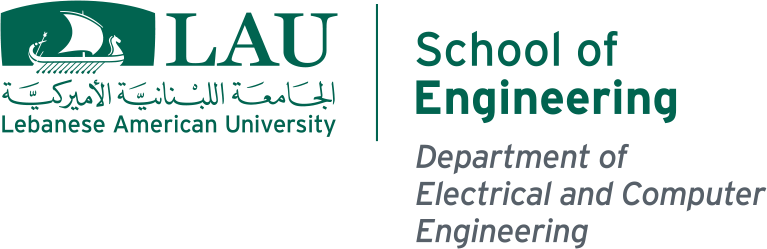
**Lebanese American University**

****

***COE322 – Logic Design Lab***

***Logic Design Lab Project***

***Logic-Controlled Board***

Nadia Al Masri 202302750

Lara Bleik 202301445

Lynn Kaafarani 202308217

13/5/2025

Table of Contents

[Table of Figures 3](#_Toc198093466)

[Table of Tables 4](#_Toc198093467)

[Abstract 5](#_Toc198093468)

[Introduction 6](#_Toc198093469)

[Components and equipment used 7](#_Toc198093470)

[Design and Analysis 8](#_Toc198093471)

[I. Boot & Switch 2 8](#_Toc198093472)

[II. Sequence detector 9](#_Toc198093473)

[III. Counter 11](#_Toc198093474)

[IV. LEDs and sequence 16](#_Toc198093475)

[V. 555 Timer 23](#_Toc198093476)

[VI. Capless hack 25](#_Toc198093477)

[VII. Circuit link 27](#_Toc198093478)

[Cost and Power consumption 28](#_Toc198093479)

[Challenges 30](#_Toc198093480)

[Advantages 31](#_Toc198093481)

[Conclusion 32](#_Toc198093482)

[References 33](#_Toc198093483)

# Table of Figures

[Figure 1 –Boot and SW2 path 8](#_Toc198093446)

[Figure 2 –The 1:1 Mapping on Quartus 8](#_Toc198093447)

[Figure 3 –The sequence detector on Quartus 10](#_Toc198093448)

[Figure 4 –The output of the Sequence Detector 10](#_Toc198093449)

[Figure 5 –Implementation of the counter on Quartus 13](#_Toc198093450)

[Figure 6 –The output of the counter 14](#_Toc198093451)

[Figure 7 -Delay Analysis 15](#_Toc198093452)

[Figure 8 –The counter circuit as a block 18](#_Toc198093453)

[Figure 9 –Implementation of the whole circuit 18](#_Toc198093454)

[Figure 10 –Output of the circuit 19](#_Toc198093455)

[Figure 11 –The case of selectors 00 output 19](#_Toc198093456)

[Figure 12 –The implementation of the LEDs and sequences on circuitos 20](#_Toc198093457)

[Figure 13 –Case of E1E0=00 21](#_Toc198093458)

[Figure 14 –Case of E1E0=01 21](#_Toc198093459)

[Figure 15 –Case of E1E0=10 22](#_Toc198093460)

[Figure 16 –Case of E1E0=11 22](#_Toc198093461)

[Figure 17 –The 555 timer in astable mode 23](#_Toc198093462)

[Figure 18 –The capless hack implementation on quartus 25](#_Toc198093463)

[Figure 19 –The output of the capless hack 26](#_Toc198093464)

[Figure 20 –The circuit link implementation on quartus 27](#_Toc198093465)

# Table of Tables

[Table 1 –The truth table of the priority encoder 9](#_Toc198093418)

[Table 2 –The truth table of the counter 11](#_Toc198093419)

[Table 3 –The K-map of the output of the counter I1 11](#_Toc198093420)

[Table 4 - The K-map of the output of the counter I0 12](#_Toc198093421)

[Table 5 –The truth table of the 2 bit encoder 12](#_Toc198093422)

[Table 6 –The k-map of E0 12](#_Toc198093423)

[Table 7 –The k-map of E1 12](#_Toc198093424)

[Table 8 –The truth table of the LEDs in seq1 16](#_Toc198093425)

[Table 9 –The truth table for the LEDs in seq2 16](#_Toc198093426)

[Table 10 - The truth table for the LEDs in seq3 17](#_Toc198093427)

[Table 11 - The truth table for the LEDs in seq4 17](#_Toc198093428)

[Table 12 –Table of the Cost and Power consumption 28](#_Toc198093429)

# Abstract

This project’s purpose is to apply and integrate the various techniques and skills learned throughout the semester in logic design. It serves as a hands-on implementation of theoretical knowledge, reinforcing core concepts such as Boolean algebra, combinational and sequential logic, state machine design, and circuit optimization. In this project we are required to design a logic-controlled board with four switches and four colored lamps (red, green, blue, and yellow), where the lamp activation sequence is dynamically determined by the last switch turned off. Although each switch is initially matched to a lamp of the same color, users can rearrange switch caps and lamp positions freely, and the system will still correctly associate each switch with its corresponding lamp. The system also includes an interactive feature that disables a switch when its cap is removed, restoring functionality once the cap is replaced. Built using digital logic, memory elements, and finite state machines, the circuit creates the illusion of intelligent control by adapting to user interactions in real time.

# Introduction

In this project, we have to design a logic-controlled board using sequential and combinational logic. We started the project with the initial focus on Switch 2. If Switch 2 was in the OFF state (logic 0), we proceeded with a custom implementation, which is detailed in later sections. However, if Switch 2 was ON (logic 1), we aimed for a straightforward one-to-one mapping approach each switch directly controlled its corresponding LED. For instance, switch 1 would turn on LED 1, establishing a simple and intuitive control mechanism as our baseline logic before incorporating more complex behaviors.

# Components and equipment used

The equipment used in this project are:

* AND Gate (74LS08)
* OR Gate (74LS32)
* NOT Gate (74LS04)
* D Flip-Flop (74LS74)
* 4x1 MUX (74LS157)
* XOR Gate (74LS86)
* Resistors
* Capacitors
* 555 timer
* Quartus II
* Circuitos

# Design and Analysis

### Boot & Switch 2

The boot state is determined by checking Switch 2 if it’s on or off. According to SW2, the path is chosen. If SW2 is ON, then we immediately go to the locked state which is 1:1 mapping; however, if SW2 is OFF, we should go to the sequence detector to check the last SW OFF, then we pass by the counter and the MUX. This is observed in the figure below.

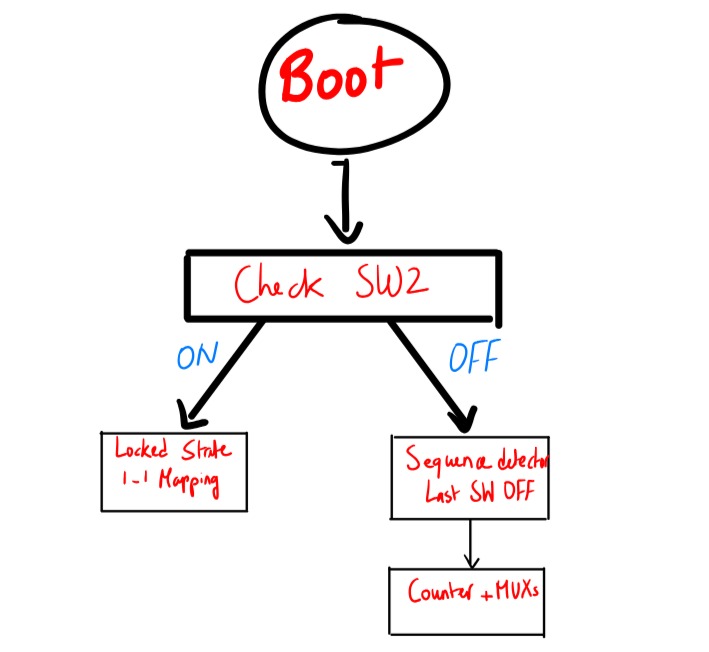


Figure 1 –Boot and SW2 path

The 1:1 mapping is observed in the figure below where each switch has a specific LED output.



