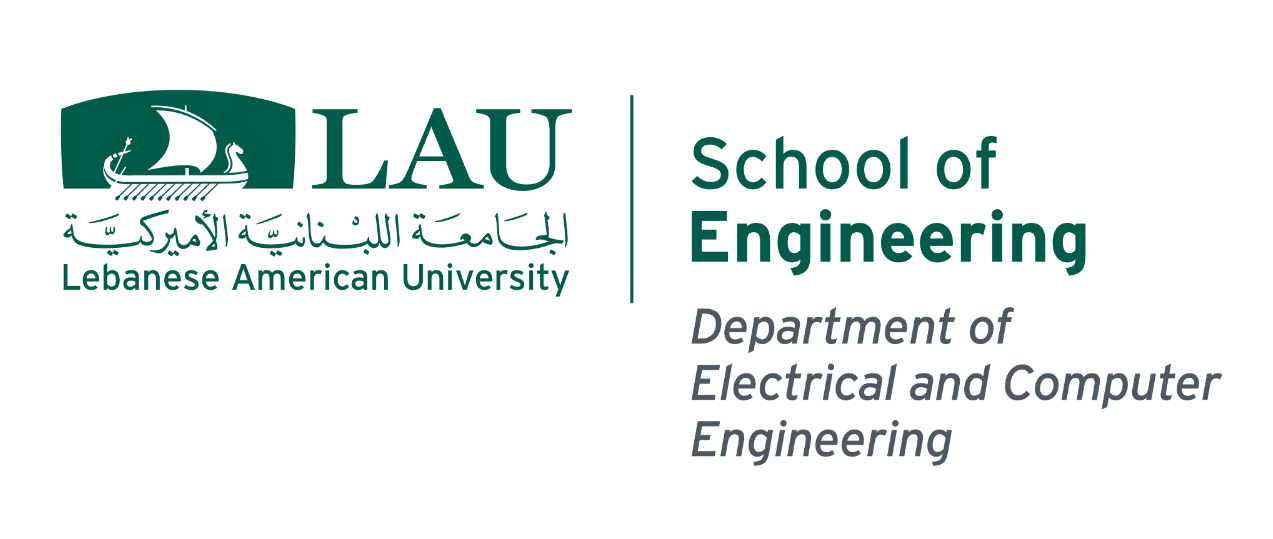
**Lebanese American University**



***COE 322 – Logic Design Lab***

***Final Project Submission***

Abbass Aoun 202307321

Kassem Smily 202306845

Mohamad Jouni 202306714

14 – 05 – 2025

**Table of Contents**

[Table of Figures 3](#_Toc198101729)

[Table of Tables 4](#_Toc198101730)

[Absract 5](#_Toc198101731)

[Introduction 6](#_Toc198101732)

[Equipment Needed 7](#_Toc198101733)

[Analysis 8](#_Toc198101734)

[Quartus Design 9](#_Toc198101735)

[Finite State Machine for sequence management 9](#_Toc198101736)

[Breadboard design and analysis 24](#_Toc198101737)

[Financial Study 25](#_Toc198101738)

[Delay analysis 27](#_Toc198101739)

[Power Consumption 28](#_Toc198101740)

[Problems Faced 29](#_Toc198101741)

[Key Design Points that Present Advantages Over Alternative Designs 30](#_Toc198101742)

[Conclusion 32](#_Toc198101743)

# Table of Figures

[Figure 1 - State diagram of the sequence manager 9](#_Toc198101744)

[Figure 2 - Truth table of the sequence manager 10](#_Toc198101745)

[Figure 3 - Example truth table used to determine mux input 10](#_Toc198101746)

[Figure 4 - Quartus sequence manager circuit 11](#_Toc198101747)

[Figure 5 - The locked handler circuit 12](#_Toc198101748)

[Figure 6 - The trick handler circuit 12](#_Toc198101749)

[Figure 7 - Last off detector Quartus circuit 13](#_Toc198101750)

[Figure 8 - Whole project 14](#_Toc198101751)

[Figure 9 - The switch memory binding block diagram 15](#_Toc198101752)

[Figure 10 - The three D-flip flops for each switch 15](#_Toc198101753)

[Figure 11 - the technique to handle sequence 2 and 3 17](#_Toc198101754)

[Figure 12 - Handling sequence 4 18](#_Toc198101755)

[Figure 13 - The decoders and ensuring that the led activates properly 18](#_Toc198101756)

[Figure 14 - Every 2 Switches being Nanded and inputted as selectors for every decoder 19](#_Toc198101757)

[Figure 15 - Entire Quartus Circuit 20](#_Toc198101758)

[Figure 16 - Quartus output part 1 21](#_Toc198101759)

[Figure 17 - Quartus output part 2 21](#_Toc198101760)

[Figure 18 - Quartus output part 3 22](#_Toc198101761)

[Figure 19 - Qaurtus output part 4 22](#_Toc198101762)

[Figure 20 - Quarus output part 5 22](#_Toc198101763)

[Figure 21 - Quartus output locked 23](#_Toc198101764)

[Figure 22 - Breadboard aftermat 24](#_Toc198101765)

[Figure 23 - Potential optimization 25](#_Toc198101766)

[Figure 24 - Last off detector iterations 29](#_Toc198101767)

# Table of Tables

[Table 1 - Prices and quantities of the components used 24](#_Toc198101770)

[Table 2 - Power analysis 27](#_Toc198101771)

# Introduction

Introduction: In this project , we implemented Logic-Controlled Board , which consists of four switches and four LEDs, this project is sequence dependent , in other words for each last switch turned off we enter a new sequence , We first implement the truth table of the project and the state diagram which included the states locked , boot, sequence manager , the 4 sequences and a certain trich in sequence 3 , we first simulated our whole project on the Altera Quartus II .To manage complexity, we divider the project into block diagrams and connected them to each other using certain gates , then we brought our components and build each block diagram, and connected them to each other.

# Equipment Needed

In this project, we used:

Wires

Altera Quartus II software for simulation

74LS00

74LS02

74LS04

74LS08

74LS32

74LS86

74LS157

74LS151

74LS175

74LS25

74LS11

74LS20

74LS139

74LS93

74LS83

Seven Segment Display

220 Ohm Resistors

Pf Capacitors

LEDs

5 Breadboards

# Analysis

The project focuses on designing a Logic-Controlled Board that dynamically changes the LED activation sequence based on the last switch turned off. The design involves the integration of combinational logic, sequential circuits, and finite state machines (FSM).

**Functional Requirements:**

* Track which switch was last turned off.
* Update the lamp activation sequence dynamically.
* Handle cap removal for special tricks.
* Display the active sequence on a 7-segment display.
* Ensure system reset after specific conditions.

**Inputs:**

* SW1–SW4 (toggle switches)
* Treset (reset timer signal)

**Outputs:**

* LEDs (L1–L4)
* 7-Segment Display (sequence indicator)

**FSM Overview:**

* **Boot State**: Initialize system, check for locked mode.
* **Sequence States (1–4)**: Each representing a different lamp activation sequence.
* **Capless Switch State**: Disable switch functionality when cap is removed.
* **Locked State**: Direct mapping (1–1, 2–2, etc.) for reset conditions.

# Quartus Design

## Finite State Machine for sequence management

A finite state machine was used to design the transitions between the 4 sequences in addition to the additional states like boot, locked and trick. Note that for combinations of inputs that are not explicitly mentioned, no state change occurs. The arrows from a state to itself have been omitted to reduce clutter.

