Lebanese American University

School of Engineering

Department of Electrical and Computer Engineering

COE 322 - Logic Design Lab

Project

Karim Koaik 202302628

Mohammad Ismail Hashem 202303895

Yehya Mazloum 202306541

May 14th, 2025

Table of Contents

Table of Figures

Table of Tables

Abstract

Our project, which is Logic-Controlled Board is an implementation of a logic circuit which smartly manages the lamp activation according to the user's interaction. This controlled board goal is to map the switches to the LEDs according to a certain sequence that is determined by the user by the last switch being turned off. Initially, all the switches are mapped to the LEDs that are facing them, however, when turning all the switches then turning them off, a new sequence of LEDs turning on will be implemented. To achieve this working procedure, the design must be based on combinational and sequential circuits, memory elements, and a 555 timer which is a clock. To visualize the active sequence, we used a 7-segment display to display the sequence number. To design the circuit and simulate the output we used Quartus II as our software technology. The implementation strategy includes dividing the circuit into multiple blocks each one with a certain functionality to simplify our work. The first part of the implementation is related to mapping the switches to its corresponding LED and the second part was for detecting the sequence and choosing the activation sequence of the lamp according to the sequence detected. This implementation is done using multiples of D-flipflops, multiplexers, decoders and logical gates. After building the circuit on Quartus II and running its simulation, it is found that our output is the same as the desired one. For example, when all the switches are turned on and then turned off, the last switch that is turned off is switch 2 which means that we are now in sequence 2. This is well seen in the simulation of the output, as when we turned on any switch the second LED turned on then the third and fourth and finally the first LED. This shows that our circuit is well implemented and that the objective of having a correct LED mapping and sequential activation is well achieved. The significance of this project is that it allows us to implement any digital system we observe in our surroundings, and their broad applicability in the real world. The understanding of the relationship between the logic circuits and the real-world applications makes a bridge between theoretical learning and practical applications.

Introduction

Using fundamental ideas from digital logic design, we investigate the design, implementation, and optimization of a logic-controlled board in this lab project. Built around switches and lamps that react dynamically to user inputs, the system's behavior is dictated by the last switch that was turned off and is improved by features like locking modes. Quartus software and 555 timer-based clock circuits are used to implement and simulate the design. Our comprehension of clock generation, memory components, and modular circuit design is strengthened by this project. Lastly, we look at the development of interactive digital systems to learn how similar reasoning controls practical applications.

Components and Equipment Used

- 74LS154
- 74LS74
- 74LS08
- 74LS153
- 74LS04
- 74LS32
- 74LS48
- 74LS266
- 74LS86
- 74LS138
- 74LS161
- 555 Timer
- LED Switches
- Seven-Segment display
- Wires
- Quartus II software
- Simulador Digital de protoboard

Analysis

In this part we will analyze the circuit to see how it is implemented, the approach used and how logical ICs were used in the implementation.

To begin with, let us have an overview of how the project works and then proceed with the analysis. The project is about a magical trick where the audience thinks that every colored switch turns on the corresponding-colored LED i.e., LEDs turn on when their switch of same color turns on. This happens not because of magic or some sorcerer but because of sequences and visual tricks. Frist, we start with initial sequence where the output LEDS will turn on from LED 1 to LED 4 respectively no matter of which switch is turned on first i.e. any switch turned on will lit the first LED, then any second switch will turn the second LED and so on. After the four LEDS are turned on, the LEDs now will remember the switch that turned it on. If we start turning LEDs on and off, the mapped switch will only affect its correspondingly remembered LED. Now, if we turn off all four switches, we will advance to another sequence depending on the last switch turned off. To explain, if switch one turns off last, we will stay in sequence, however if the last switch is switch two, we will advance to sequence two and same for switches 3 and 4 leading to sequences 3 and 4, respectively.