Figure 2 –The 1:1 Mapping on Quartus

## Sequence detector

The sequence detector is implemented using D flipflops and logic gates in order to determine the falling edge of each switch. We have 4 switches (S0, S1, S2 and S3) and each switch has a falling edge where it is obtained using the D flip-flop, so it is the AND of the NOT current state and previous state. The falling edge is to determine the transition from HIGH to LOW (1 to 0). After obtaining the falling edge of each switch, we implemented a priority encoder which had two conditions, the falling edge of each switch and the condition of having all switches off. The priority encoder is not used in real-life sequence detectors; however, we needed to implement it in case we have two switches that are ON or OFF at the same time, so it handles conflicting inputs. At this case, we must choose the greater index switch; for example, if we have SW2 and SW3 we choose SW3. Finally, we should implement at the end of the circuit two D flip-flops in order to save the sequence. So, to explain further, once the falling edge signals for each switch are extracted, the circuit passes this information to the priority encoder. This encoder has a dual function: it first ensures that it responds only when a falling edge is detected on one of the switches, and second, it handles the case where multiple switches are triggered simultaneously or are already in an active state. This design decision ensures deterministic behavior and prevents ambiguity in the sequence being detected. Finally, the output of the encoder, which is typically a 2-bit binary code representing the selected switch, is stored using two more D flip-flops. These final flip-flops hold the result until a new falling edge is detected, allowing the system to track and store the most recent switch release in a stable and reliable manner.

The equations of the priority encoder were derived from the truth table below where F is the falling edge of each switch AND the condition of having all switches off and Y0 and Y1 are the outputs of the priority encoder since it is a 2-bit encoder.

Table 1 –The truth table of the priority encoder

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F1 | F2 | F3 | F4 | Y1 | Y0 |
| 0 | 0 | 0 | 0 | X | X |
| 1 | 0 | 0 | 0 | 0 | 0 |
| X | 1 | 0 | 0 | 0 | 1 |
| X | X | 1 | 0 | 1 | 0 |
| X | X | X | 1 | 1 | 1 |

According to the above truth table, we derive the following equations of the outputs Y0 and Y1:

Y1=F3+F4

Y0=F4+F2.F3’

When implementing on Quartus, we found:

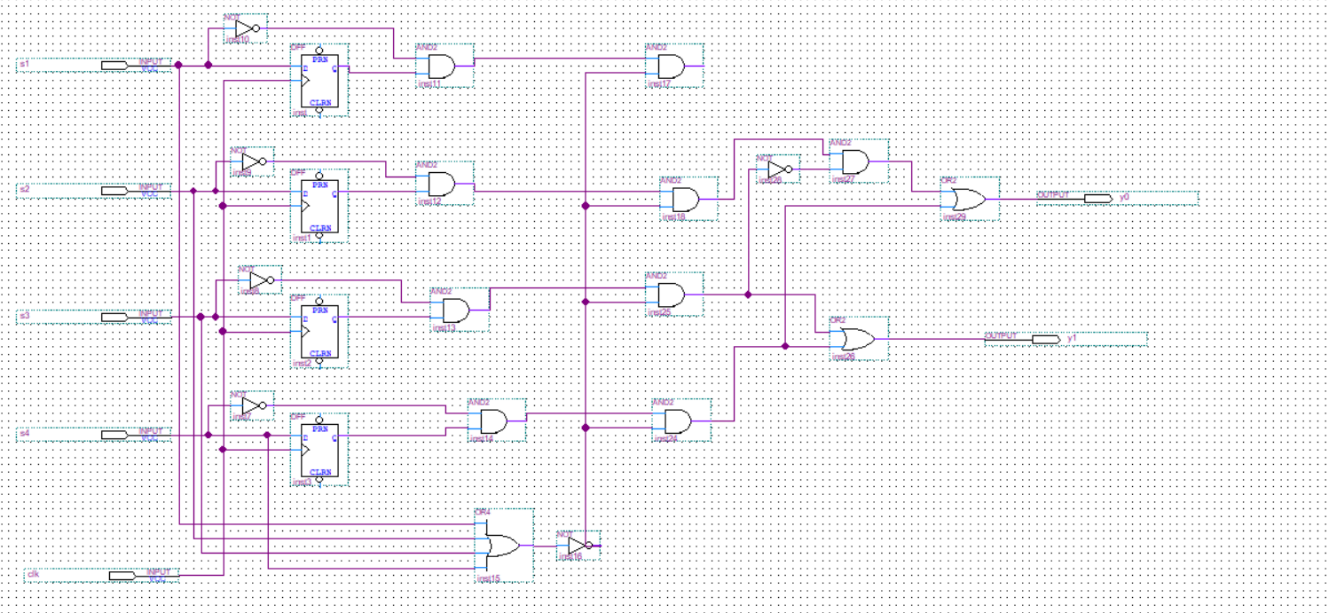


Figure 3 –The sequence detector on Quartus

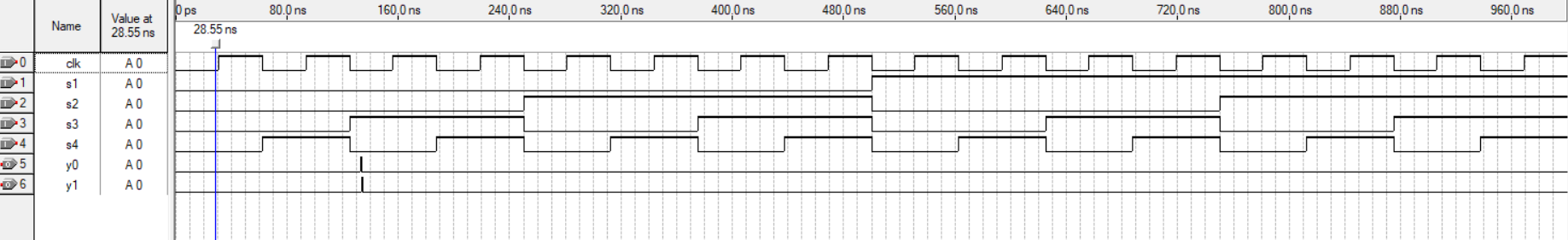


Figure 4 –The output of the Sequence Detector

After analyzing the output of the sequence detector, we observed that the output consistently remains at 0, even when the correct sequence is applied. This was one of the main issues we faced during testing, and initially, we were unsure how to address it. The logic of the design appears correct; however, the problem seems to be related to the clock synchronization and the state transitions within the detection cycle. There may also be a feedback or back-path issue in the FSM (Finite State Machine) that causes it to either reset prematurely or skip the final output state, preventing the correct detection of the sequence. This highlights how important proper clocking and state transition design is in sequential circuits, even when the logical concept is accurate.

## Counter

In this section, we were aiming to design a counter that counts the number of switches ON.

