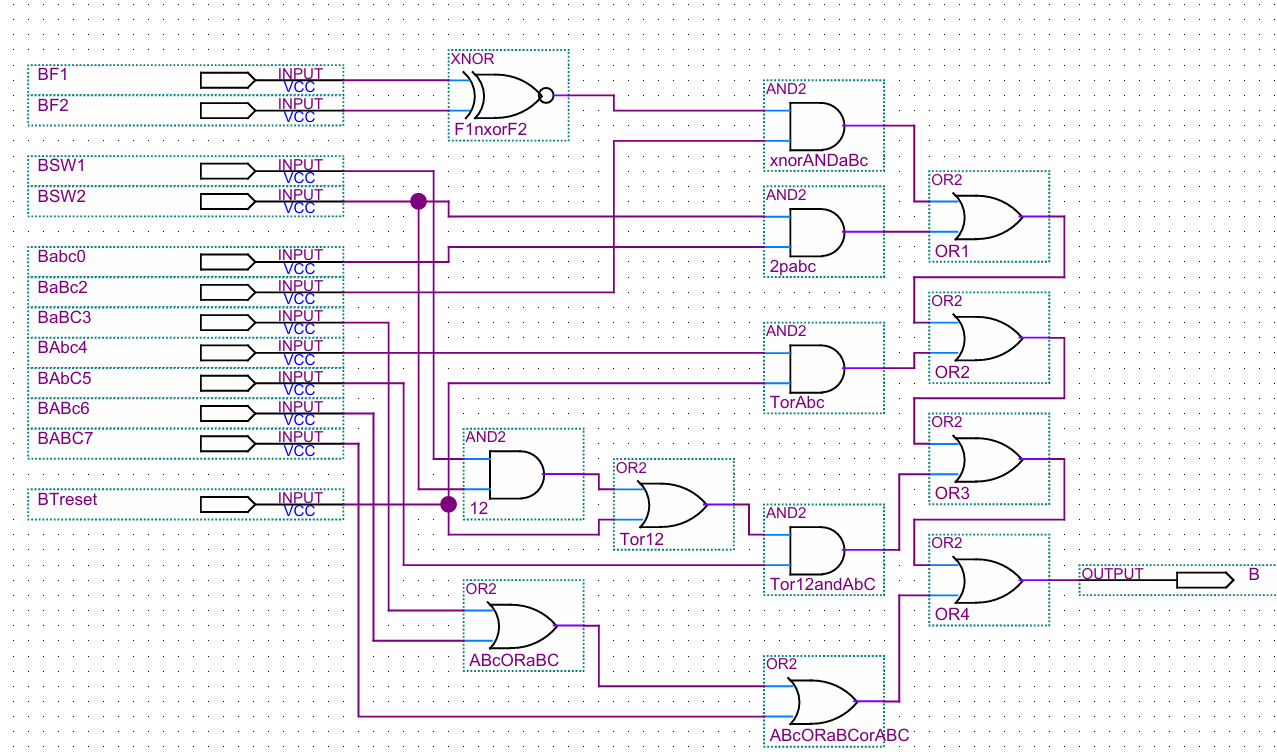


Figure 16 – Quartus design for B

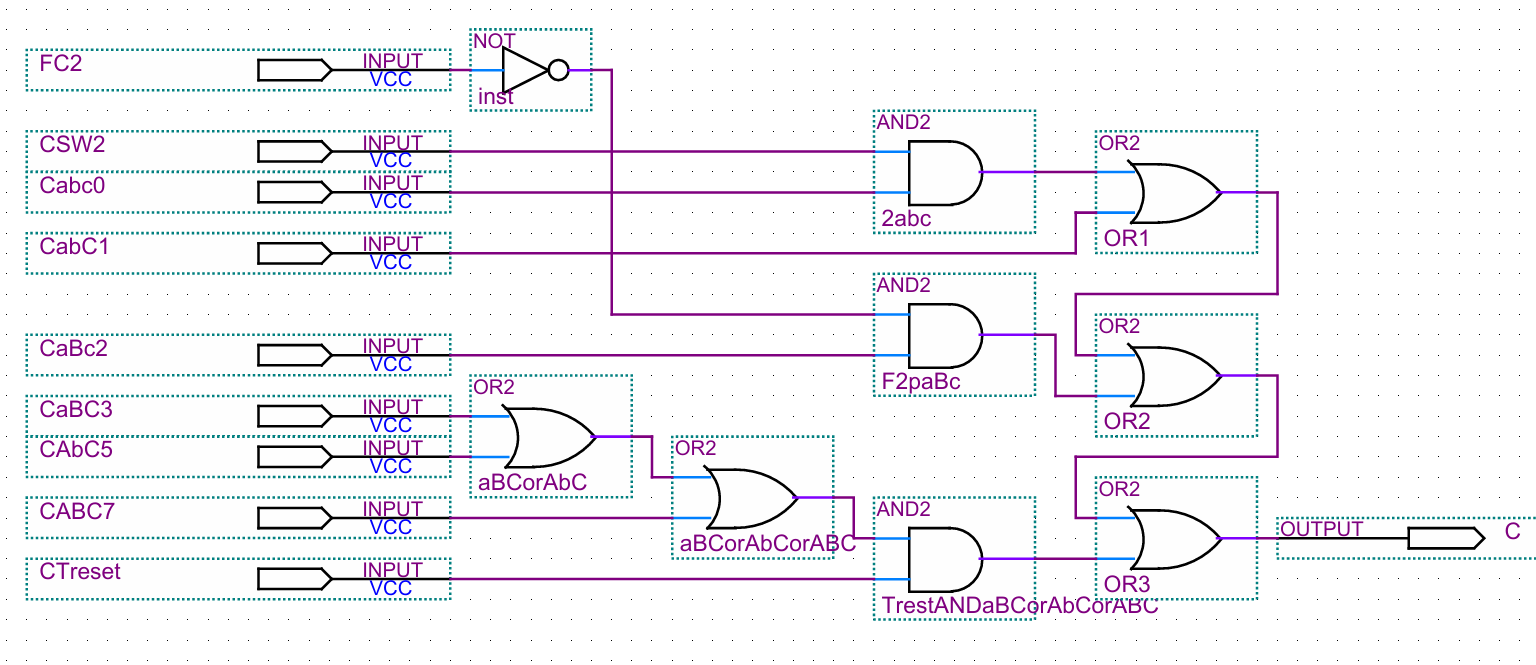


Figure 17 – Quartus design for C

In the following, the Quartus simulation for the **7-segment display** demonstrates the segment outputs derived from the simplified logic based on state bits.