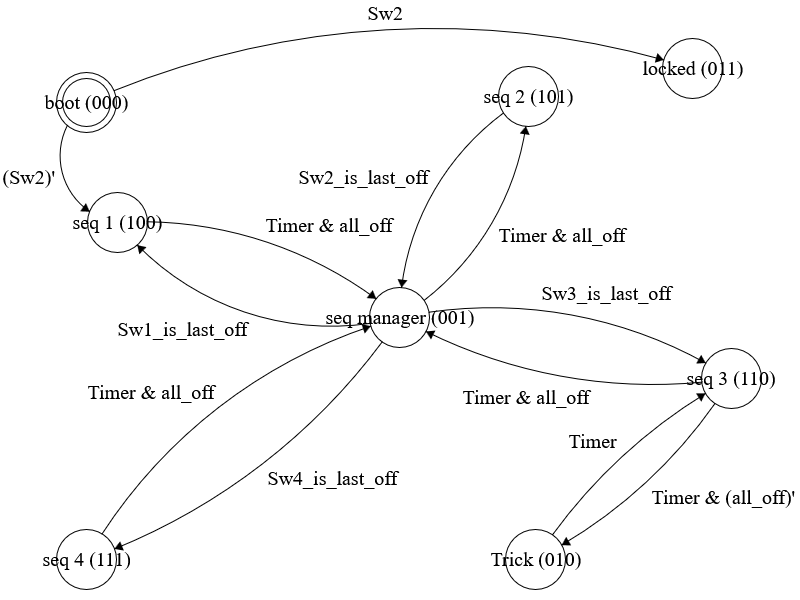


Figure 1 - State diagram of the sequence manager

We start at the boot state 000. From there, if Sw2 is turned on, we enter the locked state, otherwise we transition to sequence 1. From any of the sequences, if the timer delay is over and all the switches are turned off, we move to a seq manager state that routes us back to the appropriate sequence based on the last switch turned off. Additionally, from sequence 3, if the timer is over but not all the switches are off, we enter the trick state that we leave after the timer is over again.

The state diagram above may be used to construct the following truth table

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S2 | S1 | S0 | Sw2 | Timer | all\_off | lastoff1 | lastoff0 |  | S2+ | S1+ | S0+ |
| 0 | 0 | 0 | 1 | x | x | x | x |  | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | x | x | x | x |  | 1 | 0 | 0 |
| 0 | 1 | 1 | x | x | x | x | x |  | 0 | 1 | 1 |
| 1 | x | x | x | 1 | 1 | x | x |  | 0 | 0 | 1 |
| 1 | 0 | 0 | x | 0 | x | x | x |  | 1 | 0 | 0 |
| 1 | 0 | 0 | x | 1 | 0 | x | x |  | 1 | 0 | 0 |
| 1 | 0 | 1 | x | 0 | x | x | x |  | 1 | 0 | 1 |
| 1 | 0 | 1 | x | 1 | 0 | x | x |  | 1 | 0 | 1 |
| 1 | 1 | 1 | x | 0 | x | x | x |  | 1 | 1 | 1 |
| 1 | 1 | 1 | x | 1 | 0 | x | x |  | 1 | 1 | 1 |
| 1 | 1 | 0 | x | 1 | 0 | x | x |  | 0 | 1 | 0 |
| 1 | 1 | 0 | x | 0 | x | x | x |  | 1 | 1 | 0 |
| 0 | 1 | 0 | x | 0 | x | x | x |  | 0 | 1 | 0 |
| 0 | 1 | 0 | x | 1 | x | x | x |  | 1 | 1 | 0 |
| 0 | 0 | 1 | x | x | x | 0 | 0 |  | 1 | 0 | 0 |
| 0 | 0 | 1 | x | x | x | 0 | 1 |  | 1 | 0 | 1 |
| 0 | 0 | 1 | x | x | x | 1 | 0 |  | 1 | 1 | 0 |
| 0 | 0 | 1 | x | x | x | 1 | 1 |  | 1 | 1 | 1 |

Figure 2 - Truth table of the sequence manager

As you can see, each output combination depends on only 2 inputs in addition to the current state variables, so an 8 to 1 multiplexer will be used to implement each of the 3 outputs. The current state will be used as the select inputs of the mux then for each starting current state a small truth table may be used to find the boolean variable that should be connected to the mux input pin corresponding to each starting state number. For Example, starting from the boot state 000, the truth table would look like this.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sw2 |  | S2+ | S1+ | S0+ |
| 1 |  | 0 | 1 | 1 |
| 0 |  | 1 | 0 | 0 |

Figure 3 - Example truth table used to determine mux input

Using this truth table, we know that the input connected to the input pin 000 of the first mux (corresponding to S2+) should be (Sw2)’, and the value connected to the pins 000 of the other multiplexers should be Sw2. Using other similar truth tables for every input pin of the mux, we get the following circuit diagram.

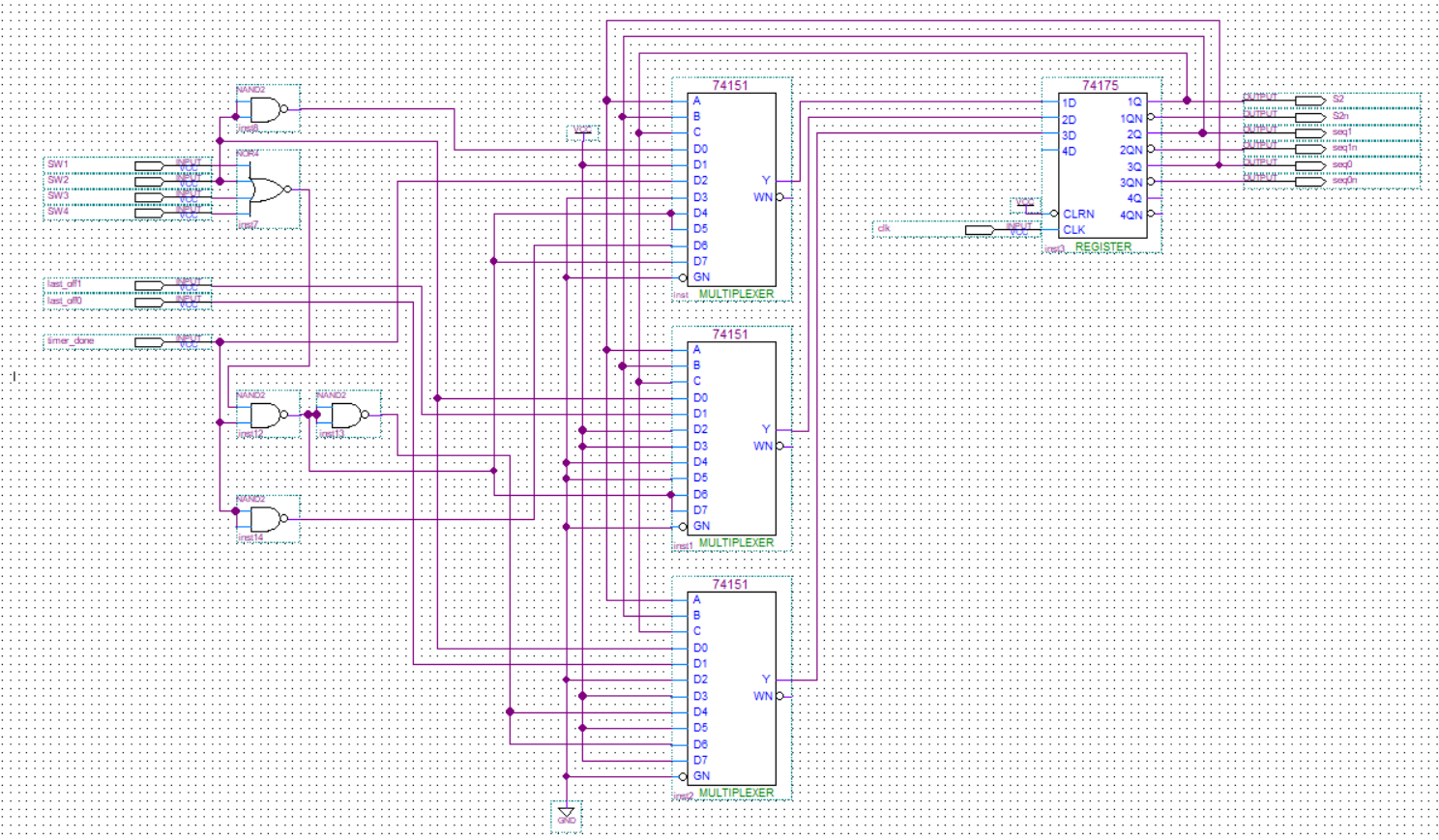


Figure 4 - Quartus sequence manager circuit

The key boolean variables used in the above circuit are Sw2, (Sw2)’, lastoff1, lastoff0 (represent the number (0-3) of the last switch turned off), timer, (timer)’, timer NAND alloff, and timer AND alloff. Also note that with the labelling of the states, we know that we are in sequence XY if the state variables are 1XY.