We also noticed that regardless of which switch is ON, the LED will turn on. The following truth table was as follows:

Table 2 –The truth table of the counter

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | | Output | | |
| S1 | S2 | S3 | S4 | I2 | I1 | I0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

If 1 switch is ON, the count is 1.

if 2 switches are ON, the count is 2.

if 3 switches are ON, the count is 3.

if 4 switches are ON, the count is 4.

Now, we derive the equations of I2, I1 and I0.

I2=S1.S2.S3.S4

To get the expression of I1, we did a K-map:

Table 3 –The K-map of the output of the counter I1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S1 S2 S3 S4 | 00 | 01 | 11 | 10 |
| 00 |  |  | 1 |  |
| 01 |  | 1 | 1 | 1 |
| 11 | 1 | 1 |  | 1 |
| 10 |  | 1 | 1 | 1 |

We get: I1=S1’.S3.S4+S1’.S2.S4+S2.S3.S4’+S1.S2’.S4+S1.S2’.S3+S1.S2.S3’

=S4S1’(S3+S4)+S1S2;(S4+S3)+S2(S1S3’+S3S4’)

To get the expression of I0, we did a K-map:

Table 4 - The K-map of the output of the counter I0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S1 S2 S3 S4 | 00 | 01 | 11 | 10 |
| 00 |  | 1 |  | 1 |
| 01 | 1 |  | 1 |  |
| 11 |  | 1 |  | 1 |
| 10 | 1 |  | 1 |  |

After simplification, we get: I0=S1⊕S2⊕S3⊕S4

Now, we encoded these 3 bits to 2 bits counter to get the following table:

Table 5 –The truth table of the 2 bit encoder

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I2 | I1 | I0 | E1 | E0 |
| 0 | 0 | 0 | X | X |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | X | X |
| 1 | 1 | 0 | X | X |
| 1 | 1 | 1 | X | X |

Now, we derived the equations of E1 and E0 using K-maps:

For E0:

Table 6 –The k-map of E0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I2 I1I0 | 00 | 01 | 11 | 10 |
| 0 | X |  |  | 1 |
| 1 | 1 | X | X | X |

E0=I0’

For E1:

Table 7 –The k-map of E1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I2 I1I0 | 00 | 01 | 11 | 10 |
| 0 | X |  | 1 |  |
| 1 | 1 | X | X | X |

E1=I0’I1’+I0I1=(I0⊕I1)’

Now, to make sure of our circuit, we implemented on Quartus:

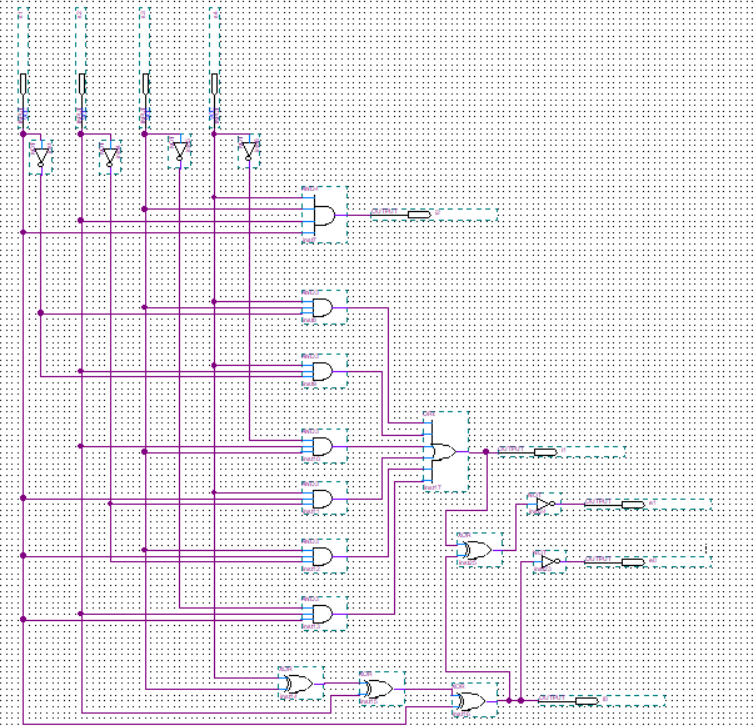


Figure 5 –Implementation of the counter on Quartus

Now, we simulate the circuit and we will analyze the obtained output:

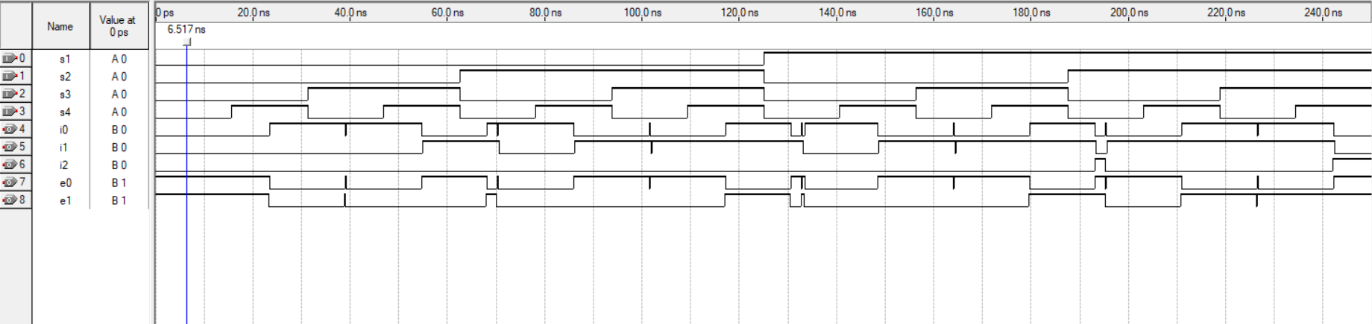


Figure 6 –The output of the counter

We can notice that one switch is on (regardless of which switch), take from 14ns to 46ns interval, I0=1 and all others are 0. This is validated by the truth table where when one switch is on, the 3 bits counter is 001 (I0 is 1) and the 2 bits encoded counter is 00.

Now when 2 switches are on, in the 47 to 62ns interval, I2 is 1 and E0 is 1 and others are 0. This is validated by the truth table where when two switches are on, the 3 bits counter is 010 (I1 is 1) and the 2 bits encoded counter is 01 (E0=1).

The analysis is also validated for all intervals.

We also noticed that the input of E0 and E1 is 1 when no switches are on, this is due to the don’t care condition and don’t affect our work since we won’t use the case where no switches are on in the upcoming parts.

However, some delays were noticed. This may be due to gates propagation

Delay analysis:

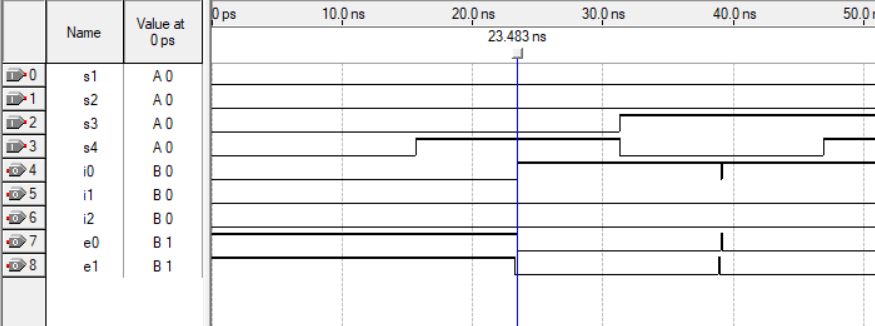


Figure 7 -Delay Analysis

Delay1=|15.23-23.483|=8.253ns

Other delays were also seen:

Delay2=|47.23-55.115|=7.885ns

Delay3=|62.2-68.1|=5.9ns

The average delay is: 7.35ns

## LEDs and sequence

The aim of the previous part was mainly to be able to do this part and relate it to it. So each sequence determines the behavior of LEDs and these LEDs turns ON depending on the number of switches ON (this is used from the counter part).