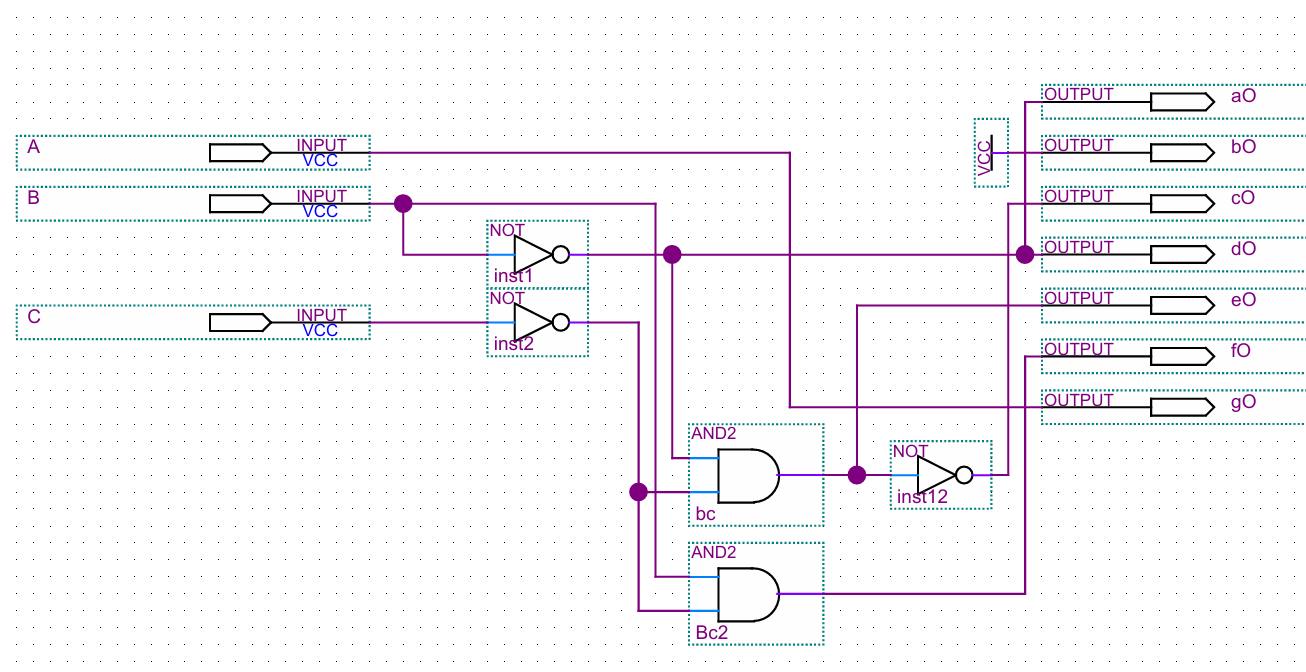


Figure 18 - Quartus design for the 7-segment display

To analyze our FSM-based system, we implemented and tested each module within Altera Quartus II, ensuring that all logic derived from truth tables and Boolean equations functioned correctly. The complete system was built around seven primary states, with transitions defined by the Treset signal, switch inputs, and flag conditions. In Quartus, we simulated Treset to verify that the system consistently returned to the BOOT state regardless of the current state, confirming reliable initialization.

Each LED output (LED1–LED4) was tested across various sequences to ensure activation only under their intended logical conditions. These outputs corresponded precisely with state-dependent Boolean expressions, such as Led1 = SW3 (AB'C) or Led2 = SW2 (ABC'), and were validated via waveform inspection during simulation.

The Sequence Detector logic was a core feature of our design. It tracked specific input patterns across states and was verified in Quartus by feeding in test patterns through the vector waveform file. Proper sequencing resulted in correct state transitions and flag activations.

Flag signals (F1, F2) were simulated in conditions defined by multiple switch combinations, such as F1 = 1’2’3’4 + 1’2’34’. Quartus simulations confirmed that these signals only activated during the defined valid sequences, helping control state logic in Sequence 1.

The BCD Decoder and 7-segment Display logic were tested by observing how the FSM states (A, B, C) translated into display outputs. The logic for segments was implemented using simplified K-map expressions and verified in simulation. The decoder consistently reflected the correct decimal or visual state output based on binary state values.

State bits A, B, and C were driven by next-state equations unique to each FSM sequence and were confirmed via simulation to toggle correctly in response to each input change. Overall, Quartus simulation validated the entire design flow from reset to state transition, output logic, flag triggering, and display decoding, confirming that the FSM operated exactly as intended in all scenarios.

# **Breadboard Design and Analysis**

In this part, the system is assembled and tested on a breadboard, emphasizing hardware layout, component integration, and evaluating its behavior under real-world conditions.

The breadboard played a critical role in the prototyping and validation of our logic-based LED control circuit. This section details the component setup, logic behavior, IC chip usage, and testing methodology as implemented during the project.

