**Lebanese American University**

A black background with green text

Description automatically generated

***COE 322 – Logic Design Lab***

***Final Project Report***

Nazih Al Akoum 202307146

Moussa El Shami 202302471

Hadi Soubra 202307632

14/05/2025

# Abstract

This report details the design and hardware implementation of a fully discrete logic–based lamp mapping system that dynamically responds to switch inputs and preserves activation order without any microcontrollers. A finite state machine (FSM) processes falling‐edge off-pulses, generated via D-flip-flop edge detectors, to encode four switch inputs into a 2-bit state, driving a decoder for lamp sequencing. A ripple counter paired with a magnitude comparator offers a logic-based timed reset, later supplanted by a monostable 555-timer circuit to provide a precise 4 s impulse for state latching. The core mapping architecture comprises:

* **Order counting:** Pulses from any switch feed a 2-bit up-counter to record activation order.
* **Sequence lookup:** A dual multiplexer translates sequence and order bits into a 2-bit Lamp\_ID.
* **Per-switch memory:** D-flip-flops latch each switch’s Lamp\_ID on activation.
* **Lamp decoding:** Decoders and gating logic ensure each lamp lights only while its original switch remains on.

A boot-time mode selector stored in a D-flip-flop enables locked (direct mapping) or unlocked (FSM mapping) operation. Comprehensive financial and power consumption analyses quantify component costs and static power draw. Challenges with gate-level propagation delays and integrating the 555 timer are discussed, and a bonus scrolling 7-segment display feature illustrates additional functionality. This work showcases how foundational digital components can implement complex, reliable, responsive systems without software logic.

# Introduction

This project presents the full hardware design and implementation of a logic-controlled lamp mapping system, constructed solely using fundamental digital logic ICs. The system responds dynamically to physical switch inputs and adapts its behavior based on a boot-time mode selection either LOCKED or UNLOCKED. It uses finite state machines (FSMs), counters, multiplexers, decoders, and flip-flops to assign and control lamp activations in a defined sequence. Key challenges included sequencing activations based on switch order, ensuring each switch remembers the lamp it turned on, and correctly driving outputs in both operational modes. The project adheres strictly to gate-level implementation without microcontrollers or software logic, making it a strong demonstration of applied digital design principles.

# Equipment Used

1. Software:

* Alteras Quartus II

1. Components Used on Alteras Quartus II:

* D-Flipflops
* T- Flipflops
* Inverters
* Or Gates
* And Gates
* 74157: Multiplexer (a MUX containing 4 2:1 Multiplexers with a common selector)
* 74139: 2:4 Decoder
* 7495: Shift Register
* 7447: BCD TO 7 Segment decoder

# Sequence detector

## Paper design:

In this part, the objective is to analyze the state transition logic of a sequence detector that is governed by the falling edges (off-pulses) of four input switches, denoted as SW₁ through SW₄, and represented respectively by the inputs A, B, C, and D. Each off-pulse triggers a transition into one of four sequences, each encoded by a 2-bit binary state (,) . Specifically, an off-pulse from SW₁ (when A = 1) results in a transition to the binary state 00 (Sequence 1). Similarly, an off-pulse from SW₂ (B = 1) transitions the FSM to 01 (Sequence 2), from SW₃ (C = 1) to 10 (Sequence 3), and from SW₄ (D = 1) to 11 (Sequence 4).If none of the switches are turned off, meaning that A = B = C = D = 0, the FSM maintains (latches) its current state. The next-state logic is mathematically expressed using the following Boolean equations:

Where E = A + B + C + D denotes the total off-pulse activity.These logical expressions guarantee that when no falling edge (off-pulse) occurs on any of the inputs, the current state remains unchanged due to the latching effect of the feedback terms and . Conversely, upon detection of a specific off-pulse, the state transitions to the corresponding binary value associated with that particular input switch, ensuring proper state encoding based on sequence selection.