**Misc circuits**

This section will cover 2 small circuits that were used to complete the switchboard design.

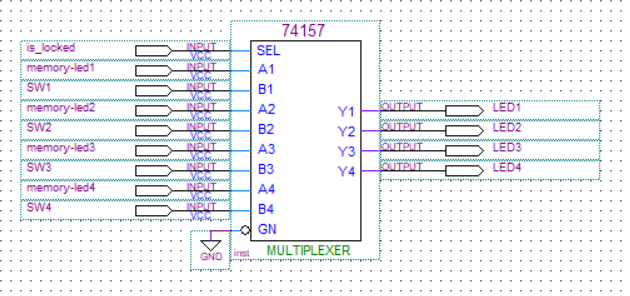


Figure 5 - The locked handler circuit

Using our finite state machine, we can transition between states and confirm that we are in a certain state by looking at our state variables. If the circuit is in the locked state, LEDs will no longer bind to switches as before instead each switch will turn on the LED facing it. We will use a quad 2 to 1 mux to handle this switching behavior. If the circuit is not locked, the output of the switch-led binding circuit will be fed to the actual LED. If the circuit is locked, the value of the switch facing the LED will be passed to the LED instead.

A diagram of a computer

Description automatically generated

Figure 6 - The trick handler circuit

The purpose of this stage is to actually prevent the last turned off switch from turning on an LED. This will be done by cutting off the switch signal of the switch involved in the trick before it even reaches the switch-led circuit. In this circuit, not\_last\_off\_X is an inverted almost one hot encoding of the last switch turned off (not\_last\_off\_X will be 1 if switch X is not the last to turn off). The And gates are used to cut off the switch signal which will pass if either the certain switch is not the last to turn off as indicated by not\_last\_off or if the circuit is not in the trick state.

**Last off detector**

A diagram of a computer circuit

Description automatically generated

Figure 7 - Last off detector Quartus circuit

Detecting which switch was the last to get turned off is a very important task for this project since we need this to determine which sequence to change to and which switch to deactivate in trick 3.

This circuit starts with a standard d flip flop into xor gate to generate a pulse on a switch change. The output of the 4 input or tells us whether any of the switches has changed.

Next, shift your focus to the next 2 chips (mux and dff). If no switch was changed the select input of the 2 to 1 mux is 0 and the value inside the d flip flops is not changed (D is connected to Q). However, if a switch is changed, the select input becomes 1, so the value stored in the D flip flop corresponding to each switch becomes whether this was the switch that was changed. The others are reset to 0. Hence, the 2nd 74175 chip stores whether each switch was the last changed or not.

For the final part of this circuit, ignore the final 74175 chip initially. For each switch, we determine not\_last\_changed OR switch\_on. This boolean value represents if the switch is not the last to turn off. The nand gates at the end form an encoder to represent the number of the switch last turned off. NAND gates are used instead of OR since the inputs are inverted.

The final 74175 chip is used as a buffer between the other parts of the circuit and the outputs to prevent the outputs form changing when we do not want them to change. In particular we do not want to change the outputs when we are in the trick state as that state depends on the output of this circuit to determine which switch to deactivate and we do not want the deactivated switch by the trick to change as we are performing the trick.

**Switch Led Memory Binding**

This circuit’s purpose is for the led to memorize which switch is turned on.It was designed using 13 ICs.

This circuit takes as an input the sequence, the switches and the reset (it resets when the sequence changes)

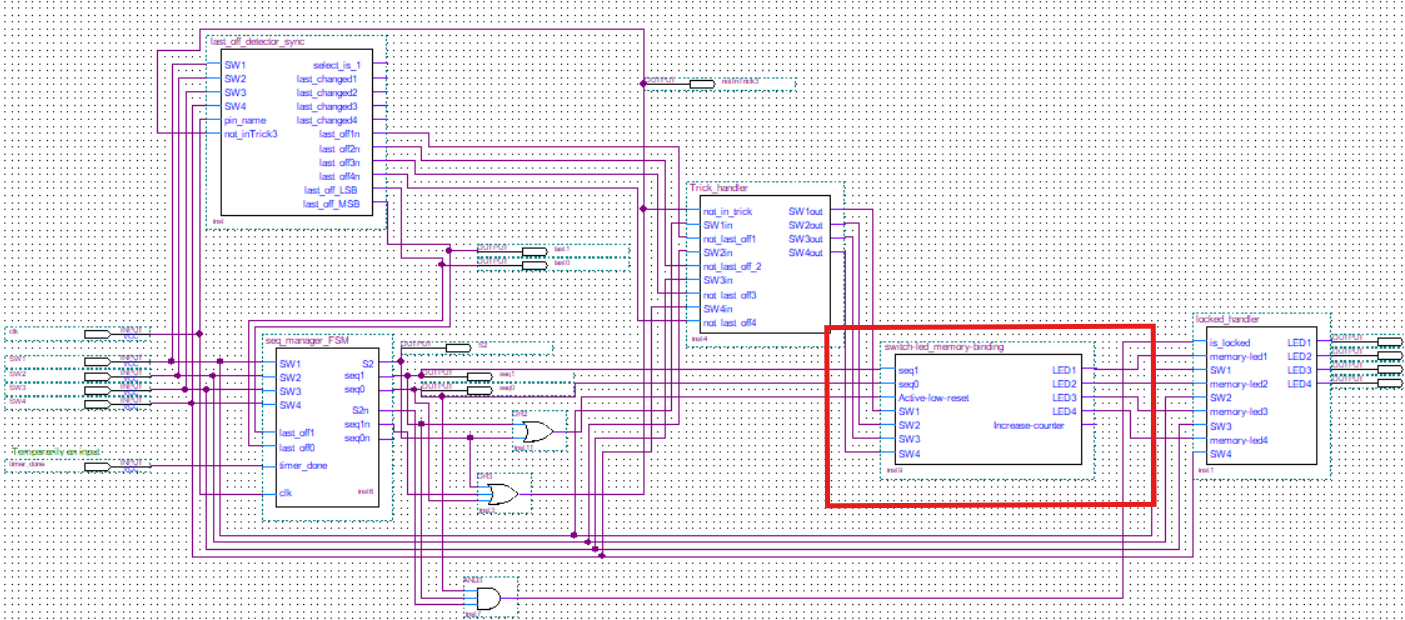
****

Figure 8 - Whole project

The figure below shows the full zoomed out snapshot of this configuration

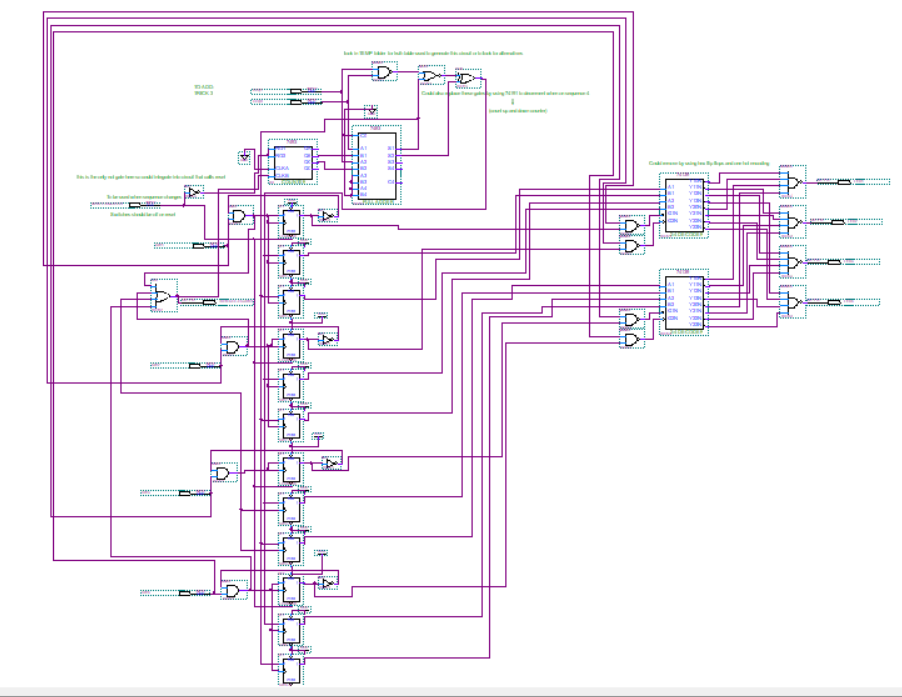


Figure 9 - The switch memory binding block diagram

Now every switch have three D-flip flops, the first one corresponds to whether the switch is connected to an led or not. The second and the third D-flip flops tell us with what this switch is connected to (i.e they give us numbers 00,01,10,11), a snapshot below will illustrate the case we mentioned above.