**Component Setup and Configuration**

We used a standard full-size breadboard to construct our circuit, allowing us to conveniently plug and test all required components without soldering. Our setup included:

* **LEDs** (Light Emitting Diodes) to visually indicate logic outputs.
* **Switches** to serve as logic inputs (representing binary 1 when pressed, and binary 0 when open).
* **Resistors**: Primarily used 220Ω resistors in series with LEDs to limit current and prevent burning them out. Additionally, 10kΩ pull-down resistors were used to prevent floating input states on ICs.
* **IC chips**:
  + 7408 (Quad 2-input AND gate)
  + 7432 (Quad 2-input OR gate)
  + 7404 (Hex Inverter NOT gate)
  + 7400 (Quad 2-input NAND gate)
  + 7402 (Quad 2-input NOR gate)

Each IC was powered with +5V (Vcc) and grounded properly on the breadboard. The switches were connected to the IC inputs through pull-down resistors to ensure the inputs default to logic LOW (0) when the switches are not pressed.

**Logical Behavior and Circuit Design**

The circuit was designed to implement combinational logic expressions where specific LED outputs would light up only for certain combinations of switch inputs. For example, in one part of the circuit, a truth table was created with three inputs (A, B, C), and an output Y based on the logic function:

**Y = (A AND B) OR (NOT C)**

Using ICs, the above function was realized as follows:

* The **AND** gate (7408) combined inputs A and B:
* The **NOT** gate (7404) inverted input C.
* The **OR** gate (7432) combined the two results.

This logical output Y was connected through a resistor to an LED. The LED lit up when the output Y was HIGH (1), confirming the circuit’s logic accuracy.

In other sections, more complex expressions using **NAND** and **NOR** gates were implemented to achieve the same functionality but using universal gates. For example:

Y = NOT (A AND B) was implemented directly using a 7400 NAND gate:

Similarly, the NOR gate (7402) was used to realize it.  
By converting standard logic expressions into NAND- or NOR-only versions, we demonstrated gate equivalency and optimized design approaches.

**Circuit Testing and Observations**

After wiring the components and ICs, we conducted systematic tests for all switch combinations. For each logic function, a corresponding **truth table** was verified manually.

We used jumper wires to simulate logic inputs and checked the output status using the LEDs. The LEDs correctly responded according to their defined Boolean expressions, which validated the IC configurations and logic gate wiring.

Additionally, we tested edge cases such as all switches being OFF (0 0 0) or all ON (1 1 1) and confirmed that the outputs still matched the logic tables.

# **Financial Study**

In this part, the financial aspects of the project are evaluated, including the total cost of components and materials required for building the logic-controlled board.

The financial aspect of this project focused on sourcing and acquiring the electronic components required for the implementation and testing of our design. All components were purchased from a local supplier, and efforts were made to minimize costs while maintaining quality by selecting reliable and affordable integrated circuits and accessories.

Below is a breakdown of the components and their corresponding costs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Items** | **Quantity** | **Unit Price** | **Cost** |
| **DIP switch 4 poles** | 1 | 0.20 | 0.20 |
| **IC 7408 2-input AND Gate** | 12 | 0.30 | 3.60 |
| **IC 7432 2-input OR Gate** | 12 | 0.45 | 5.40 |
| **IC 7486 2-input X-OR Gate** | 2 | 0.50 | 1.00 |
| **IC LM555 – Timer** | 1 | 0.25 | 0.25 |
| **IC 7404 Hex Schmitt Inverter** | 8 | 0.30 | 2.40 |
| **IC 7474 – Dual D Flip-Flop** | 5 | 0.50 | 2.50 |
| **Breadboard Double-sided PCB** | 2 | 2.50 | 5.00 |
| **IC socket** | 45 | 0.10 | 4.50 |
| **IC 7447 – 7-segment decoders** | 1 | 0.75 | 0.75 |
| **IC 74138 – 3 to 8 Line Decoder** | 2 | 0.5 | 1.00 |
| **Total** |  |  | $26.60 |

It is worth mentioning that some components were purchased in slightly larger quantities than strictly necessary. This precaution was taken to account for possible component damage during soldering, accidental burning, or loss during handling. Ensuring a small surplus allowed us to avoid delays and complete the project without interruptions due to part shortages.

The total expenditure of **USD 26.70** covers all required parts and spares, making the project both **cost-effective** and **reliable** in its execution.