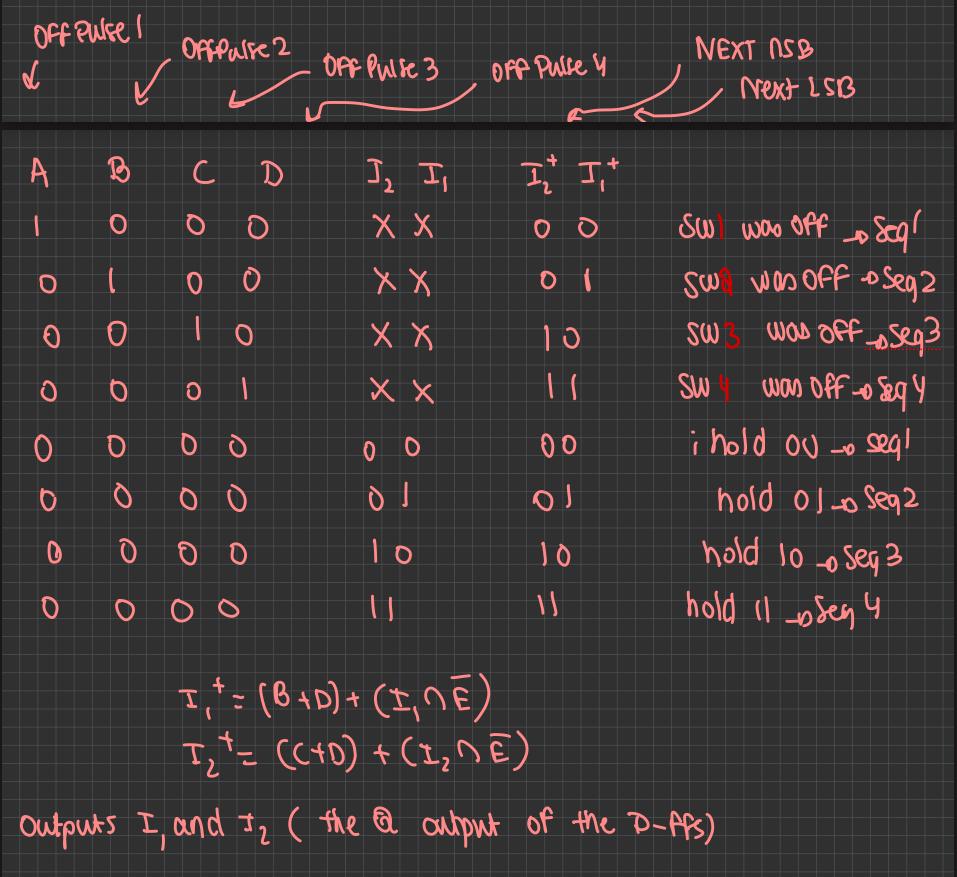


Figure 1– Paper Design / Truth Table for the Sequence Detector

## Altera Quartus II:

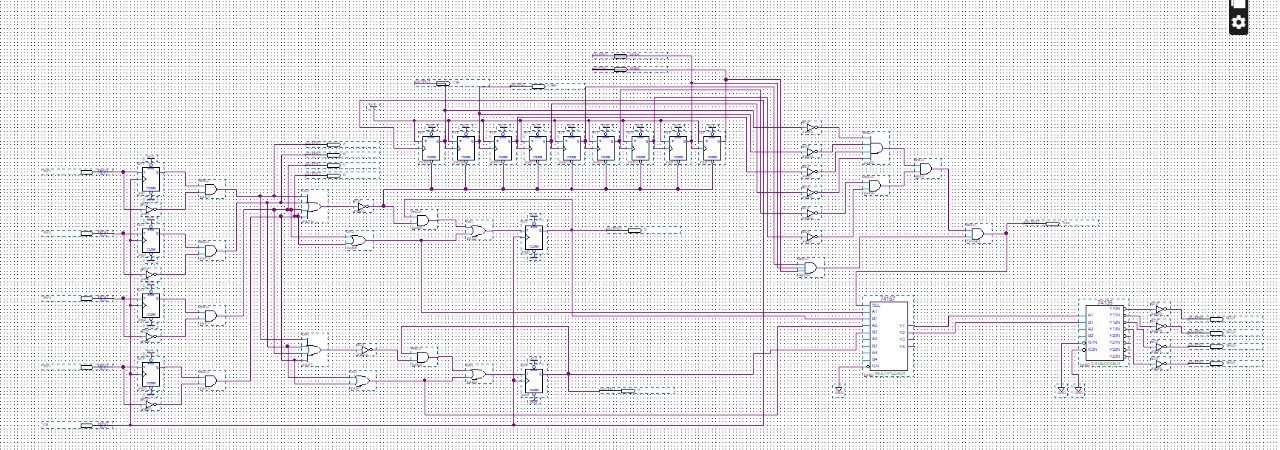


Figure 2 – Full Sequence Detector Circuit

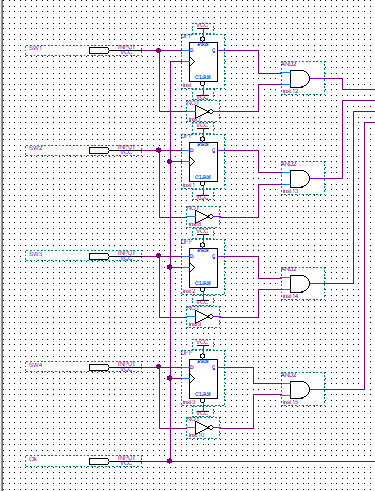


Figure 3 - Off - Pulse Detection

In this part, the objective is to generate off-pulses corresponding to each input switch (SW₁ to SW₄) through an edge-detection circuit. The purpose of this circuit is to detect the falling edge, which is the transition from logic high to logic low, of each switch signal. This is achieved by utilizing a D-type flip-flop, where the input D is directly connected to the corresponding switch signal. The clock input of the flip-flop is shared among all switches to ensure synchronized operation.The output Q of the flip-flop retains the previous state of the switch, effectively storing its last logic level. A NOT gate is then used to invert the current switch signal, and the inverted value is ANDed with the stored previous value Q. The output of this AND gate momentarily goes high when the switch transitions from 1 to 0, indicating that the user has released the switch.As a result, a pulse, termed as an off-pulse, is generated for each switch and is labeled respectively as A, B, C, and D. These off-pulses serve as critical inputs to the finite state machine (FSM), enabling it to detect the activation of a particular switch and trigger the appropriate transition in the sequence state.

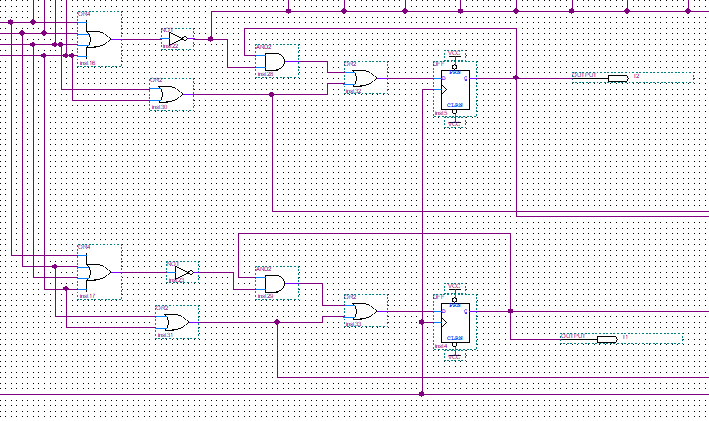


Figure 4 – Sequence Detection

This part was already explained in the Sequence Detector Paper Design

A computer diagram with many lines

AI-generated content may be incorrect.

Figure 5 - Bonus Part : Ripple Counter + Magnitude Comparator

Bonus Implementation (New Functionality) – Timed Reset Signal Using Ripple Counter and Magnitude Comparator Logic

In this part, we implemented a bonus feature to generate the control signal using a fully automatic 9-bit ripple counter combined with a magnitude comparator. The aim of this design was to eliminate the need for a manual timer or external clock input and instead use logic-based timing with internal clearing conditions governed by system activity.The ripple counter was constructed using a series of nine T flip-flops. The first T flip-flop receives a 100 Hz clock input, resulting in toggling every 10 ms. Each subsequent T flip-flop receives its clock input from the Q output of the previous flip-flop, forming a ripple configuration. With each T flip-flop set to toggle on every rising edge when T=1 , this configuration creates a binary counter that increments once every 10 ms.To ensure that the counter resets upon any switch activity, we connected the asynchronous clear input of all T flip-flops to the signal E = A + B + C +D, where A,B,C,D represent the four switch inputs. This configuration ensures the following behavior:-When E=1 (at least one switch is pressed), the counter is immediately cleared and held at zero.-When E=0 (no switches are pressed), the counter is allowed to increment, effectively starting a 4-second countdown.The 9-bit counter was chosen specifically to accommodate the value 400 in binary (400₁₀=110010000₂), which corresponds to a 4-second delay since the counter increments every 10 ms.To detect when the counter reaches 400, a custom magnitude comparator was designed using logic gates:

-The bits that are 0 in the binary representation of 400 (Q6, Q5,Q3, Q2,Q1,Q0) were passed through NOT gates and ANDed together.-The bits that are 1 (Q8,Q7,Q4) were directly ANDed together.-The outputs of these two AND gates were then combined using a final AND gate to produce the control signal .

This design ensures that is asserted high only when all bits match the exact binary pattern of 400, indicating 4 seconds of inactivity.This fully automated timing mechanism enables the FSM to detect whether user inactivity, exceeds 4 seconds (for the case where my switches are off), in which case it may re-enter detection mode or reset its internal state as appropriate.

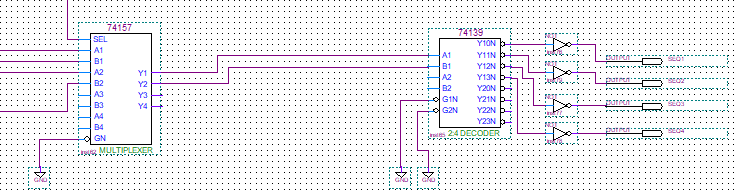


Figure 6 – + Multiplexer and Decoder

In this part of the system, we implemented the logic responsible for controlling the FSM’s state transitions by managing both the state encoding and the decoding processes. To achieve this, we utilized the 74157 integrated circuit, which is a 2-to-1 multiplexer. This component was used to select between two sets of 2-bit input signals: the stored state bits and the updated transition state bits. The selection is controlled by the signal .

When the multiplexer selects the stored state bits, maintaining the FSM in its current sequence. This ensures that the FSM remains latched and does not transition. However, when , the multiplexer switches to the updated state inputs, allowing the FSM to transition to the next state in the sequence. The selected output of the multiplexer, representing the current or next 2-bit state, is then fed into a 74139 2-to-4 decoder. This decoder generates four active-low output signals, labeled SEQ₁ to SEQ₂, with only one signal active at any given time. These decoded outputs are used to drive the corresponding lamp outputs in the system and represent the currently active sequence. This configuration simplifies the overall logic required for visual sequence indication by ensuring one output control. Only one output is low at a time, which guarantees unambiguous activation of a single lamp corresponding to the current state of the FSM.

Following the creation of our schematic, we compiled and simulated the design to ensure it operates correctly in a digital simulation.

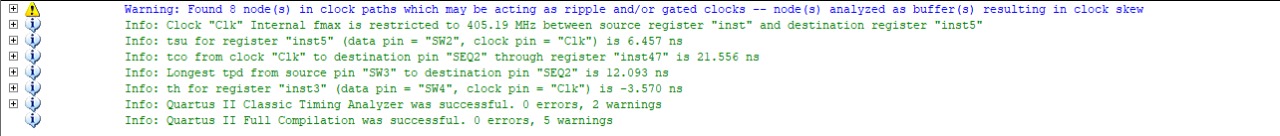


Figure 7 - Successful compilation of the Implemented Circuit Design +

After a successful compilation of our circuit, we need to simulate it. We start by opening a Vector Waveform File and use the node finder to add all the inputs to the waveform table.