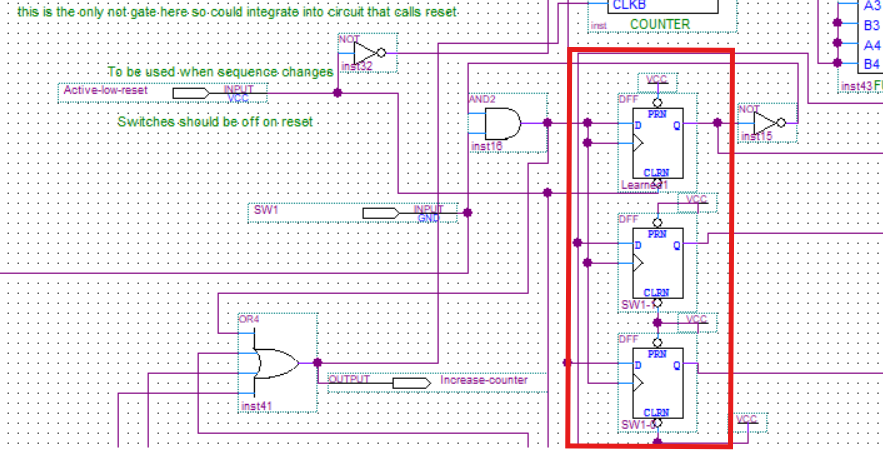


Figure 10 - The three D-flip flops for each switch

Note that for every switch there is the same configuration (Three D-flip flops) with each serving the same purpose mentioned above.

Now if we were in sequence 1, If we turn on a switch and has not been learned yet, we will make it learned (enter 1 for the learned). Simultaneously, the switch will be associated with the current LED number, which is determined by a counter, this counter starts counting from 00 and increase sequentially, for example in our case (00,01,10,11). Thus, each time a switch is learned, it is assigned the next Led number from the counter, ensuring that this mapping follows the right sequence.

In the sequence 2 or 3, we incorporate the usage of the full adder. Specifically, by adding the current value of the counter to the logic 1 using the adder. To elaborate more, in sequence 2, the design behaves like as in sequence 1, where the counter increments normally (00,01,10,11). However, here we are not using the output of the counter directly, instead we are feeding it into the adder so we have an added value of logic 1 to the current counter value.

For Instance , if the counter starts counting at 00, the full adder will output 01, and if counter starts at 01 , the adder will output 10 and it will continue in the same manner , hence we have a shift in the sequence by using the adder, because if we are in sequence 2 the counter is (01,10,11,00), and if it is at sequence 3 , the counter is (10,11,00,01).

Reminder that this technique is used only for sequence 2 and 3 whereas the sequence 4 is treated in another way.

The figure below presents where this technique happens in the configuration:

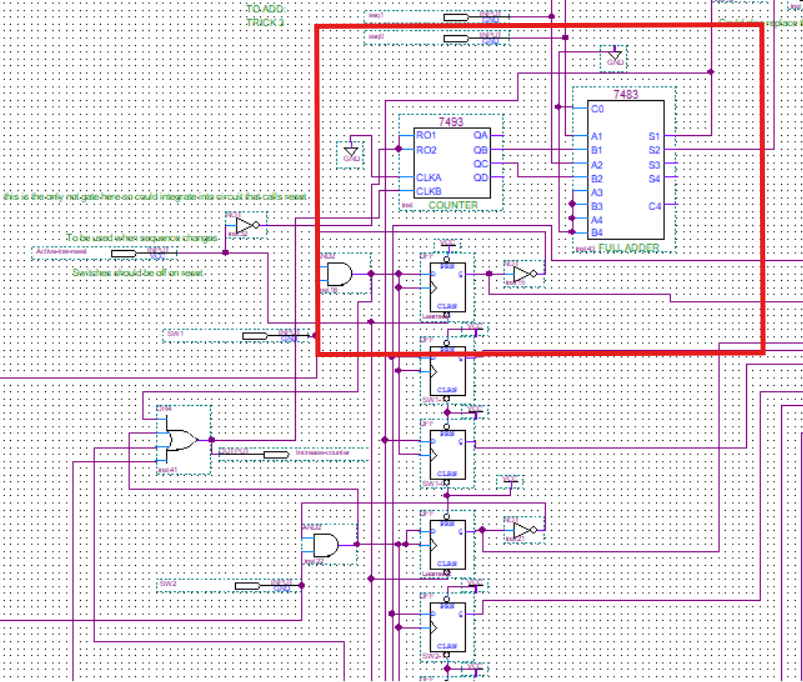


Figure 11 - the technique to handle sequence 2 and 3

For sequence 4 we will have to treat it differently, as it cannot be managed the same way sequence 2 and 3 were managed. So according to the truth table above we can deduce that we have to use a combination three gates in order to implement sequence 4, which are the gates Nand , Nor and XOR ,where the inputs for each gate are classified as follows:

-The Nand inputs are the sequence bits (2 bits),

-Nor gate inputs are the output of the Nand with the input of the first D-flip flop of each switch

-The XOR inputs are the output of the NOR gate with the output of the full adder (used in previous sequences).

**Note that we could also replace these gates by using 74191 to decrement when on sequence 4 which is a counter that counts up and counts down.**

This combination allowed us to implement the logic of sequence 4, introducing a unique handing for the sequence.

A figure below illustrates the way sequence 4 is handled

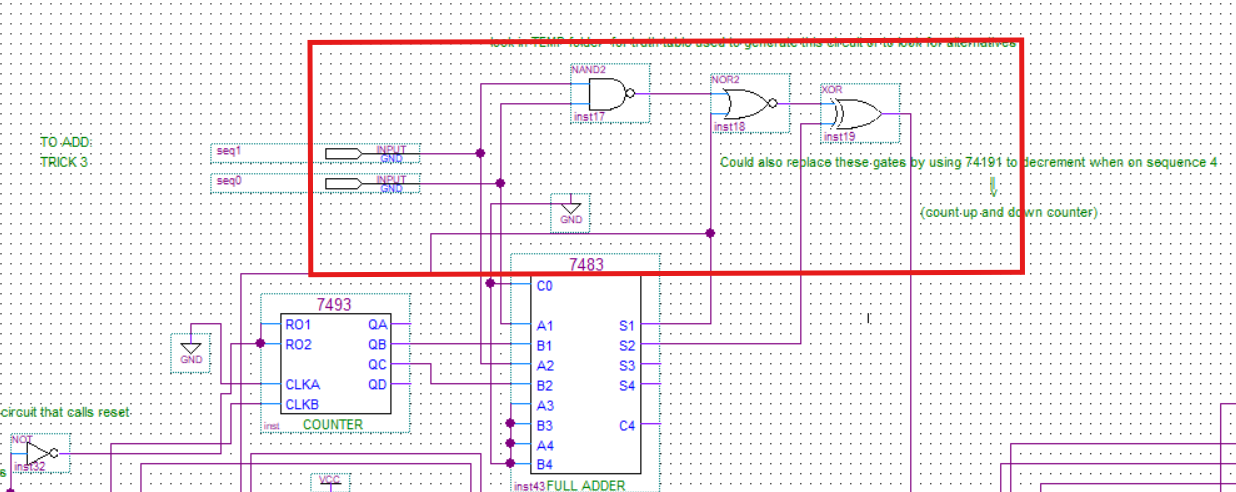
****

Figure 12 - Handling sequence 4

On the LEDs’ side, we used 2 decoder 74139 (inside each IC there is a 2 decoders ) that are active if and only if the switch is on and learned (each responsible to activate the led based on the switch states ), so it will result in four outputs corresponding to the encoded number stored in the D flip-flops and each output is active low(decoders’ output).So, to secure the led proper activation, we connect each decoder first output to a 4-input Nand gate (because as we mentioned that the decoder output is inverted) , and each decoder second output to another 4-input Nand gate , and the same is done for each decoder third and fourth output .This design helps us to maintain that only LED is activated at a time .