# **Delay Calculation**

In this part, timing analysis is conducted to calculate the delays within the system, ensuring accurate synchronization between switch inputs and LED outputs.

**555 Timer Delay Calculation**

The logic-controlled board uses a **555 timer in astable mode** to provide a continuous square-wave clock signal for the finite state machine (FSM). This clock governs the timing of state transitions and LED updates.

The period T of the 555 timer in astable mode is given by:

Where:

* R1 = 1kΩ
* R2 = 1kΩ
* C = 470μF

Substituting the values:

Thus, the timer generates a clock pulse **approximately every 0.98 seconds**. This 1-second timing interval ensures that:

* The system updates at a human-perceivable rate,
* Logic and propagation delays (on the order of nanoseconds) are negligible,
* State changes and LED activations remain stable and synchronized.

In this system, several logic gates and memory elements are used to process input signals and drive outputs. While these components introduce slight propagation delays, their impact is negligible compared to the 1-second clock interval generated by the 555 astable timer.

Table 18 Delay Time for Components

| **Component** | **Description** | **Typical Propagation Delay** |
| --- | --- | --- |
| **7408** | 2-input AND gate | ~10 ns |
| **7432** | 2-input OR gate | ~10 ns |
| **7486** | 2-input XOR gate | ~10 ns |
| **7404** | Inverter (NOT gate) | ~10 ns |
| **74138** | 3-to-8 Line Decoder | ~25 ns |
| **7474** | Dual D Flip-Flop | ~20 ns |

# **Power Consumption Analysis**

In this part, the system’s power usage is analyzed by estimating the energy consumption of all active components during operation.

The power consumption of the logic-controlled board is estimated based on the typical current draw of each digital IC and passive component used in the system. While the project is not power-critical, estimating consumption helps assess overall efficiency and battery requirements.

**1. Voltage Supply**

The system operates at a standard **+5V DC**, which is compatible with all 74-series TTL logic chips.

2. **Typical Current Consumption per IC**

Table - Components' current consumption

| **Component** | **Quantity** | **Typical Current per IC** | **Total Current** |
| --- | --- | --- | --- |
| **7408 (AND gate)** | 12 | ~4 mA | ~48 mA |
| **7432 (OR gate)** | 12 | ~4 mA | ~48 mA |
| **7486 (XOR gate)** | 2 | ~4 mA | ~8 mA |
| **7404 (NOT gate)** | 8 | ~4 mA | ~32 mA |
| **74138 (3-to-8 decoder)** | 2 | ~10 mA | ~20 mA |
| **7474 (D Flip-Flop)** | 5 | ~8 mA | ~40 mA |
| **LM555 (Astable Timer)** | 1 | ~10 mA | ~10 mA |
| **LEDs (estimated)** | 4 | ~10–15 mA each (with resistors) | ~40–60 mA |
| **7-Segment Display** | 1 | ~30 mA (varies with active segments) | ~30 mA |

**3. Estimated Total Current**

Summing up the estimated draws:

This is a **conservative estimate**, assuming all components are active simultaneously, which may not always be the case in practical operation.

4. **Power Consumption**

Using the power formula:

So, the system consumes approximately **1.4W** under full activity. This level of power is acceptable for battery powered operation.

# **Problems Faced During the Project**

Throughout the development of the Logic-Controlled Board, the team encountered several technical and practical challenges that required careful troubleshooting and collaborative problem-solving. These issues ranged from hardware integration difficulties to design logic adjustments and software simulation mismatches.

**1. Incorrect Wiring on Breadboard**

One of the most common challenges was managing the large number of ICs and connections on the breadboard. At times, small wiring mistakes—such as misplacing jumpers or misaligning IC pins—led to non-functional circuits or unexpected behaviors. Debugging these errors required systematic pin-by-pin checks using a multimeter and careful review of the schematic.