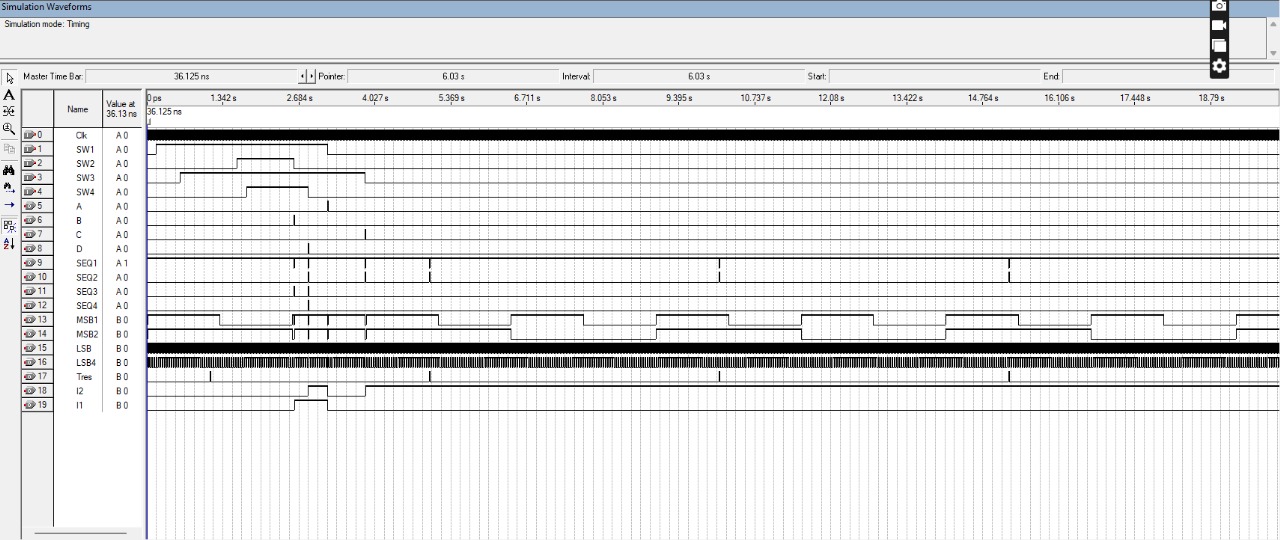


Figure 8 - Simulation of the Implemented Circuit Design +

The simulation results confirm that the output of the sequence bits and ​ accurately followed the expected behavior, matching the correct binary sequence as designed. The system's state transitions were functionally correct, and the logic for updating the FSM operated as intended.

However, we observed that the decoded outputs SEQ₁, SEQ₂, SEQ₃, SEQ₄ experienced significant delays. This delay was directly attributed to the timing of the signal in the original design. Since was generated using a 9-bit ripple counter and a magnitude comparator composed of multiple T flip-flops and logic gates, the overall propagation delay was exceptionally high. As a result, each transition of the SEQ outputs was delayed by approximately 10 seconds.Due to this excessive delay, the original design was deemed not feasible for practical implementation. Therefore, we redesigned the circuit to utilize a manual clock input to drive the counter. This revised approach significantly reduced the timing delay and allowed us to achieve immediate responsiveness during testing and verification, as presented in the updated circuit implementation down below.

# Sequence detector with 555 timer (T-reset)

In the previous part, we implemented a custom counter using T flip-flops. While this implementation was successful, it had numerous errors due to internal circuit delays and the inflexibility of delay propagation. To achieve a more accurate and intuitive design, we decided to use a 555 timer to generate a 4-second impulse when all switches are off. This way, once the switches are off, we start counting the 4 seconds, and upon completion, we can accurately update the final two registers to obtain the true sequence we need.

A diagram of a computer

AI-generated content may be incorrect.

Figure 9 - Sequence detector with T-reset

The circuit above represents the theoretical implementation of our idea. The T-reset here is connected to the last two D flip-flops. Essentially, the first two D flip-flops function as previously described, dynamically updating as long as at least one switch is on, reflecting the sequence we are transitioning into. The last two D flip-flops only update when the T-reset receives a negative edge, which corresponds to the 555-timer impulse. This timer provides a 4-second impulse when all switches are off, and on the negative edge, it causes the last two D flip-flops to update, saving the sequence from the first two flip-flops. The last two D flip-flops store the state as a 2-bit binary value, which then needs to be decoded for further use in the circuit, similar to how the previous circuit's output was decoded.

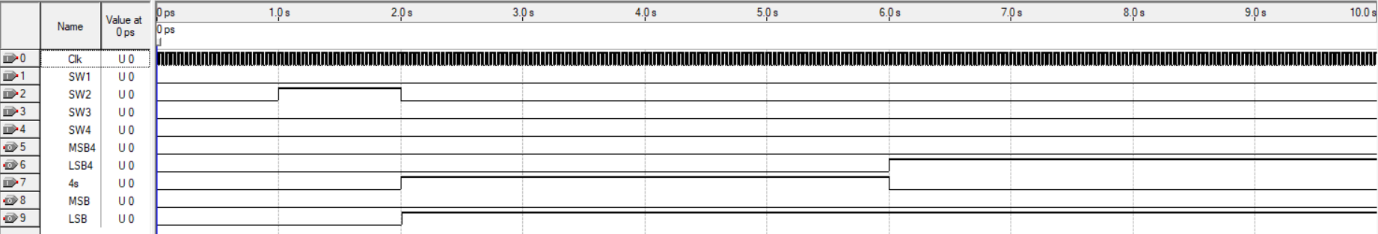


Figure 10 - circuit with T-reset simulation

In the simulation we ran, we observed that once the switch is turned off (goes to zero), the system transitions into state one, represented by the binary bits 01—where 0 is the most significant bit and 1 is the least significant bit. The preliminary most significant and least significant bits update immediately when the switch turns off. However, the final, correct values for the most significant and least significant bits only update after 4 seconds, or when the T-reset signal goes to zero after counting down those 4 seconds. This ensures that the final state is stable and accurate.