A figure below illustrates the 2 decoders and their functions:

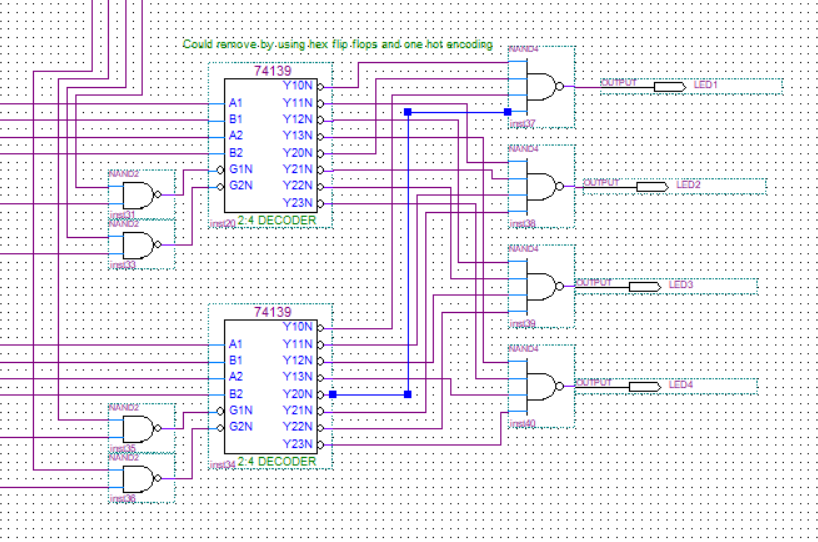


Figure 13 - The decoders and ensuring that the led activates properly

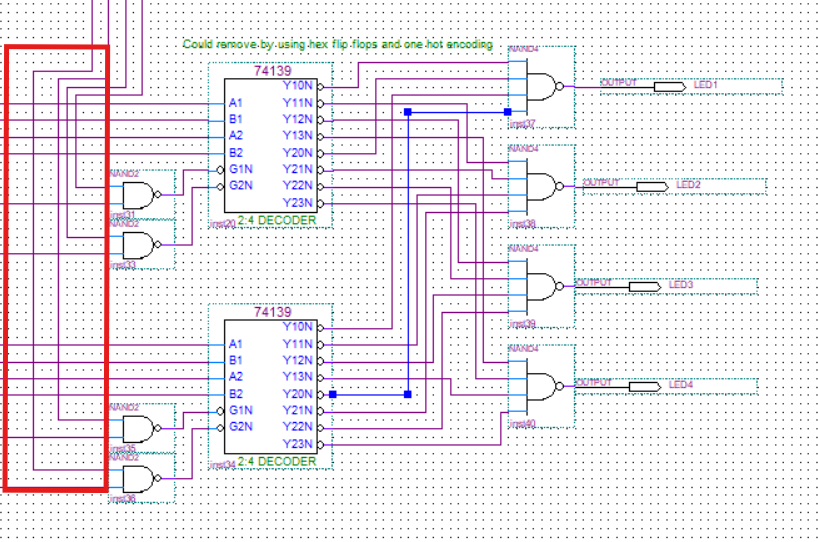


Figure 14 - Every 2 Switches being Nanded and inputted as selectors for every decoder

Noting that these in red are connected to the switches and every two switches are Nanded with each other to be as a selector for both 74139 decoders.

**Putting Everything Together**

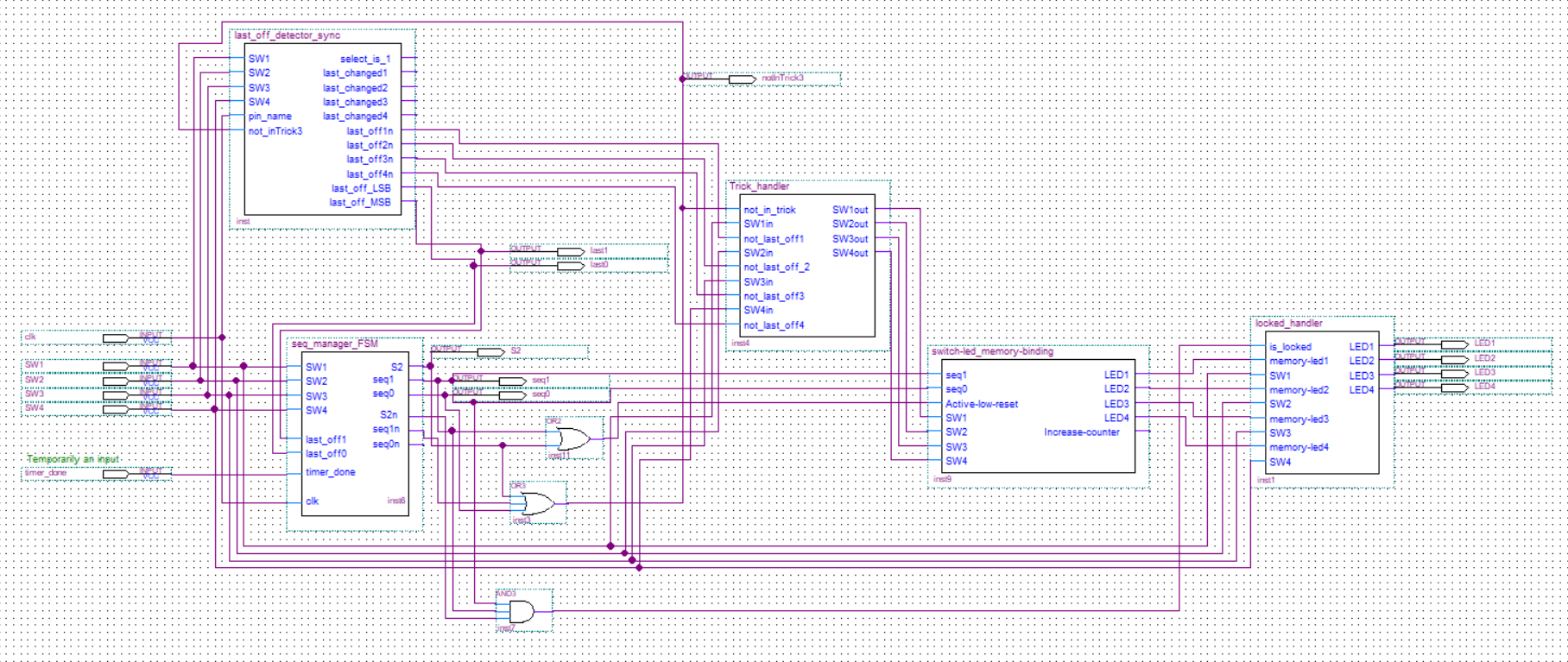


Figure 15 - Entire Quartus Circuit

Some gates are used to determine the not\_in\_trick and not\_locked signals. Moreover, the active low reset signal was connected to S2 OR seq1 (seq1 is the middle bit in the state number (S1)). This way the signal remains high (does not reset) unless we enter the sequence manager state (001).