In sequence 1, the LEDs turns ON as follows: 1 -> 2 -> 3 -> 4

To express each LED in terms of the number of switches on:

Table 8 –The truth table of the LEDs in seq1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **E1** | **E0** | **L1** | **L2** | **L3** | **L4** |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

When deriving the output of each LED in sequence 1, we get:

L1=1

L2=E1+E0

L3=E1

L4=E1.E0

In sequence 2, the LEDs turns ON as follows: 2 -> 3 -> 4 -> 1

To express each LED in terms of the number of switches on:

Table 9 –The truth table for the LEDs in seq2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **E1** | **E0** | **L1** | **L2** | **L3** | **L4** |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

When deriving the output of each LED, we get:

L1=E1.E0

L2=1

L3=E1+E0

L4=E1

In sequence 3, the LEDs turns ON as follows: 3 -> 4 -> 1 -> 2

To express each LED in terms of the number of switches on:

Table 10 - The truth table for the LEDs in seq3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **E1** | **E0** | **L1** | **L2** | **L3** | **L4** |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

When deriving the output of each LED, we get:

L1=E1

L2=E1.E0

L3=1

L4=E1+E0

In sequence 4, the LEDs turns ON as follows: 4 -> 3 -> 2 -> 1

To express each LED in terms of the number of switches on:

Table 11 - The truth table for the LEDs in seq4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **E1** | **E0** | **L1** | **L2** | **L3** | **L4** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

When deriving the output of each LED, we get:

L1=E1+E0

L2=E1

L3=E1+E0

L4=1

Then, we were finding a way to relate these sequences together.

We decided to insert 4 4-1 MUX for each led that takes as selectors the sequence we are in (this is taken from the output of the sequence detector). In this way, we linked the circuits together to get the LED behavior depending on the switch.

Further explanation will be done when analyzing the output of the Quartus circuit.

Now, we will implement it using Quartus:

First, we did the counter circuit above as a block to use it directly.

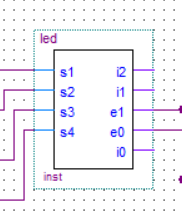


Figure 8 –The counter circuit as a block

Then we implement the whole circuit:

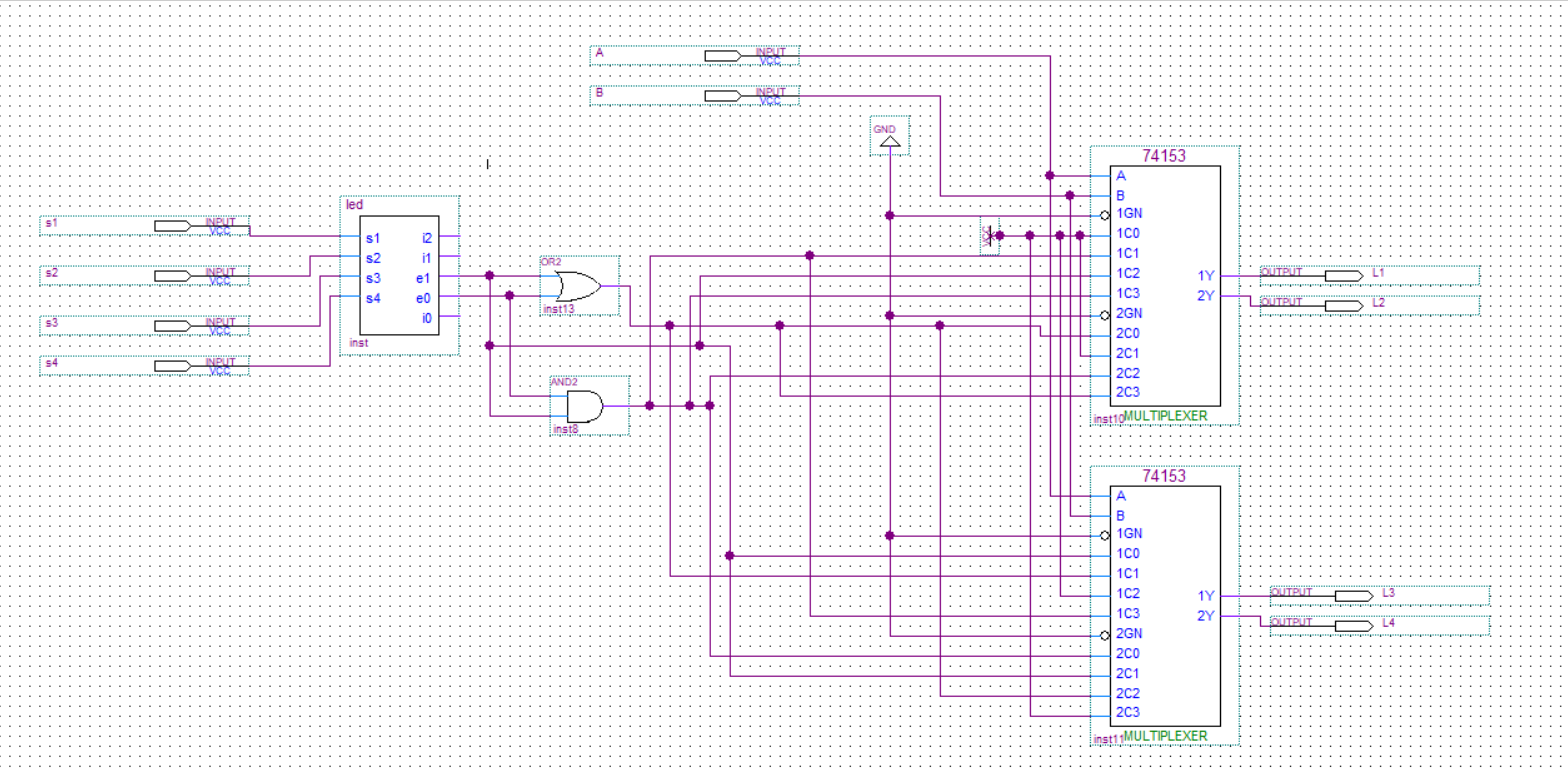


Figure 9 –Implementation of the whole circuit

For the selectors A and B, in this section I set them as inputs to make sure of the output I will get (since the sequence detector we designed is not very accurate).