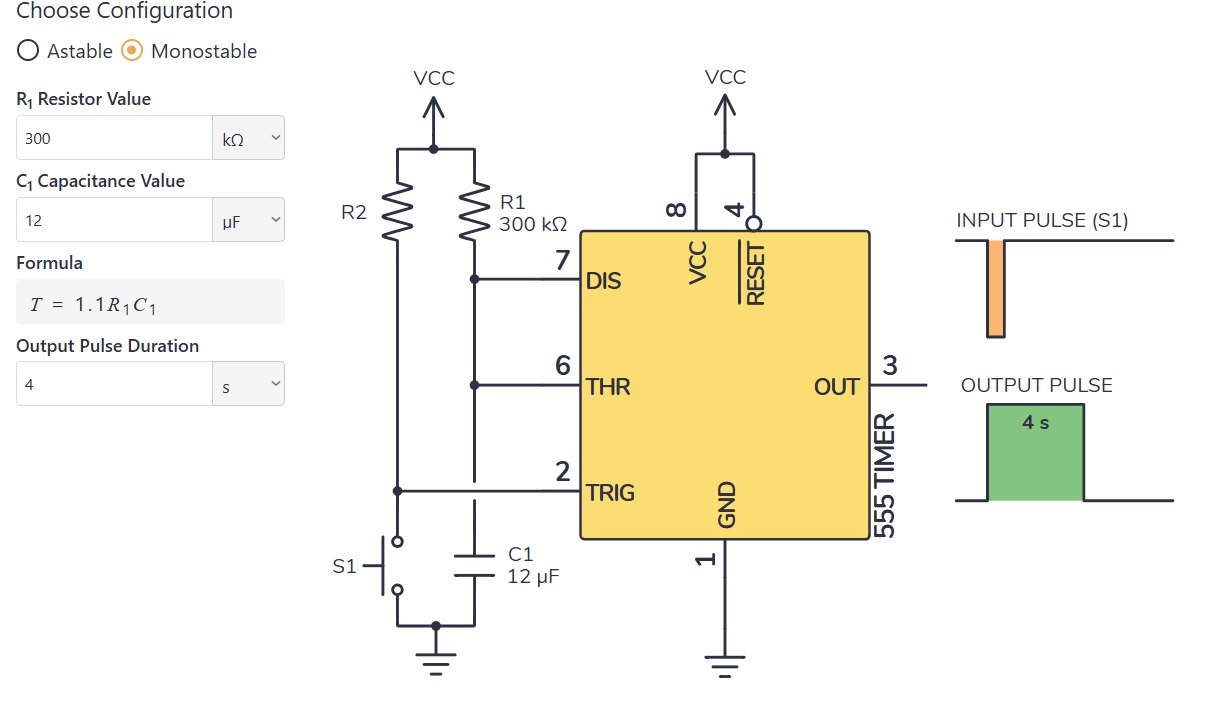


Figure 11 - 555 timer implementation

In the monostable mode, the 555 timer takes a negative impulse, where the signal transitions from 1 to 0, and outputs a positive impulse, from 0 to 1, with a duration that can be calculated using the formula

Using equation 1 to get T=4s we get the following values:

We also note that in place of switch (S1) in figure 3, this is where we should input our negative impulse, which transitions from 1 to 0. This negative impulse is created by feeding the signals of all the switches into a NAND gate and then using the output of that NAND gate in place of the switch. Thus, when all the switches are off (set to zero), the output of the NAND gate goes from 1 to 0, triggering the 555 timer and starting the 4-second count. The timer resets every time a switch changes, ensuring that the 4-second countdown only proceeds when all switches remain off continuously.

To wrap everything together, by using the 555 timer we just implemented in the first circuit and replacing the manual T-reset input, we achieve a more precise and reliable output. This setup provides the same desired sequence detector functionality but with minimal delay and more accurate results, enhancing the overall performance of the circuit.

# Lamp mapping and Activation sequence

In this part, we will implement the core “trick” of the board: no matter which switch you flip first, second, third or fourth, it will always light the correct lamp in the current sequence. We break the problem into four modular blocks—order counting, sequence lookup, per‐switch memory, and final lamp decoding—so that each switch “grabs” a lamp index when it turns on and later frees exactly that lamp when it turns off.

## Thought process:

1. **Activation‐order**: We need a 2-bit binary counter (0,1,2,3) that increments once each time any switch goes from OFF→ON, then resets when all are OFF.
2. **Sequence lookup**: Given a 2-bit sequence index (4 possibilities) and a 2-bit order count, we must output a 2-bit Lamp\_ID (00 for lamp 1/01 for lamp 2/10 for lamp 3/11 for lamp 4) according to the table for Sequence 1–4.
3. **Per-switch storage**: Each switch latches the Lamp\_ID it received on its rising‐edge, so it “remembers” which lamp to turn off later.
4. **Lamp drive**: Compare each stored ID against the four lamp numbers and gate the physical LED only while the switch remains ON.

## Circuit implementation

**Stage 1 – Counting the Switch-On Order**  
Each time any switch *n* goes from OFF to ON, a small edge-detector circuit (a D-flip-flop sampling the switch plus an AND gate) produces a single-cycle pulse, “EDGE\_SWn”. All four of these pulses are combined in an OR gate to form “**COUNT\_ENABLE**”, which clocks a two-bit up-counter. The counter’s outputs are **ORD1** (MSB) and **ORD0** (LSB): on the first switch-on you get ORD = 01, on the second ORD = 10, then ORD = 11, and finally it wraps back to ORD = 00. Whenever all four switches are turned OFF, an **ALL\_OFF** signal asynchronously clears both flip-flops in the counter, resetting ORD to 00 for the next cycle.

**Stage 2 – Finding the Lamp ID**  
We also have a separate input that picks one of four preset sequences. That one bit sequence code (example 0001 for sequence 1) is turned into a two-bit number SEQ\_BIN1 (MSB) and SEQ\_BIN0 (LSB). We feed **SEQ\_BIN1** and **SEQ\_BIN0** together with ORD1, ORD0 as the four select lines of a dual 4-to-1 multiplexer (74LS153). The top half of the mux outputs the Lamp\_ID MSB (**LAMP\_ID1**), and the bottom half outputs the Lamp\_ID LSB (**LAMP\_ID0**). Depending on which sequence and which order count you’re at, the mux produces exactly the two-bit lamp number (00,01,10,11) that corresponds to that step.

**Stage 3 – Each Switch Remembers Its Lamp**  
To “remember” which lamp each switch lit, each switch *n* drives two D-flip-flops (74LS74) clocked by **EDGE\_SWn** and reset by **ALL\_OFF**. On the rising edge of SWn, those flip-flops sample LAMP\_ID1 and LAMP\_ID0 and store them as **MEM1** (MSB) and **MEM0** (LSB). From that moment until the switch is released, those two bits hold the index of the lamp this switch activated.

**Stage 4 – Driving the LEDs**  
Finally, for each switch we take its two-bit stored Lamp\_ID (as MEM1 and MEM2) and feed it into a small 2-to-4 decoder (74LS139). Its active-low outputs Y0N… to Y3N are inverted to form four one bit lines. We AND each of those lines with the switch’s current ON/OFF state so the switch only drives its lamp while it’s held ON. Then all switches’ results for each lamp are OR’ed together so that Lamp 1 lights if any switch mapped to “1” is ON, Lamp 2 lights if any switch mapped to “2” is ON, and so on. This guarantees that turning a switch OFF always only turns off the specific lamp it first turned on, perfectly preserving the sequence.

Now we will implement the above in a corresponding circuit:

## Altera Quartus II:

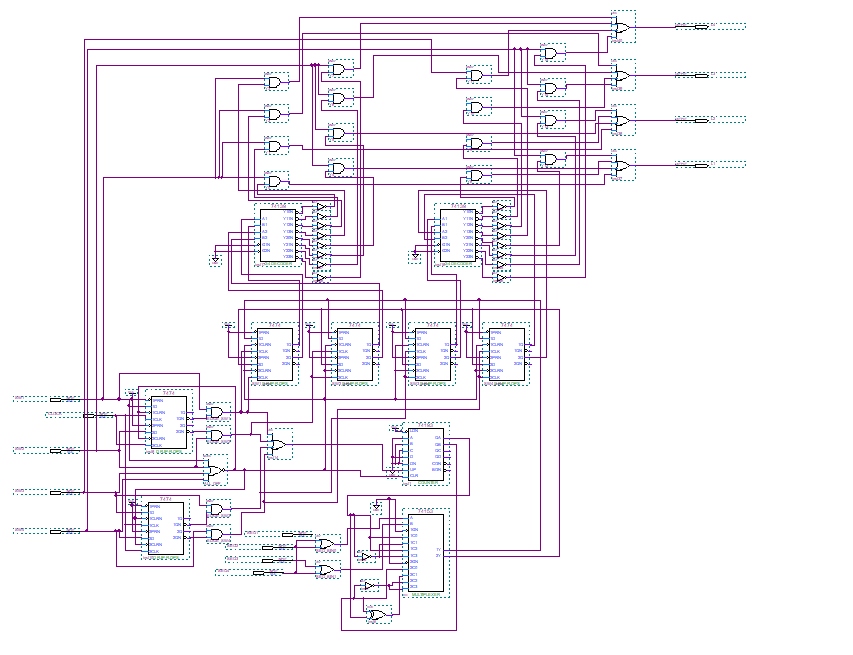


Figure 12 - Overall view of the lamp mapping system

The figure above shows the outline of the finalized lamp mapping system. We will dig deeper into the circuit and show each part of the system.