**2. LED Overload Without Resistors**

Some LEDs initially burned out or were too bright because they were connected without series resistors. This mistake caused excessive current draw and disrupted voltage levels for other components. It was corrected by adding 330 Ω resistors in series with each LED to limit current to safe levels

**3. Forgetting to Ground the Switches**

During early testing, the team forgot to connect one or more switch terminals to ground. As a result, the switches provided floating inputs to the logic gates, causing unpredictable or inconsistent behavior. This was resolved by properly grounding one side of each switch and ensuring pull-down resistors were in place where needed.

**4. Unstable Timer Output**

Initially, the 555 timer in astable mode did not consistently produce a clean 1-second pulse due to poor capacitor quality or imprecise resistor values. This led to timing instability in the FSM. After replacing the capacitor with a reliable 470μF electrolytic and verifying the resistor values with a multimeter, the timer stabilized.

**5.** **Unwanted Looping:** When all switches remain OFF for 4 seconds, the T-reset signal activates and sends the system to the sequence detector. However, since the OFF state persists, the system immediately loops back, causing an unwanted cycle. To fix this, we disabled T-reset while in the sequence detector by adding logic: **T-reset = T-reset · (NOT (sequence detector)) = T-reset · (010)'**, preventing re-entry and ensuring proper progression.

**6.** **Lost Flag Information:** The system relies on knowing which switch was last ON to determine the correct sequence (with 2-bit flags: 00 for 1000, 01 for 0100, 10 for 0010, 11 for 0001). But since the detector only activates when all switches are OFF, this info would be lost. We solved this by using two D flip-flops to store the flag, enabled only when one switch is ON. The enable condition is: **Enable = 1′2′3′4 + 12′3′4′ + 1′23′4′ + 1′2′34**, ensuring the correct flag is latched before transitioning.

# **Advantages of our Design**

The Logic-Controlled Board presents several advantages that distinguish it from static or conventional switch-lamp configurations. The integration of finite state machines, adaptive switching logic, and interactive features creates a responsive and educational system with the following benefits:

**1. Minimal Power Consumption**

Despite its complexity, the system is power-efficient. It operates on a 5V supply and consumes less than 1.5W under full load, making it suitable for USB or battery-powered setups.

**2. Clear Visual Feedback**

The use of **LEDs** and a **7-segment display** provides immediate and intuitive visual feedback to the user. This helps in understanding which sequence is active and how the system responds to switch activity.

**3. Low-Cost Implementation**

With a total cost of approximately $26, our design demonstrates how a fully interactive digital logic project can be implemented on a tight budget, making it accessible for students and hobbyists.

**4. Manual Lock/Unlock Mechanism**

We implemented a simple yet effective physical method to lock and unlock the dynamic behavior (by toggling Switch 2 during power cycles). This eliminated the need for additional circuitry or memory components and reduced complexity

# **Conclusion**

In conclusion, the Logic-Controlled Board project successfully demonstrated the integration of digital logic design, finite state machines, and user interaction to create an engaging and dynamic system. Through careful planning, paper design, simulation in Quartus, and real-world implementation on a breadboard, the project achieved all functional objectives, including adaptive lamp sequencing, cap-based switch behavior, and lock/unlock mechanisms. Additional analyses, such as delay calculations, power consumption, and financial assessment, provided a comprehensive understanding of the system’s performance and feasibility. Overall, this project not only reinforced key theoretical concepts but also highlighted their practical applications in building interactive digital systems.

# **References**

alldatasheet.com. (n.d.-b). *NE555 Datasheet(PDF)*. Texas Instruments. <https://www.alldatasheet.com/datasheet-pdf/pdf/1082493/TI1/NE555.html>

EFY Bureau. (2024a, September 26). *555 Timer IC Pin diagram, Circuit, Working, Datasheet, Modes*. Electronics for You. <https://www.electronicsforu.com/technology-trends/learn-electronics/555-timer-working-specifications>

Winter, Z. (2023, February 14). IC 555 Timer uses and applications. *EEWeb*. <https://www.eeweb.com/typical-555-ic-circuit-applications/>

<https://www.ti.com/lit/ds/symlink/ne555.pdf>

<https://www.ti.com/lit/ds/symlink/sn5445.pdf>