The output obtained is:

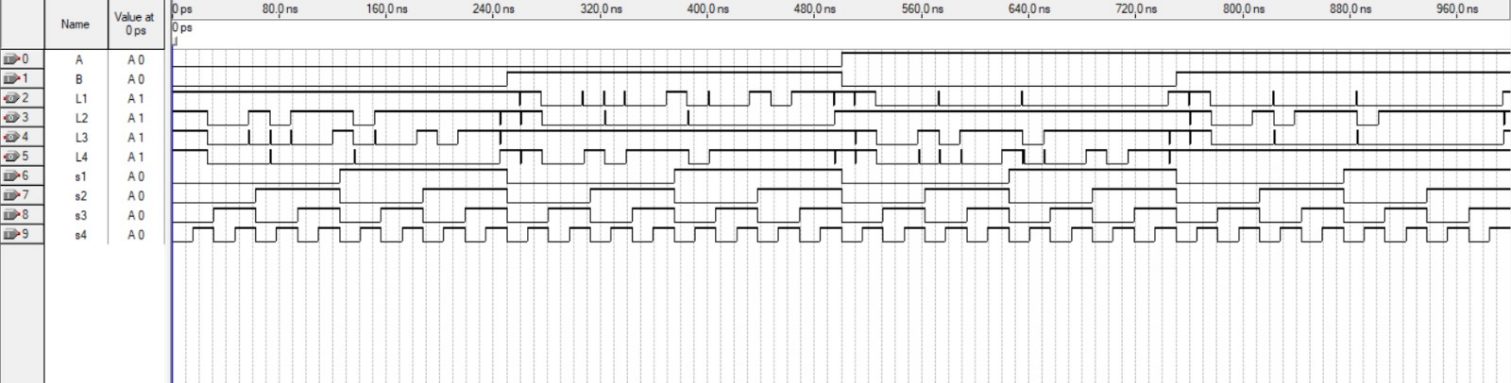


Figure 10 –Output of the circuit

We will zoom to the case of selectors 00 (determining sequence 1) to analyze it.

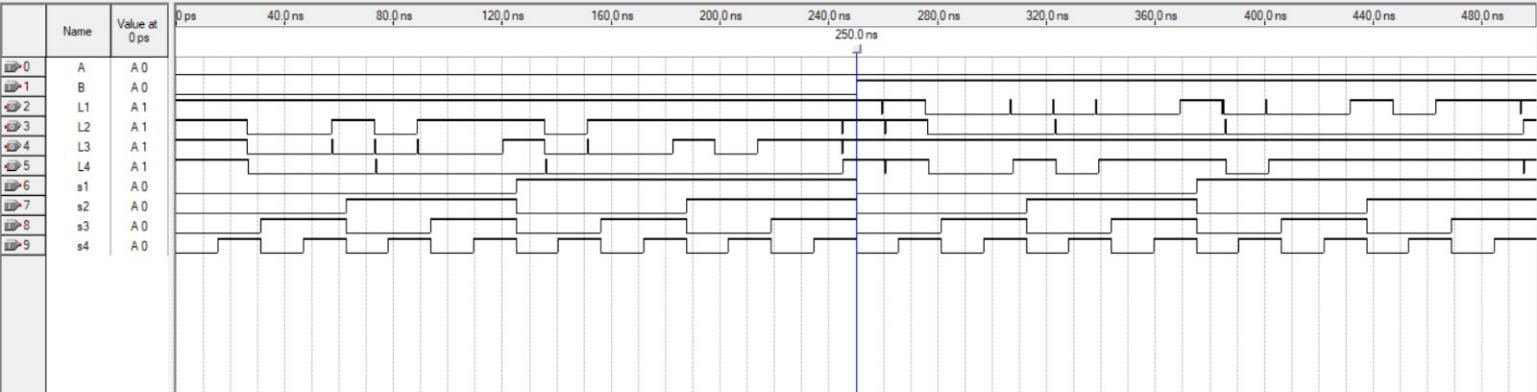


Figure 11 –The case of selectors 00 output

For the interval 0 to 250ns, we are in sequence 1:

L1 is always ON which is true.

L2 turns ON when 2 switches are on (at 53ns) L2 is ON since S3 and S4 are on (with delay to be analyzed later).

L3 is ON at 184ns (with delay) when S3,S1 and S4 are (3 switches ON).

Finally led 4 turns ON when all switches are ON at 245ns.

The same analysis is done for all the sequences, and the output is correct which validates our approach.

However, it is important to notice that all LEDs starts to be ON even tough no switches are ON. This is due to don’t care condition of the counter. This problem is solved when completing the whole project since this case ideally isn’t defined in this part and should not appear as an input.

Delay analysis:

We noticed that a delay exists in the output, this is due to the delay of each gate and now the circuit is more complex, which also increases the delay. The output of the circuit 11 is delayed and introduced as input to circuit 2 so now we will get two consecutive delays.

The calculated delays were:

Delay1=|47.566-57.989|=10.423ns

Delay2=|78.243-88.573|=10.33ns

Delay3=|110.06-120.015|=9.955ns

Delay4=|172.169-182.115|=9.946ns

Delay5=|202.89-214.12|=11.230ns

The average delay is 10.327ns.

We can also notice some glitches and jumps . This is due to unintended changes in output due to differences in signal arrival times at logic gates.

Also it occurs in combinational circuits with multiple paths (our case).

Now, we implemented this circuit on the virtual breadboard.

We used XOR, NOT, AND and OR chips. We made sure that all pins are correctly grounded and set to VCC. We implemented the expression to get I0, I1,I2, E0 and E1.

Then, we connect the led expression in each sequence.

We didn’t tried the MUX in the virtual breadboard, we immediately implement the value to try all cases of sequences.

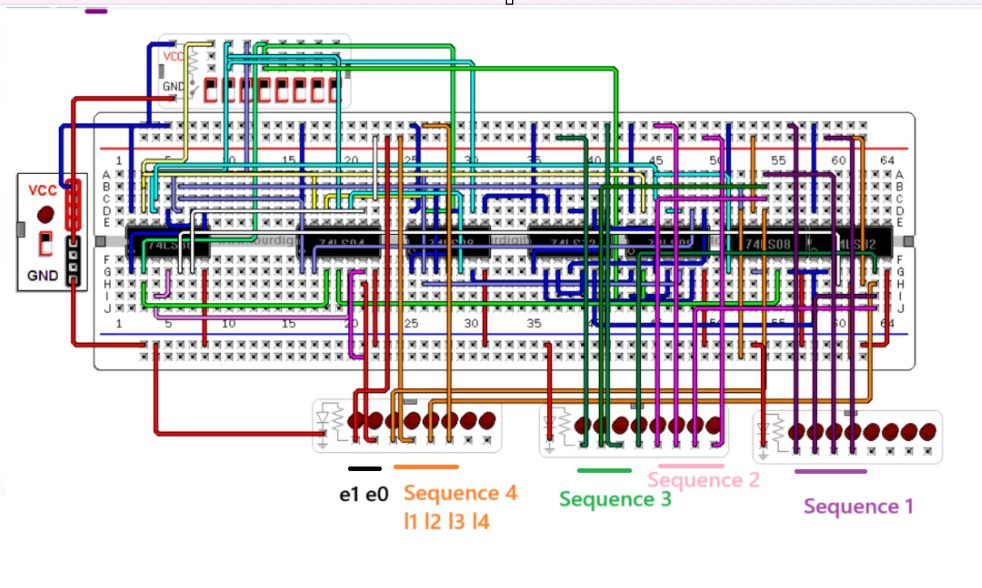


Figure 12 –The implementation of the LEDs and sequences on circuitos

Case of E1E0=00 (1 switch is on)