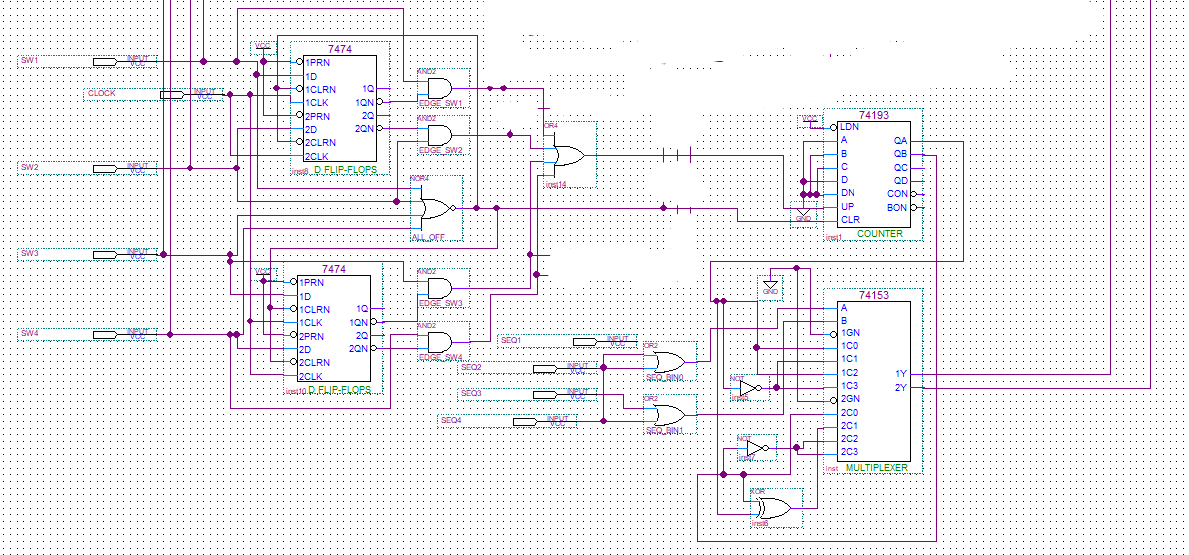


Figure 13 - Circuit implemented to get the lamp IDs using the counter and sequence

The figure above shows the combination of the counter we used that accepts the **COUNT\_ENABLE** as input for the up count and resets when all switches are 0. The counter increments whenever an input switch pulse is received. Then, this counter with a combination with the sequence obtained using the LSB and MSB, form the selectors for our multiplexer, which output will be the lamp ID stored in two bits (MSB, LSB).

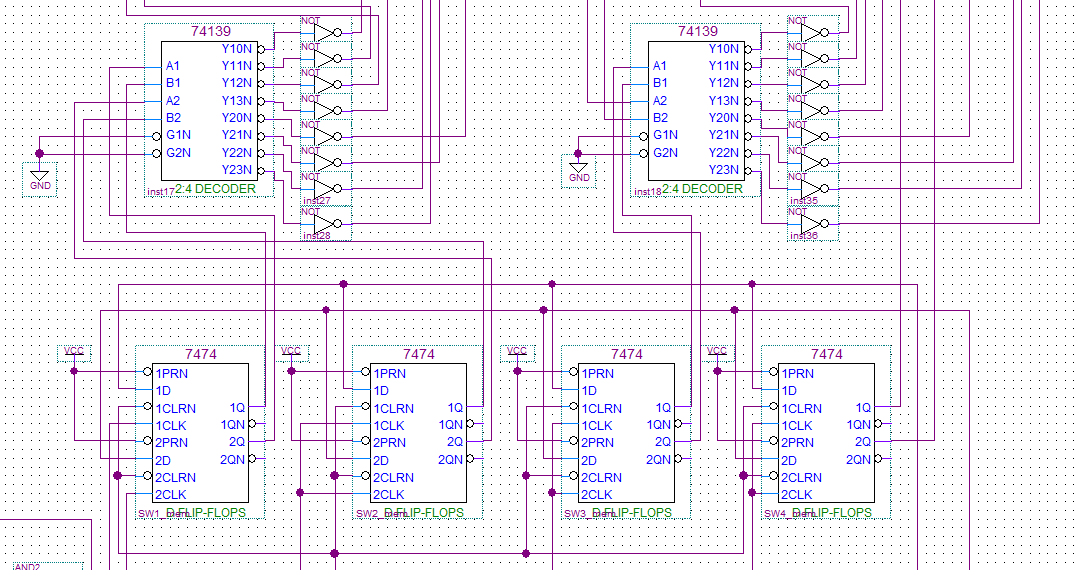


Figure 14 - Lamps IDs stored into flipflops and then sent to the decoders

Now, the lamp IDs are fed into 8 different flipflops, where we used the clock for the first 2 flipflops containing the LSB and MSB of the lamp id that should turn on, as an impulse of each switch respectively, thus first 2 flipflops having a clock of impulse Switch 1, and the second 2 having a clock of impulse Switch 2 and so on. Now, after a switch impulse has been input, it updates the corresponding D flipflop so that it remembers which switch has updated it, then it outputs the Lamp IDs to the decoders we used.

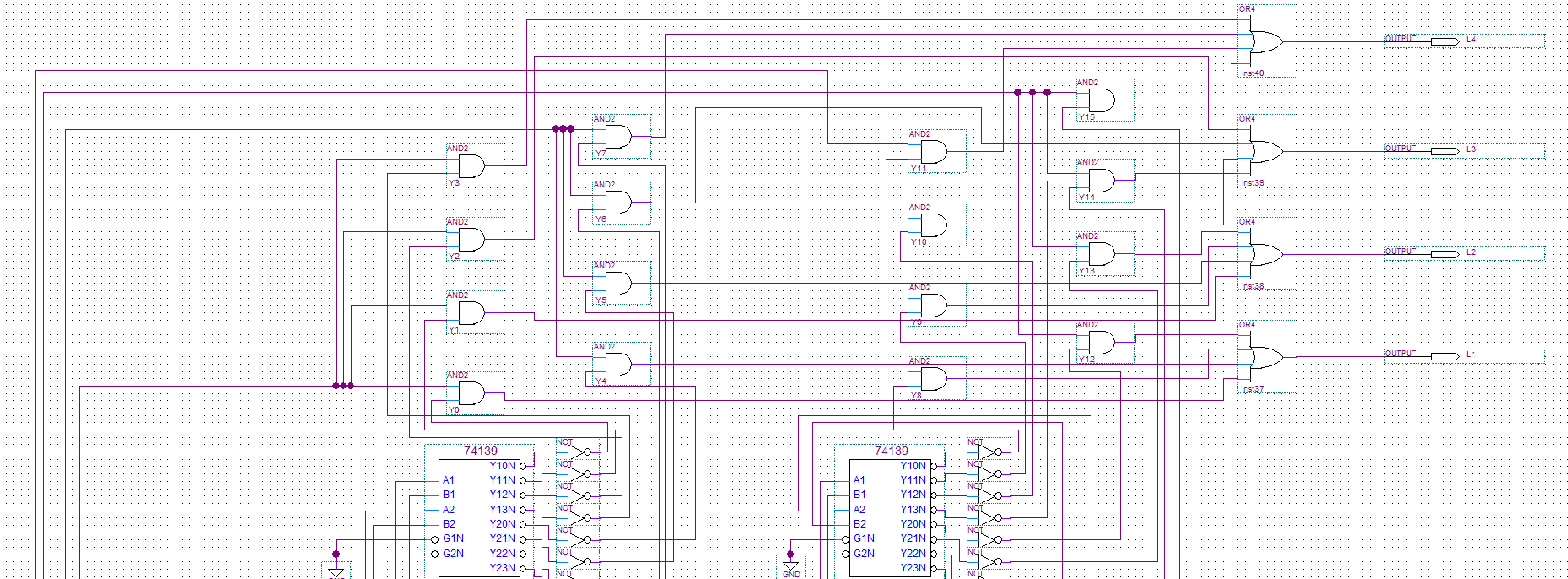


Figure 15 - Decoders' outputs implemented for driving the leds

Finally, the decoders’ outputs were inverted since it’s an active low IC, we AND each of those lines with the switch’s current ON/OFF state so the switch only drives its lamp while it’s held ON. Then all switches’ results for each lamp are input into an OR together so that Lamp 1 lights if any switch mapped to “1” is ON, Lamp 2 lights if any switch mapped to “2” is ON, and so on.