**Output Analysis**

This section is dedicated to analyzing the output of the Quartus circuit, verifying its correctness. For the 4 sec delay circuit, we used an RC with 555 timer circuit that was not implemented in Quartus. For this reason, timer\_done was treated as an input in Quartus, where we manually provided a pulse when the delay ends. Note that while the circuit used to implement the delay does not provide a pulse, its intended to be reset on upon sequence and switch changes, so practically a pulse would be the result.

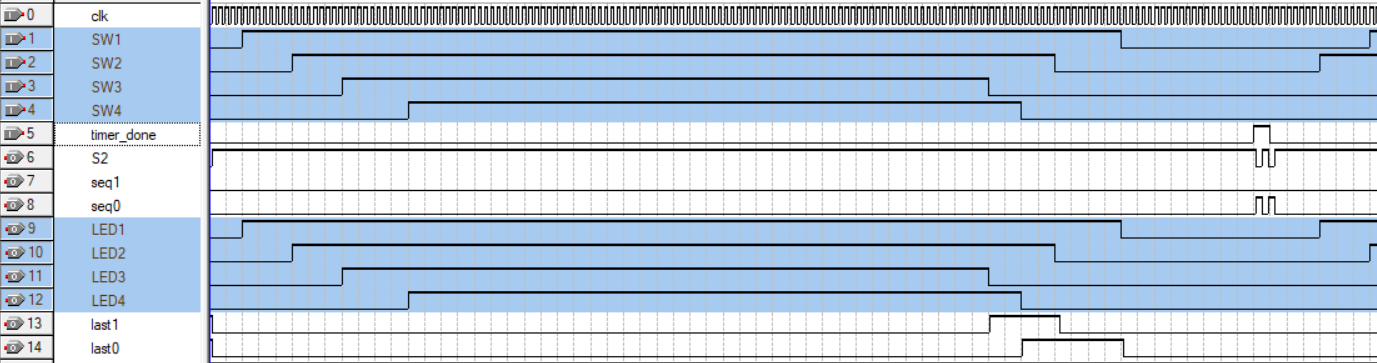


Figure 16 - Quartus output part 1

The simulation starts by turning on switches 1 to 4 in order. As a result, the LEDs also turn on in that order. When turning them off, the LEDs turn off in the same order as the switches since they have been directly mapped. Switch 1 is turned off last, so we will go back to sequence 1.

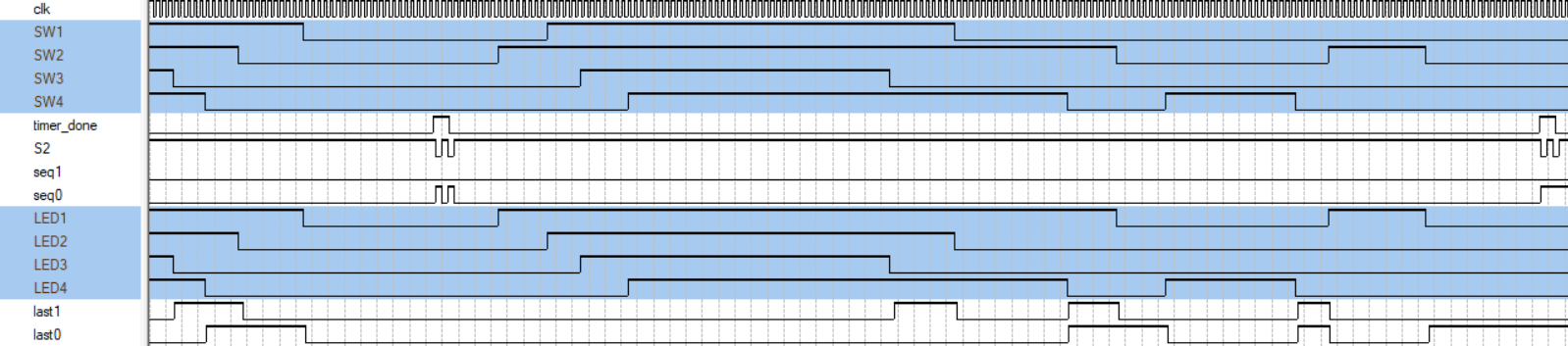


Figure 17 - Quartus output part 2

Then, the switches are turned on in the order 2134. The LEDs still turn on in the order 1234 as intended for sequence 1. They are mapped properly as well. This mapping remains even after all switches are turned off briefly. The mapping is reset and we move to sequence 2 (since switch 2 was last to turn off) when the timer is done.

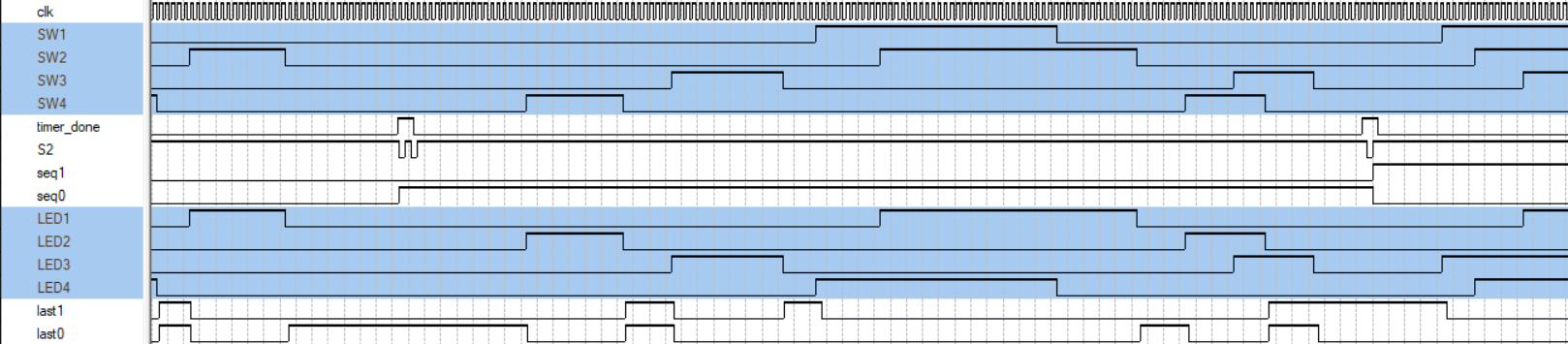


Figure 18 - Quartus output part 3

This time, in sequence 2, the LEDs turn on in the order 2341 as intended, and the mapping works properly. Each Switch is linked to an one LED consistently throughout this section. On the next timer pulse, we enter sequence 3.

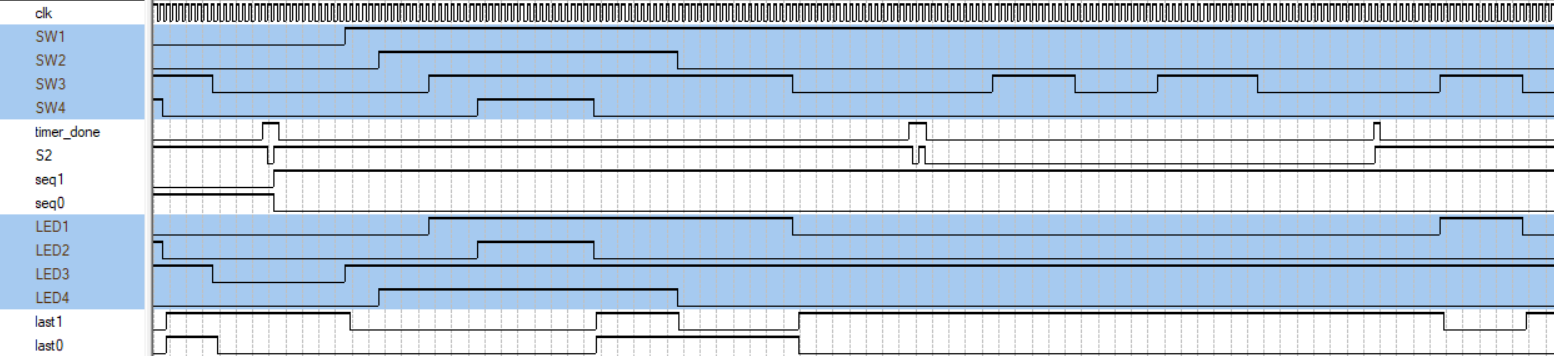


Figure 19 - Qaurtus output part 4

In this part, we see the LEDs turn on in the order 3412 as intended. After the switches are mapped, a timer pulse is provided with sw1/led3 on. This makes us move to the trick state as we can see by observing S2, seq1, seq0. After the trick is activated, sw3 which is linked to LED1 is turned on and off, but LED1 remains off. Throughout the trick, LED3 remains on. Then, another timer pulse is provided and when sw3 is turned on, LED1 turns on.