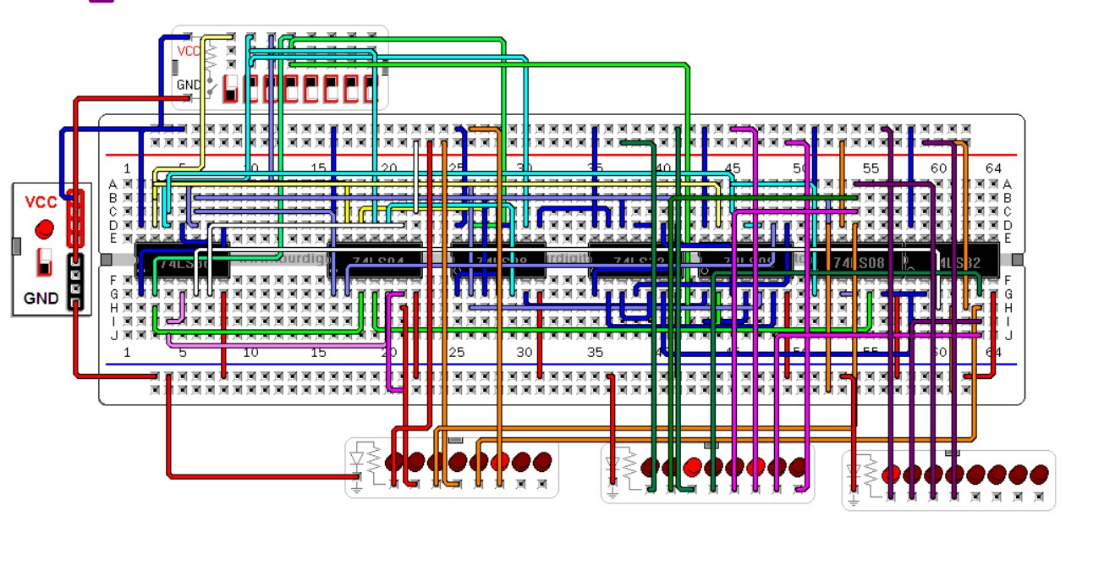


Figure 13 –Case of E1E0=00

Case of E1E0=01 (2 switches are on)

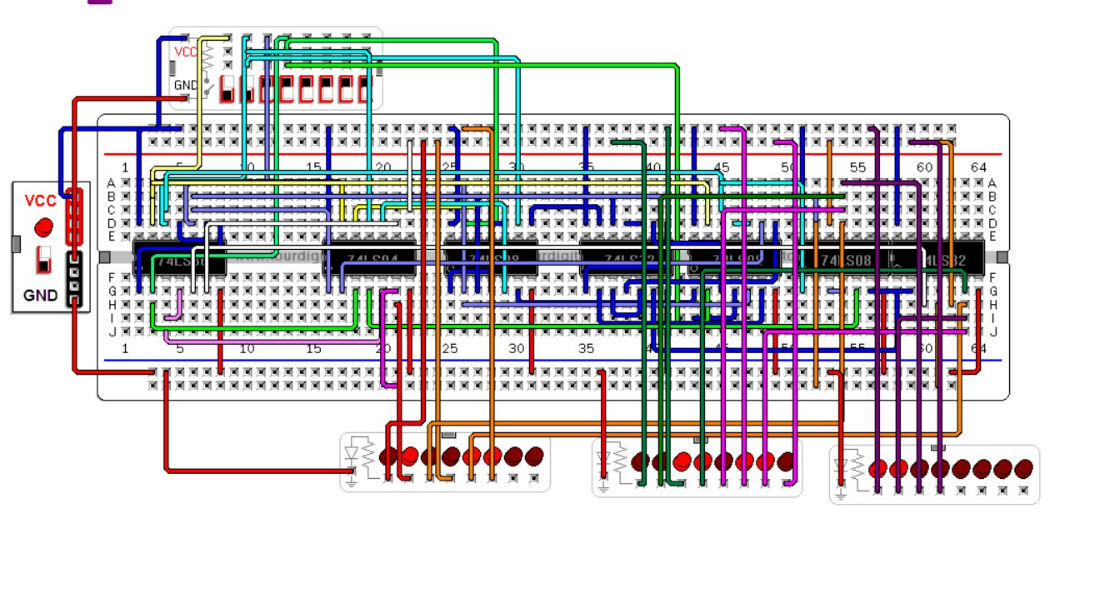


Figure 14 –Case of E1E0=01

Case of E1E0=10(3 switches are on)



Figure 15 –Case of E1E0=10

Case of E1E0=11(4 switches are on)

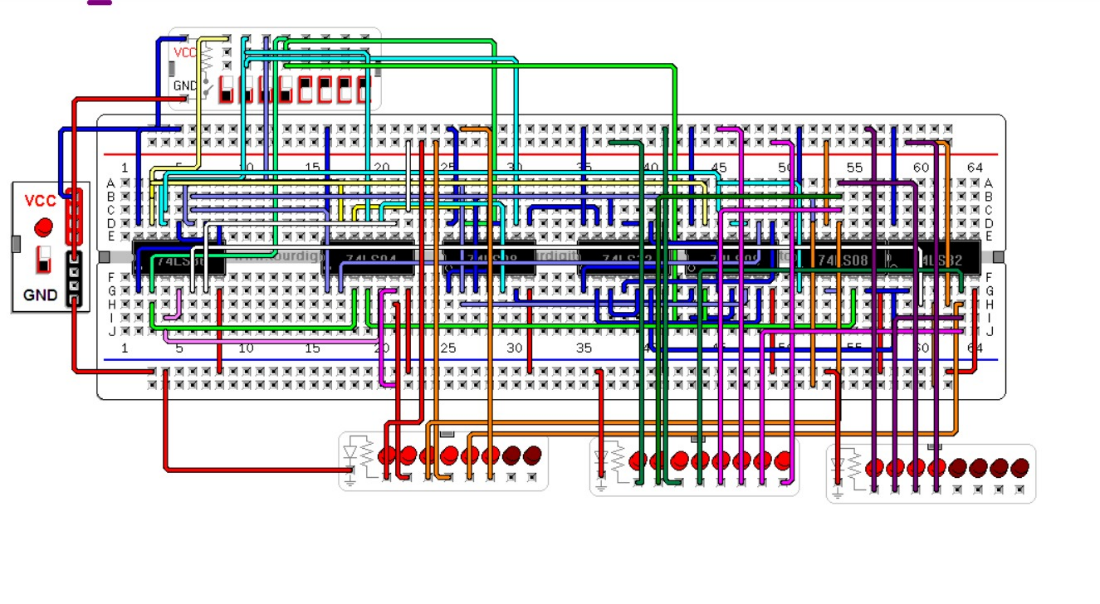


Figure 16 –Case of E1E0=11

The output obtained is correct, which verifies our implementation.

## 555 Timer

The 555 timer is a chip used in 3 different modes. In this project we used it in two different modes which are astable mode and Monostable mode. The astable mode is responsible for building a clock using this 555-timer chip where we implemented a clock of 4 seconds for the change from 1 to 0, from high to low, in the switches without external triggering. Usually, the clock is used automatically, and we are not responsible for creating one, however, for demonstration we built this clock. In order to obtain a 4 second clock, we used specific values for R1, R2 and C1 which are 180kΩ, 180kΩ and 10µF respectively. Then we apply the formula of the Frequency f to find T since .

So, .

Then, . So, it is close to 4s.