This guarantees that turning a switch OFF always only turns off the specific lamp it first turned on, perfectly preserving the sequence and correctly mapping the lamps to turn on with respect to the sequence we’re in and the number of switches that are on.

# Locking and Unlocking Mechanism

The system can be locked to ensure the audience cannot detect any patterns in the switch-lamp sequence.

How it works:

* **Booting:** Initial power-on state. The system checks whether Switch 2 is ON to determine whether to enter the normal FSM flow or lock mode.
* **Locked:** Activated when the system is booted with Switch 2 ON. In this state, each switch is directly mapped to its corresponding LED (1→1, 2→2, etc.), and no dynamic behavior occurs.
* **Unlocked:** Activated when the system is booted with switch 2 OFF. In this state, the FSM is implemented when turning the lamps on and off.

## Thought process:

To solve this problem, we will find a way for both systems to work in parallel, hence we will not check first if the system is booted with Switch 2 on or off then choose which state to follow, rather we will run both systems simultaneously, and then at the output we will choose which state (locked or unlocked) will determine the output of the lamp.

First, we need to store the initial switch 2 when booting the system and store it in a D flipflop in order for our system to know in which state to operate. We want this flipflop to store only the booting value of Switch 2, then we don’t want it to update while using the system. Thus, we will use a clock that consists of a 100 uF capacitor and a 2k ohm resistor so that our capacitor fully charges and stops the clock pulsing after 1 second (5 Tao = 0.2 x 5 = 1s). Now we will take the output of this flipflop and connect it to an AND gate, with the second input being a Switch (take switch 1 for example), then we will take the output of this flipflop again, invert it then AND it with the output L1 of the previous circuit which is the lamp mapping. And then both outputs of these and gates will be connected to an OR gate, and the output of this OR gate will be our final Lamp1 output pin, where it will be connected directly to the lamp 1 in the circuit. Finally we will do the same steps for all 3 other switches, taking every time the same output of the one flipflop we used. In this way, if switch 2 was initially on when booting the system, the output of the LAMP1 will be switch 1, thus it will be directly mapped, which represents the locked state. However, if Switch 2 was initially off, the output will be the FSM Lamp1 output which would represent the unlocked state.

Now we will draw the corresponding circuit:

## Altera Quartus II:

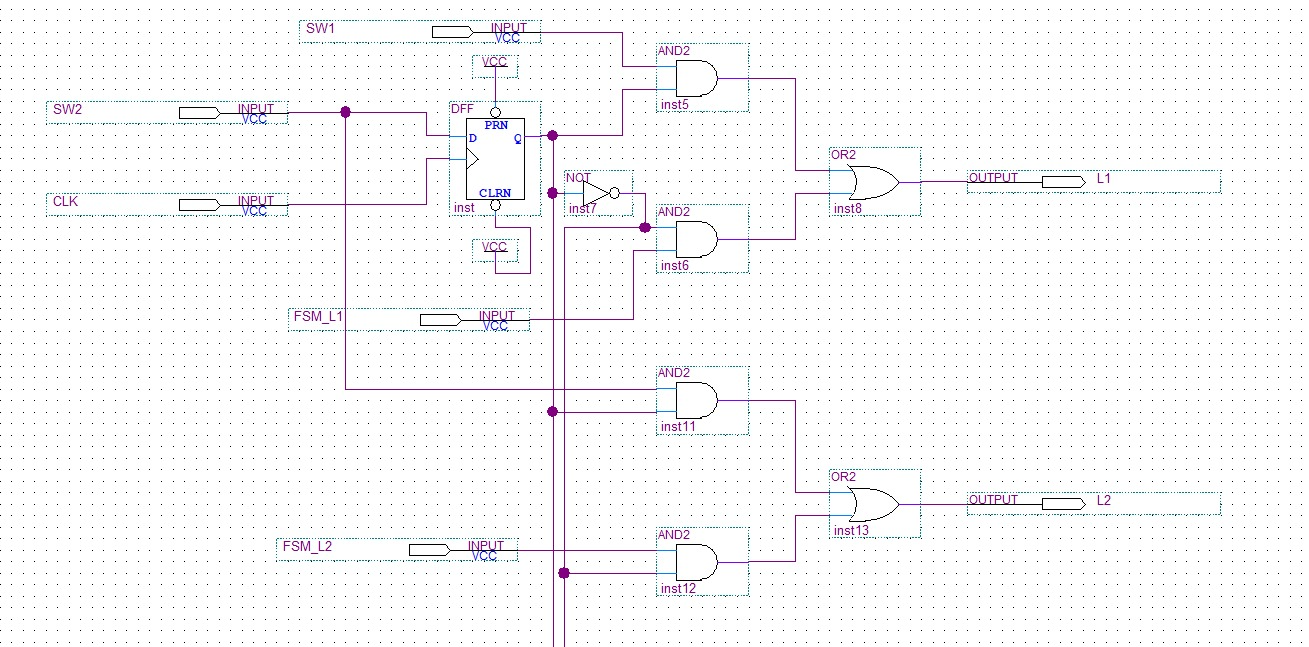


Figure 16 - Booting system for lamps 1 and 2

The figure above shows the implementation we described with switches 1 and 2, showing how the output of Lamps 1 and 2 will choose either from SW1 (Switch 1) or FSM\_L1 (the output of L1 in the lamp mapping circuit). Same for Switch 2.

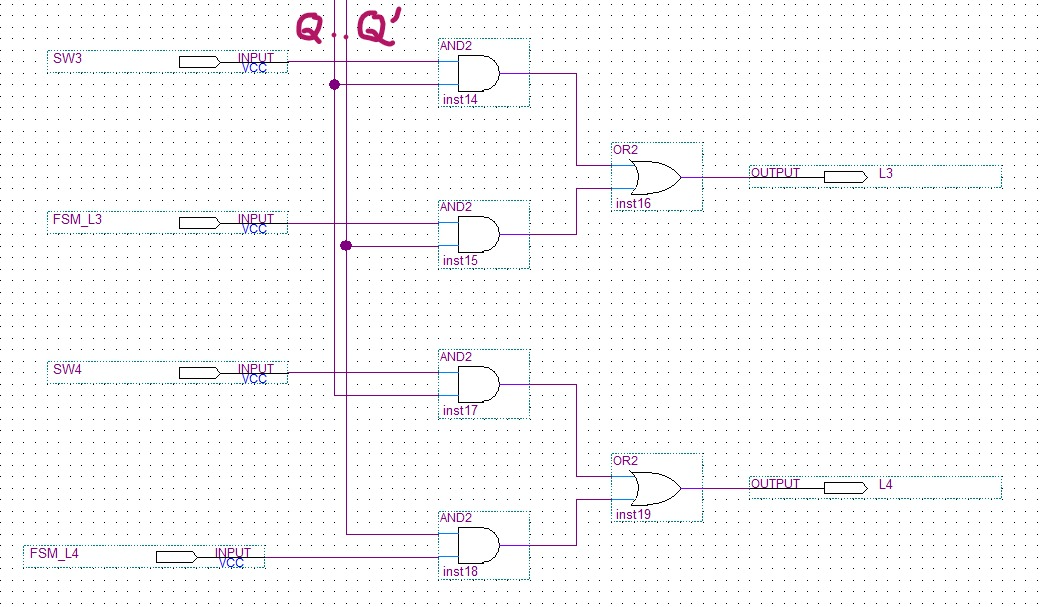


Figure 17 - Booting system for lamps 3 and 4

The figure above shows the same implementation described above but for switches 3 and 4.