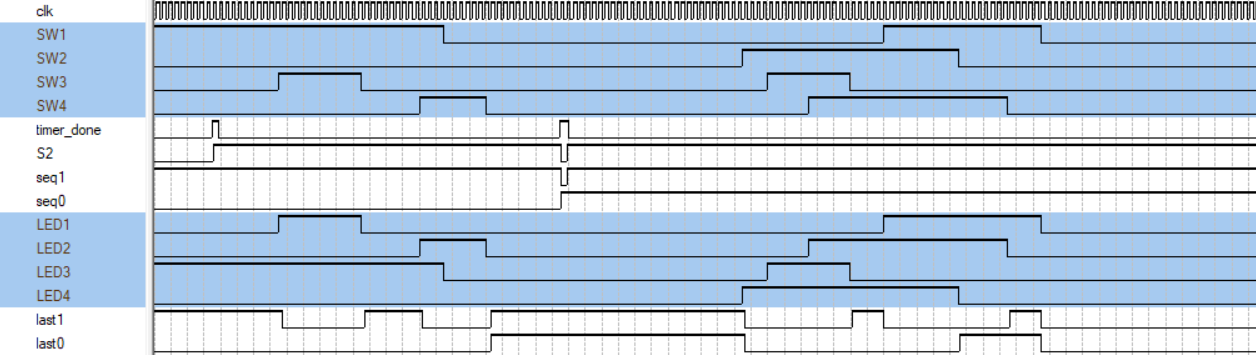


Figure 20 - Quarus output part 5

Then, after all the switches are turned off and timer finishes, we move to sequence 4. The switches are mapped properly with LEDs with the LEDs turning on is the order 4321.

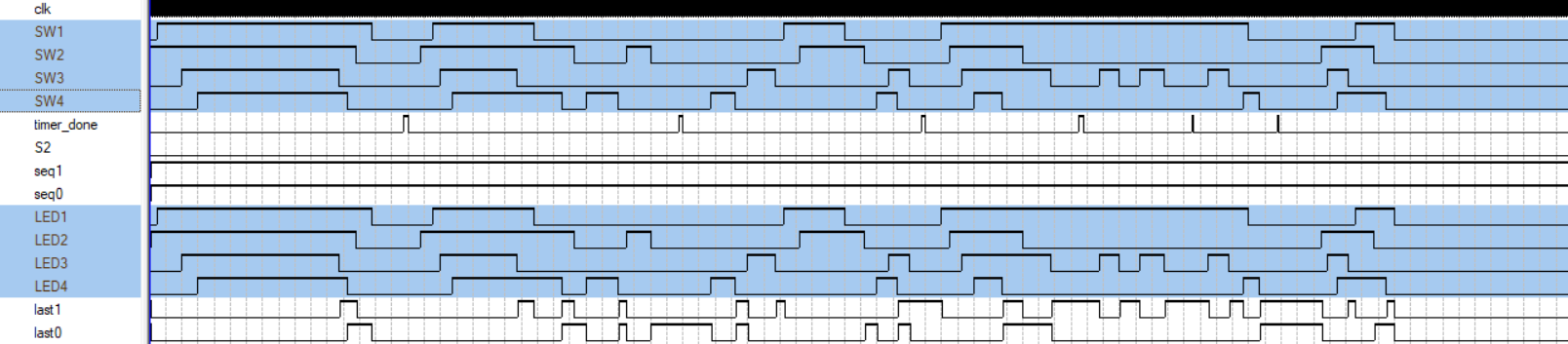


Figure 21 - Quartus output locked

Finally, this snapshot is the same as the entire previous simulation but switch 2 was turned on before turning on the circuit. It is clear that the switches and LEDs are now mapped directly (1-1, 2-2,…).

# Breadboard design and analysis

Due to time constraints, we were unable to fully map and plan the breadboard layout before starting the physical assembly. Instead of preparing a detailed breadboard design, we directly translated the Quartus digital design into physical connections on the breadboard.

While the Quartus simulations were successful, this approach led to several practical issues during hardware implementation:

* Increased wiring complexity caused by long and unorganized connections.
* Signal integrity problems, including noise and unstable outputs.
* Difficulty in troubleshooting and isolating faults, as the physical layout did not follow a clean modular structure.

As a result, the board did not function properly on the first attempt, requiring significant time for debugging and corrections. This experience highlighted the importance of breadboard mapping and modular assembly, especially in complex designs with multiple ICs and interconnected subsystems.

In future implementations, dedicating time to properly plan the breadboard layout, using clear block diagrams and wire management strategies, would greatly enhance the reliability and ease of assembly. The following figure shows the aftermath of 40 ICs on crammed 5 breadboards:

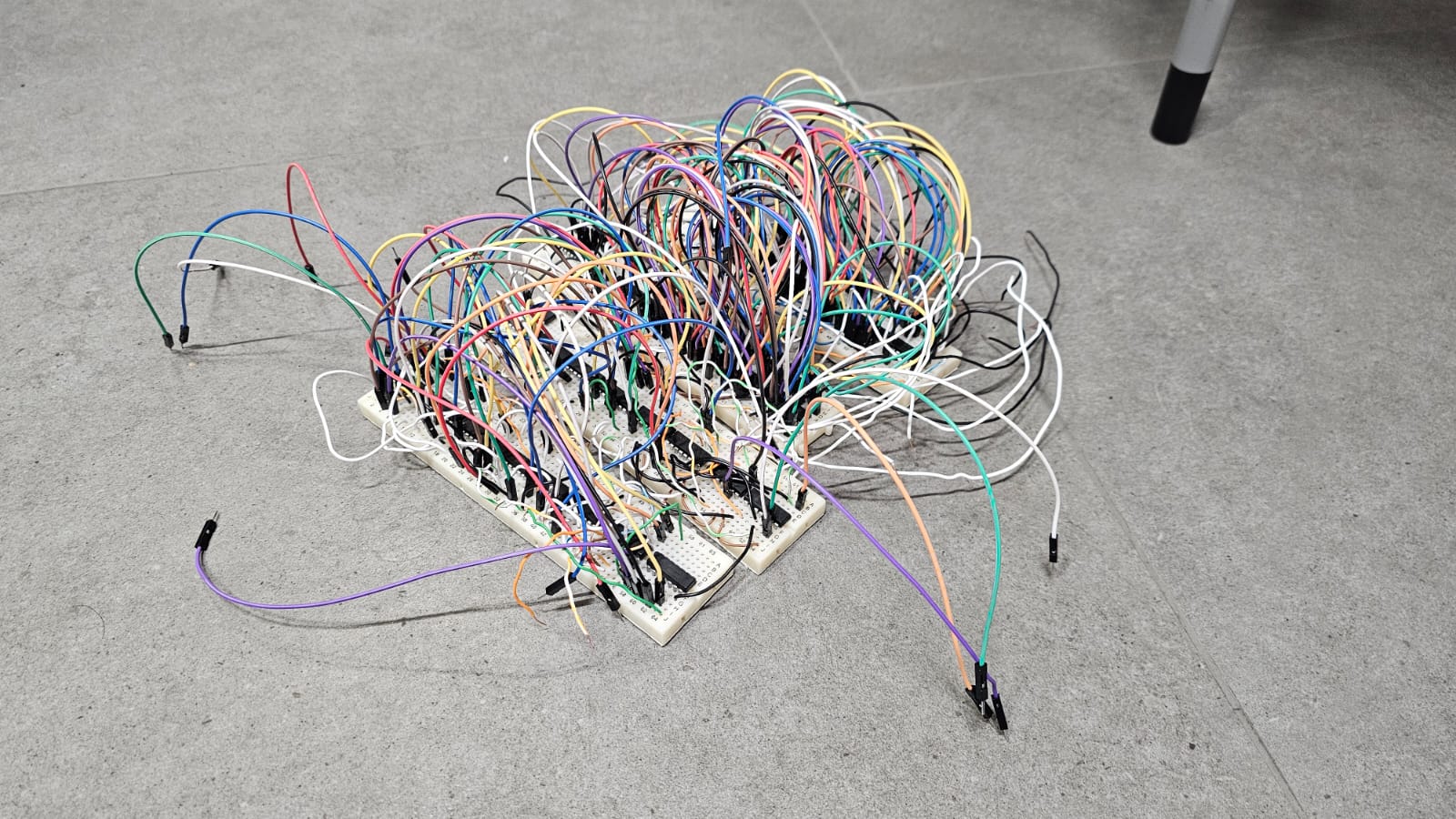


Figure 22 - Breadboard aftermat

# Financial Study

The table below shows a detailed financial analysis of our design, as we tended to maximize the efficiency and minimize the number of gates as possible.

Table 1 - Prices and quantities of the components used

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Quantity | Unit price (USD) | Total Cost (USD) |
| Logic (74LS)  74LS00,08,32,86,157,151,175,25,11,20,139,93,83 | 40 | 0.5 | 20 |
| LEDs | 4 | 0.15 | 0.6 |
| Seven Segment Display | 1 | 0.9 | 0.9 |
| 220 Ohm resistors (packet) | 1 | 0.3 | 0.3 |
| pF capacitors | 4 | 0.1 | 0.4 |
| Breadboard | 6 | 1.2 | 7.2 |
| Wires | 100 | 0.01 | 1 |
| Switches | 4 | 0.8 | 3.2 |

Noting that extra wires were used from the project members own resources.

Multiple revisions were made in order to reduce the number of ICs used. The approach was intuitive. As for the main optimizations, we used many multiplexers (74LS157, 74LS151) reducing many gates. Registers were used instead of flipflops where it applies. And many small minimization were used and one example of that is shown in the following block:

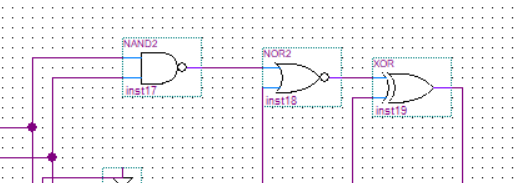


Figure 23 - Potential optimization

The small block in the memory binding block is quite small but uses 3 different ICs when implemented on a breadboard. This could have been optimized by using a 74LS191 chip Up and Down counter that lets us dynamically move between the sequences.

# Delay analysis

In our Quartus simulations, we did not encounter any propagation delay issues that required modifications to our design. This is because we used functional simulation, which focuses on the logical correctness of the circuit rather than modeling gate-level or interconnect delays. Functional simulation assumes ideal conditions, where signal changes propagate instantly through logic gates. Additionally, our design follows a fully synchronous approach, with all state changes occurring on the rising edge of a global clock. This eliminates the risk of glitches or race conditions in simulation and ensures reliable timing behavior in hardware. Should the design be synthesized and implemented physically, Quartus timing analysis tools (such as TimeQuest) would be used to verify that all paths meet required setup and hold times. Therefore, the absence of observable delays in simulation is expected and consistent with best practices for synchronous digital design.

# Power Consumption

We now look for the datasheet for every gate, and find the current it draws

Table 2 - Power analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Quantity | Current drawn | Total current |
| 74LS logic ICs | 40 | 9.6 mA | 384mA |
| LEDs | 4 | 15.5mA | 62 mA |
| Sevensegment display | 1 | 60 mA | 60 mA |

Now using the general power formula we can find that

Power (P)=V×I=5V×0.506=2.53 Watts

# Problems Faced

One of the main challenges encountered during the project was adapting the **Last-Off** Detector Block to properly support Special Trick 3. Specifically, the problem arose from the fact that during Trick 3, the system should retain the last switch-off information, even if other switches are toggled. Initially, the detector would update its state with every switch toggle, which conflicted with the intended trick behavior.

To address this, we iterated through three design versions of the Last-Off Detector block:

* **First Iteration:** A basic design where every switch toggle would update the last-off state unconditionally. This failed to meet Trick 3 requirements, as it could overwrite the last-off information during the trick.
* **Second Iteration:** An improved version attempted to block updates during Trick 3 by gating some control signals, but it still introduced race conditions and inconsistencies when exiting the trick state.
* **Final Iteration (Accepted Design):** The final solution introduced multiple register stages, with a dedicated control flag indicating whether the system was currently in Trick 3. The key idea was to add a buffer register stage that only updates its value when the system is not in Trick 3. This way, even if switches are toggled during the trick, the last-off state remains unchanged, ensuring correct behavior.

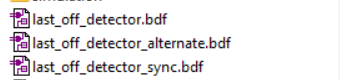


Figure 24 - Last off detector iterations

# Key Design Points that Present Advantages Over Alternative Designs

Our Logic-Controlled Board design offers several key advantages over alternative approaches implemented by other groups. The design was carefully optimized to balance functionality, complexity, and hardware efficiency. Below are the main points highlighting its superiority:

**1. Minimized IC Count**

One of the most significant advantages of our design is the remarkable reduction in the number of ICs used. While most groups required over 50 ICs to implement the project, our team successfully completed the design using only 40 ICs. This was achieved through:

* Careful optimization of Boolean functions.
* Smart reuse of multiplexers (74LS157, 74LS151) to replace multiple logic gates.
* Efficient use of registers (e.g., 74LS175) instead of discrete flip-flops where applicable.

**2. Modular Design Architecture**

We structured our system around five main functional blocks, each responsible for a specific task:

* **FSM Sequence Manager Block**
* **Switch-LED Memory Binding Block**
* **Locked Mode Handler**
* **Trick Handler Block (Cap Removal)**
* **Last-Off Detector Block**

This modular approach provided multiple benefits:

* Simplified troubleshooting and debugging.
* Easier testing and verification of individual blocks before full integration.
* Clear documentation and presentation of system functionality.

**3. Resource-Efficient Implementation**

By optimizing the design at both the logic and hardware levels, we reduced:

* Chip area usage on the breadboard.
* Wiring complexity, leading to a cleaner, more reliable build.
* Power consumption, since fewer ICs and interconnections translate to lower current draw.

**4. Flexible and Scalable FSM Design**

The use of a finite state machine (FSM) as the core of our control logic allowed for:

* Smooth handling of all project sequences and tricks.
* Simplified addition of new features (such as bonus tricks) without major redesign.
* Logical separation between states like Boot, Locked, Sequences 1–4, and Trick state.

# Conclusion

The Logic-Controlled Board project was a comprehensive application of digital logic design, combining theoretical knowledge with practical hardware implementation. By developing a system that dynamically reconfigures LED activation sequences based on the last switch turned off, we successfully demonstrated advanced use of finite state machines, sequential circuits, and logic optimization techniques.

A key achievement was reducing the hardware complexity to just 40 ICs, the lowest among all groups, through a modular design approach and efficient resource utilization. This not only minimized cost and power consumption but also simplified circuit debugging and improved reliability.

Throughout the project, we overcame technical challenges such as modifying the Last-Off Detector for Trick 3, managing switch debouncing, and ensuring stable circuit operation. These challenges enhanced our problem-solving skills and deepened our understanding of practical digital system design.

In conclusion, the project successfully met all design objectives, provided valuable hands-on experience, and highlighted the importance of structured design, optimization, and real-world implementation of logic circuits. It stands as a solid example of bridging theory with practice in engineering education.