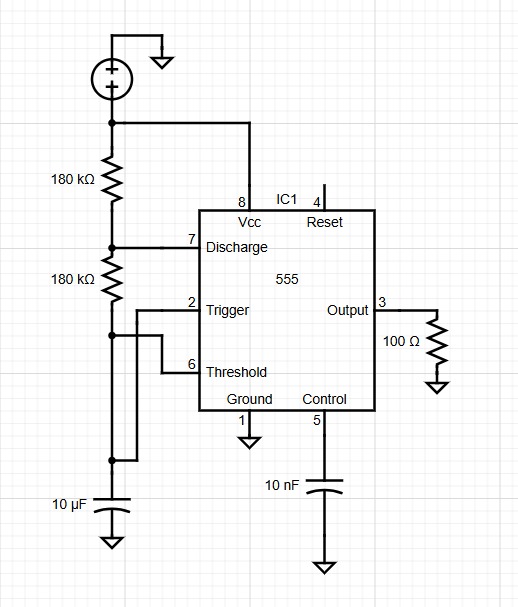


Figure 17 –The 555 timer in astable mode

Now, regarding the monostable 555 timer, it has a one stable state, low, and one unstable state, high, where it has a trigger that outputs HIGH for a precise time period and then it automatically returns to LOW. In order to obtain a monostable 555 timer, we have to update the above circuit with some changes. First, we should remove one of the 180kΩ resistors and remove the connections between pins 6 and 2 and the midpoint of the voltage divider and we should connect these pins 2 and 6 together directly. Then, we should include a 10µF capacitor between pins 6/2 and ground. Moreover, we should include a 180kΩ resistor between pins 6/2 and Vcc. As a trigger input, we should use a push button that opens normally, between pin 2 and ground. The reset pin 4 should be connected to Vcc and the control voltage pin 5 should be connected to ground through a 10nF capacitor. Finally, the discharge pin 7 is left unconnected in the monostable mode. All these changes should be applied to our circuit in order to convert it from astable mode to monostable mode. The monostable mode of the 555 timer is used for the capless hack which is discussed in part VI below.

## Capless hack

This special feature simulates a switch that appears broken or inactive for the first 4 seconds after system start or reset. It enhances the illusion and interactivity of the logic-controlled board. The behavior is achieved using an inverted 4-second timer signal (in the monostable mode discussed above), and works as follows:

* Initial Condition:

The timer output is initially HIGH (logic 1).

We invert this signal, so the circuit receives a LOW (logic 0) for the first 4 seconds.

* During the First 4 Seconds:

The switch appears to be in the OFF state, regardless of user input.

The connected LED does not respond, even if the switch is pressed or toggled.

This gives the illusion that the switch is not functional or has no cap.

The FSM (Finite State Machine) may stay in a holding state or ignore input from this switch.

* After the 4-Second Delay:

The timer output transitions to LOW (0), but after inversion, the circuit receives a HIGH (1).

The switch is now reactivated and begins to function normally.

Pressing or toggling the switch will now affect the corresponding LED as expected.

The FSM resumes normal processing for this input line.

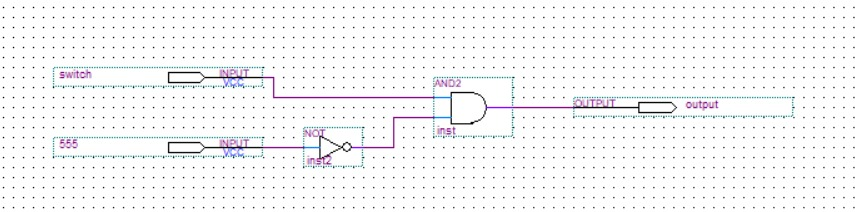


Figure 18 –The capless hack implementation on quartus

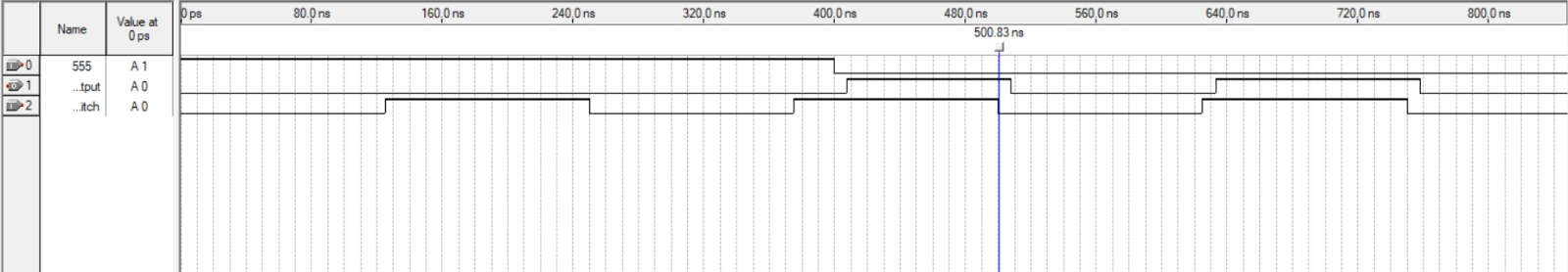


Figure 19 –The output of the capless hack

The obtained output mirrors the switch behavior only after 4s (which is the wanted trick).

Note that in Quartus, we set the 555 timer manually to 400ns to check if the concept is correct. However, in real life we will use the designed monostable timer.

A delay in the output is shown. This is dur to gates delay.

Delay from 500 ns to 508 ns = 8 ns

Delay from 625.411 ns to 632.867 ns = 7.456 ns

Delay from 751.245 ns to 757.749 ns = 6.504 ns

Average delay: (8+7.456+6.5043)/3=7.32 ns

Note: This input can be inserted in the sequence detector instead of the switch alone. In this way, we are deactivating the circuit for the first 4 sec.

## Circuit link

In this section, we linked the whole circuit together.

First, instead of the switch alone, we include the capless hack that used the 555 monostable timer. Instead of the selectors A and B we have created, we put the actual sequence detected by the sequence detector built.