Now we will show a simulation example for lamp 1 of how this circuit works.

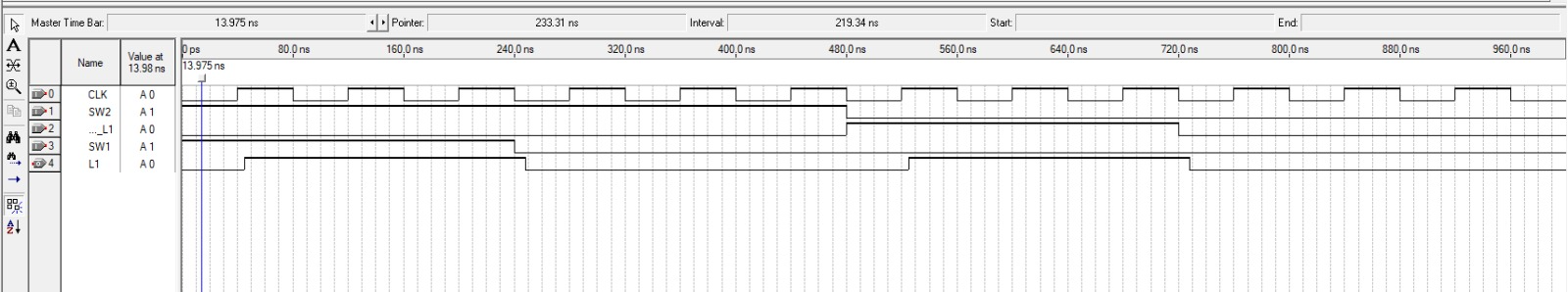


Figure 18 - Booting system simulation for lamp 1

The figure above shows the simulation of the Booting circuit in Altera Quartus II, showing that at the positive edge of the clock, when the D flipflop stores the initial value of Switch 2, when we had Switch 2 high (On when booting), this means that we should boot in locked mode and the lamp L1 should take the value of Switch 1, which is clearly shown between the intervals 40ns -> 240ns. And when the Switch 2 was turned off at the positive edge of the clock which corresponds to the case where we boot our system with switch 2 off, it will load in the unlocked state, which means that the Lamp L1 will take the value of the FSM L1 output from the lamp mapping circuit, which is clearly shown between the intervals 520 ns -> 720ns.

Additionally, we notice a bit of delay in the output lamp which is due to the internal propagation delay of the logic gates.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 19 - Booting system simulation for lamp 1 delay

To study the delay, first we can check that the rising edge of the clock was at 40ns, thus now since we have SW2 initially on, L1 should have the same value as SW1 (value of 1), however we noticed that it didn’t directly follow it;

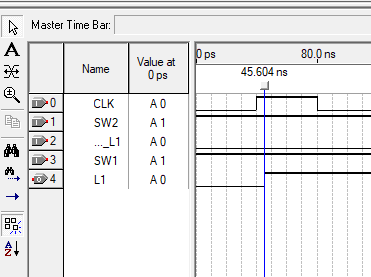


Figure 20- Booting system simulation for lamp 1 delay

From the picture above, we can see that L1 switched from a value of 0 to 1 at 45.6 ns, thus we can calculate the delay to be of 5.6ns.

# Financial study:

## Sequence Detector Financial Study

Total IC Needed to implement the Sequence Detector Circuit + :

* 3 74LS74 IC D flipflops
* 3 74LS08 AND IC
* 2 74LS04 NOT IC
* 1 74LS32 OR IC
* 5 IC T Fliflops -> Converted into 5 IC D flipflops + 1 IC Not
* 74157 4 2:1 Multiplexer
* 74139 Decoder 2:4

Considering the unit prices of each required IC and the total quantity used in the implementation, we now calculate the overall financial cost of the complete Sequence Detector circuit, including the​ logic. The breakdown of individual IC costs is provided above, leading to the following total:

Total Cost = (8×0.70)+(3×0.70)+(3×0.70)+(1×0.70)+(1×1.00)+(1×1.00)

Total Cost=5.60+2.10+2.10+0.70+1.00+1.00= $12.50

## 555 circuit:

The financial analysis will focus solely on the 555-timer circuit itself and not on the entire circuit. This is because the financial study for the rest of the circuit was already completed in the previous part. In this section, the only modification is the addition of the 555-timer circuit, so we will only detail its costs and not repeat the calculations already made.

* 555 Timer IC: Usually around $0.20 to $0.50 per unit.
* Resistors (300 kΩ and 10 kΩ): $0.10 each.
* Capacitor (12 µF): $0.10 to $0.20.

Total cost: around $0.40 to $0.80

## Mapping circuit

* 6 × SN74LS74AN @ $0.63 each → $3.78
* 1 × SN74LS153N @ $1.09 → $1.09
* 1 × SN74LS193 @ $0.73 → $0.73
* 2 × SN74LS139AN @ $0.79 each → $1.58
* 5 × SN74LS08N @ $0.74 each → $3.70
* 5 × SN74LS32N @ $0.71 each → $3.55

Total cost = 3.78 + 1.09 + 0.73 + 1.58 + 3.70 + 3.55 = $14.43

## Boot-lock circuit:

Each Dual D flipflop IC will cost approximately $0.63, the 74LS08 will cost around $0.74 and the 74LS08 will cost approximately $0.71, which will bring the total to $2.82.

# Total Power Consumption:

## Sequence Detector circuit:

Total IC Needed to implement the Sequence Detector Circuit + :

* 3 74LS74 IC D flipflops
* 3 74LS08 AND IC
* 2 74LS04 NOT IC
* 1 74LS32 OR IC
* 5 IC T Fliflops -> Converted into 5 IC D flipflops + 1 IC Not
* 74157 4 2:1 Multiplexer
* 74139 Decoder 2:4

Now calculating the power of each one:

* Typical static power for the 74LS74 IC D flipflops:

P = = 8 5V 4mA= 160mW

* Typical static power for the 74LS08 AND IC:

P = = 3 5V 4.8mA= 72mW

* Typical static power for the 74LS04 NOT IC:

P = = 3 5V 4.5mA= 67.5 mW

* Typical static power for the 1 74LS32 OR IC:

P = = 5V = 31 mW

* Typical static power for the 74157 4 2:1 Multiplexer:

P = = 5V = 80 mW

* 74139 Decoder 2:4:

P = = 5V = 30 mW

Total Power Consumption of the circuit would be: 160 + 72 + 67.5 + 31 + 80 + 30 = 440.5 mW

## 555 circuit:

The Power analysis will focus solely on the 555-timer circuit itself and not on the entire circuit. This is because the Power study for the rest of the circuit was already completed in the previous part. In this section, the only modification is the addition of the 555-timer circuit, so we will only detail its costs and not repeat the calculations already made.

The 555 timer in monostable mode typically draws around 3 to 5 mA from the supply.