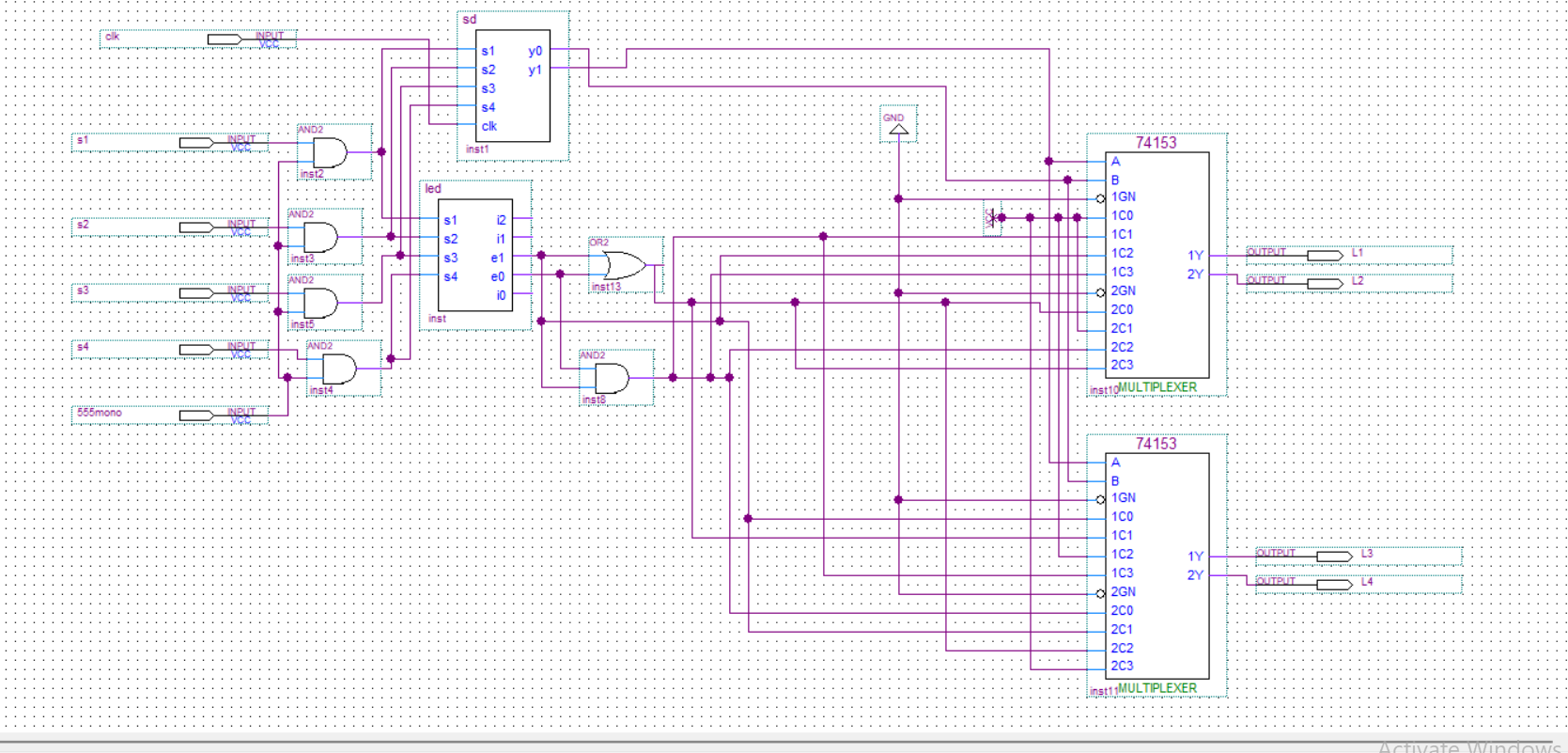


Figure 20 –The circuit link implementation on quartus

# Cost and Power consumption

Although we didn’t implement our design on a real breadboard, we looked into the prices of the components we will use and the power consumption to analyze them.

All details are shown in the table below.[1]

To calculate power consumption, we use the formula: P = V × I, where V is the supply voltage (Vcc) and I is the current drawn by the component (Icc)

Table 12 –Table of the Cost and Power consumption

|  |
| --- |
| **AND Gate (74LS08)**: **0.35$** |
| Typical power consumption: ~10 mW to 15 mW (depends on load and frequency of operation) |
| **OR Gate (74LS32)**:**0.5$** |
| Typical power consumption: ~10 mW to 15 mW (depends on load and frequency of operation) |
| **NOT Gate (74LS04)**:**0.5$** |
| Typical power consumption: ~5 mW to 10 mW |
| **D Flip-Flop (74LS74)**: **0.65$** |
| Typical power consumption: ~20 mW to 30 mW |
| **4x1 MUX (74LS157)**: **0.9$** |
| Typical power consumption: ~20 mW to 30 mW |
| **XOR Gate (74LS86)**: **0.45$** |
| Typical power consumption: ~15 mW to 20 mW |
| **Breadboard**: **180\*120mm for 3$** |
| Breadboards themselves don’t consume power. However, the power consumed depends on the components connected to them. |
| **Wires**: **50PCS for 2.5$** |
| Wires do not consume power by themselves; they just conduct current based on the components they're connected to. |
| **Resistors**: **0.02$** |
| Power consumption depends on the resistance and current flowing through them, but typically in the milliwatt (mW) range. (P = V²/R; varies by resistance and current) |
| **Capacitors**: **0.25$** |
| Capacitors do not consume power, but they store and release energy. Power consumption occurs during charging and discharging cycles, generally low unless the voltage is high. |
| **555 timer**:**1.35$** |
| Typical power consumption: ~1 mA to 10 mA at 5V (which is ~5 mW to 50 mW) |
| **Battery**: |
| Power consumption depends on the capacity and current drawn by the system. Common battery power ratings range from 100 mAh to 5000 mAh, with varying output voltage. |
| **Switches**: **0.75$** |
| Mechanical switches themselves do not consume power. They just open or close a circuit. |
| **LEDs**: |
| Power consumption depends on the type and color of the LED. Typical power consumption:  Standard LED: ~20 mA at 2V to 3V (usually around 50-60 mW)  High-power LED: Can consume 100 mW to 1W or more depending on brightness and design. |

The circuit will cost approximately up to 50 to 60 $ in our case.

To minimize this cost, we can simplify as much as possible our expressions to get minimal numbers of ICs. Moreover, before buying any equipment, we make sure that the circuit is correct after implementing it on the virtual breadboard. Also, we should try each IC to prevent burning other components.

Power consumption analysis:

In this project, power consumption varies significantly across components. Low-power logic gates such as AND, OR, NOT, and XOR typically consume between 5 mW to 20 mW each, making them suitable for efficient small-scale designs; however, using many of them can add up to substantial power usage. Moderate-power components like D Flip-Flops and 4x1 multiplexers consume 20–30 mW due to their clocked and control-based functions, thus contributing more significantly to the overall draw. The 555 timer shows variable consumption ranging from 5 to 50 mW, depending on load and configuration, and can become a major factor in timing-intensive applications. Passive components such as resistors and capacitors typically do not consume steady power—resistors dissipate energy based on the current and resistance, while capacitors draw minimal power during charging and discharging cycles. LEDs, although small, are among the most power-consuming elements, averaging 50–60 mW each; a few active LEDs can consume as much power as several logic gates combined. Lastly, elements like switches, wires, and breadboards do not draw power themselves but can affect system efficiency through resistance and voltage drops.

# Challenges

The project presented several significant challenges. At the beginning, we felt quite lost, as understanding the overall behavior of the system was not straightforward. The required approach demanded out-of-the-box thinking and exploring multiple strategies to make progress. We had to attempt several iterations to address persistent lags and bugs, which consumed considerable time and effort. The complexity of the design made it difficult to determine how the various components should be connected and interact with one another. Additionally, time management became a challenge due to the trial-and-error nature of our debugging process. Even now, some issues remain unresolved, reflecting the intricacy of the project and the steep learning curve involved.

# Advantages

Despite the challenges we faced, this project was highly beneficial in linking sequential and combinational circuits. It helped us better understand how the concepts learned in the course can be applied to real-life scenarios. Moreover, it demonstrated various approaches to problem-solving and highlighted different ways to tackle complex design issues.

# Conclusion

In conclusion, this project allowed us to integrate multiple digital design concepts into an interactive logic-controlled board. We successfully implemented a sequence detector to track switch activity, developed a capless hack to enable or disable switch functionality dynamically, and used the 555 timer in astable and monostable modes for timing and control. Additionally, we designed LED-based visual output, managed the activation sequence logic, and incorporated a **counter** to track events or system states. Throughout the development process, we encountered and overcame various challenges, but we managed throughout the project to benefit from several advantages due to this project such as linking sequential and combinational logic and demonstrating different approaches to problem-solving skills. Furthermore, we paid close attention to power efficiency, ensuring that the system operates with minimal power consumption by using logic gates and flip-flops selectively and disabling unused components when possible. Overall, this project not only reinforced our understanding of digital logic and circuit design but also highlighted the importance of combining functionality with user experience in embedded systems.

# References

*SN74LS04 data sheet, product information and support | TI.com*. (n.d.). <https://www.ti.com/product/SN74LS04?keyMatch=74LS04&tisearch=universal_search>