Using a 5 V supply, the power consumption would be approximately 15 to 25 mW using the power equation:

## Mapping circuit:

* 74LS74 (dual D-flip-flop)  
  P = VCC × ICC ≈ 5 V × 4 mA = 20 mW  
  6 chips → 6 × 20 mW = 120 mW
* 74LS153 (dual 4:1 MUX)  
  P ≈ 5 V × 6.2 mA = 31 mW
* 74LS193 (4-bit counter)  
  P ≈ 5 V × 19 mA = 95 mW
* 74LS139 (dual 2-to-4 decoder)  
  P ≈ 5 V × 6 mA = 30 mW  
  2 chips → 2 × 30 mW = 60 mW
* 74LS08 (quad AND gate)  
  P ≈ 5 V × 4.8 mA = 24 mW  
  5 chips → 5 × 24 mW = 120 mW
* 74LS32 (quad OR gate)  
  P ≈ 5 V × 6.2 mA = 31 mW  
  5 chips → 5 × 31 mW = 155 mW

Total ≈ 120 + 31 + 95 + 60 + 120 + 155 = 581 mW (≈ 0.58 W)

## Boot-lock circuit:

* Typical static power for the 74LS74 flipflop IC:  
  P = VCC×ICC ≈ 5V × 4 mA = 20 mW
* Typical static power for the 74LS08 IC:

P ≈ 5 V × 4.8 mA ≈ 24 mW; Now we have 2 of them so it will be 48mW

* Typical static power for the 74LS32 IC:  
  P ≈ 5 V × 6.2 mA ≈ 31 mW

The total power consumption of this circuit would be 20+48+31=99 ≈ 100mW

# Problems Faced

During the design and implementation of the sequence detector, several challenges were encountered. Initially, the tick counter introduced significant delays, leading to incorrect outputs despite the correct current state of the sequence. The updated version with the T-reset also suffered from inaccuracies due to these delays. When implementing the 555 timer, the main challenge was the inability to fully integrate it into the circuit design; instead, the clock signal had to be manually input during simulations. Thus, the 555 implementation remained theoretical, although it should function as intended. Meanwhile, the lock and D-lock components were implemented smoothly without issues. Lastly, the mapping process was complex and introduced additional delays, making it challenging to design but ultimately successful.

Overall, these challenges highlighted the importance of minimizing delays for accurate sequence detection and demonstrated the need for a practical implementation of the 555 timer to ensure the circuit’s reliability and performance. Despite these obstacles, the final design achieved the desired functionality with improved accuracy and efficiency.

# Key Design Points Advantages

One of the key advantages of our design is the lock system, which efficiently utilizes a single circuit instead of requiring separate circuits for the locked and unlocked states. This streamlined approach makes the design both efficient and optimal, reducing complexity and resource usage. Additionally, the mapping component offers a significant advantage by enhancing the illusion's prominence and making it more intriguing to the viewer. These advantages contribute to a more streamlined, effective, and engaging design.

# Bonus questions

## Scrolling 7 Segment Display for Custom Messages + Implementation of a new Functionality (2 Bonuses in one)

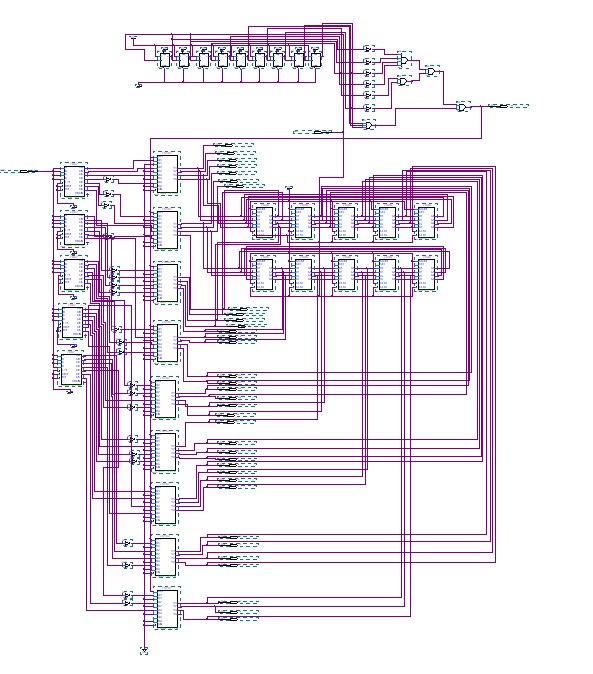


Figure 21 – Scrolling 7 Segment Display for the Word “Logic”

In this part, the objective is to design a dynamic scrolling display on a 7-segment LED module in order to display the word “LOGIC” or any other desired word.

To begin, we first analyzed the segment configuration (a to g) required to represent each character of the word “LOGIC”. For example, in order to display the letter “L”, we identified which segments (a to g) must be activated to correctly form the shape of the character on a 7-segment display.After determining the segment pattern for each letter, we used five BCD to 7-segment decoders. Each decoder receives a constant binary input of 0000 (BCD input 0), which by default outputs the digit '0' on the display. To display custom letters (e.g., L, O, G, I, C), we inverted the individual segment outputs as needed using NOT gates, thereby creating the required shapes for each character.The outputs from each decoder corresponding to a letter were then fed into multiple 2:1 multiplexers, with all multiplexers sharing the same select line, the 4-second counter signal used in our project. Each multiplexer receives:-Input A: the segment pattern for the intended letter (e.g., L, O, G, I, C)-Input B: a constant 0The logic behind this configuration is as follows:-When = 0 (i.e., within the initial 4-second interval), the multiplexers select Input A, allowing the segment patterns to pass through and form the word “LOGIC” across the 5 displays. During this phase, the word is displayed statically without scrolling.-To enable scrolling, we connected the outputs of each BCD to 7-segment decoder to a circular shift register located after each LED. We used two shift registers per character in order to store and transfer all 7 segment values, operating in Mode = 1 (parallel load using D0–D3), with CLKL set to high, and CLKR connected to the shift clock. This configuration ensures that the data output from each shift register is fed into the subsequent LED to generate a continuous scrolling effect.As long as =0 the display prioritizes the direct segment input (Input A), and the word “LOGIC” remains visible for a span of 4 seconds. During this time, the shift registers are loaded and synchronized in preparation for the scrolling cycle.Once the 4-second interval is exceeded and =1, all multiplexers switch to Input B, which is tied to logic 0. As a result, the direct letter outputs are disabled, and the display now solely receives input from the shift registers. At this stage, the CLKR clock signal becomes active, causing the shift registers to begin shifting the preloaded segment patterns. This initiates the scrolling effect across the display for the next 4 seconds, creating the illusion of the word “LOGIC” moving from right to left.After another 4-second interval, returns to 0, reactivating the original letter patterns and restarting the cycle. This concept mirrors the scrolling text effect commonly seen on pharmacy signage or storefront displays.

## GitHub Repository with Comprehensive Documentation

We have assembled the complete project repository, including all necessary schematics, Quartus source files, simulation waveforms, and comprehensive documentation, and organized them into a clear, logical directory structure for effortless navigation. In the root directory you’ll find a detailed README that explains the overall purpose of the project, describes each file and folder, provides step-by-step setup and usage instructions, and shares insights into key design decisions. Finally, we recorded a demonstration video, uploaded it to YouTube, and embedded its hyperlink directly in the README for your convenience.

<https://github.com/hadi-soubra/Logic-lab-project?tab=readme-ov-file#logic-lab-project>

## Recreating the 555 Timer Design and Documentation in LaTeX

We implemented the design and theoretical description of the 555 timer in LaTeX, using mathematical formulas. We also utilized the LaTeX circuit renderer to illustrate the 555 connections. The explanation of how the 555 timer works and was implemented was mentioned earlier in the 555 circuit implementation part. For this documentation, we provided a screenshot of the rendered circuit, and the code for it is available in the linked GitHub repository.

## A diagram of a circuit AI-generated content may be incorrect.

Figure 22 - 555 timer render circuit

# Conclusion

The designed system successfully fulfills the functional and technical requirements of the logic-controlled board using only discrete ICs. By combining FSM-driven state control with edge detection, D flip-flop memory, and structured mapping through multiplexers and decoders, the circuit achieves accurate and reliable lamp tracking per switch. The use of a boot-mode selector enhances the system's flexibility, allowing it to operate in two distinct modes. Despite facing challenges in signal synchronization, ROM alternatives, and pin mapping, careful planning and simulation ensured a robust and fully functioning design. This project highlights the power of foundational logic components in building complex, responsive systems without relying on programmable devices.