We will follow divide and conquer strategy to make this project. So, we are going to divide the tasks where each one is responsible for a part of the project. We first must start with four input switches and S1-S4 and four output LEDs red, blue, yellow, and green. Since we are dealing with multiples output combinations for only four switches, we will decode the inputs using a decoder. We must map each switch to the corresponding LED and that can only happen using memory registers which are the famous D-Flip-Flops. Then, we must-after finishing default sequence-proceed to the next state if all switches are turned off for a specific time denoted by 4sec from the 555 timer. This timer is implemented using capacitors and resisters to adjust the time to 4sec. The timer will be used to mark that we will enter the second sequence after portion of time when all switches are turned off. We will also use D Flip-Flops to save the last turned off switch to know which sequence we will be in next. If for example we turn SW3 off last, we will advance to a sequence where switches in any order will light LEDs 3, 4, 1 and 2 respectively. If we turn switch 4 lastly, the sequence will become 4-3-2-1. The sequence for switch 2 is 2-3-4-1 and if switch 1 is turned off last, we will stay in the same sequence as the default one.

Moreover, we have a locked mode that when we enter it, switch one will light LED one, and switches two, three and four will light LEDs two, three and four, respectively. Locked mode is entered if we turn on switch two at first in the default state.

Paper Design

In this section we will follow up with how we can implement a 555 timer to use it in the progressing between sections.

First let's explain what a 555 timer is and what it does. The 555 timer IC is an incredibly versatile and widely used chip, serving roles from clock generation and signal delays to pulse creation and oscillation. In its monostable configuration, the 555 works as an RC (resistor-capacitor) network with an external resistor and capacitor. When a trigger pulse arrives at the trigger pin, the IC begins charging the capacitor internally. Once the capacitor's voltage reaches two-thirds of the supply voltage, charging stops and the output returns to its low state, ready for the next trigger event. By selecting different resistor and capacitor values, you can adjust how quickly the capacitor charges—and thus lengthen or shorten the duration of the rectangular output pulse.

We must first establish the equation of this timer to know which values to be used for capacitors and resistors.

Charging of capacitor follows the following equation:

$$V_C(t) = V_{CC}\left(1 - e^{\frac{-t}{RC}}\right)$$

The 555's threshold comparator switches the output low when:

$$V_C(t) = \frac{2}{3}V_{CC}$$

We take this value and replace V_C above by 2 over 3 V_{CC}:

$$\frac{2}{3}V_{CC} = V_{CC}\left(1 - e^{\frac{-t}{RC}}\right)$$

$$\frac{2}{3} = \left(1 - e^{\frac{-t}{RC}}\right)$$

$$\frac{-t}{RC} = ln\left(\frac{1}{3}\right)$$

$$t = RCln(3)$$

So, $t \approx 1.1RC$

We need a four second time, so the values for R and C to satisfy the equation are:

R=360 kΩ

C= 10 μF

So, $t \approx 1.1(360 \times 10^3)(10 \times 10^{-6}) \approx 4\,sec$

Now, we will implement this timer using these values of R and C in addition to the 555 timer IC:

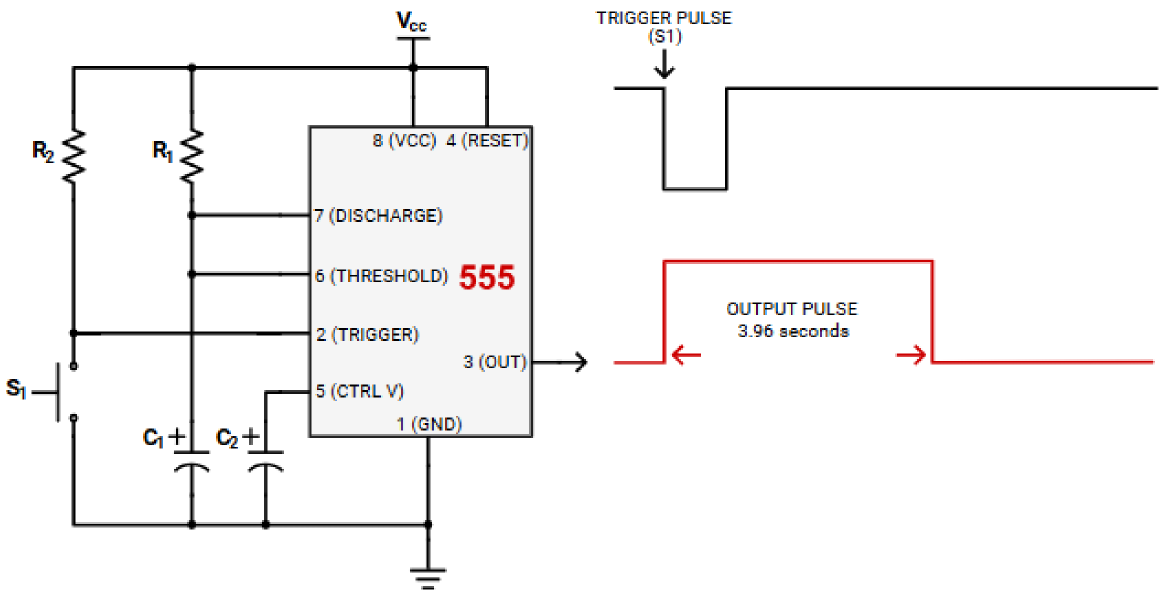


Figure 1 - 555 timer circuit

Note that R_1=360 kΩ

C_1= 10 μF

Quartus design and analysis

We will start Quartus design by entering the switches into a 4 to 16 decoder as follows:

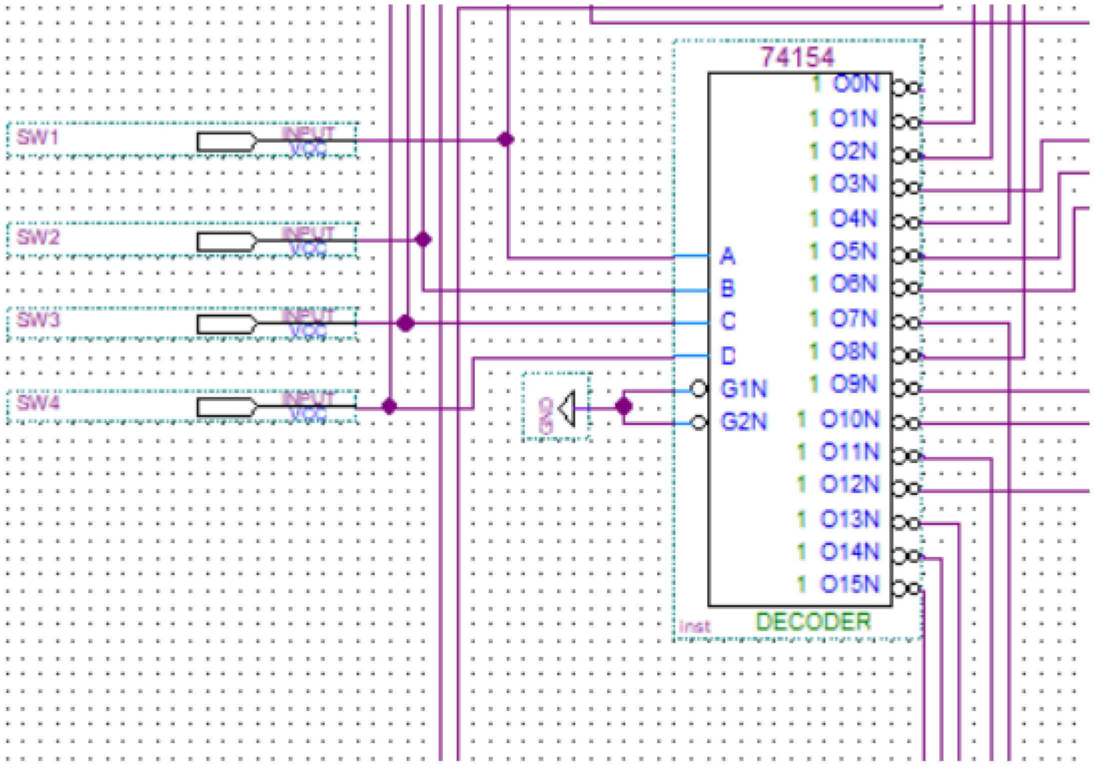


Figure 2 - Quartus design showing how inputs are entered into a 4 to 16 decoder.

The output of the decoder is taken based on the number of 1's in the input which means the number of ones in the input corresponds to the number of switches turned on. So, if only one 1 is entered i.e. 0001or 0010 or 0100 or 1000 the outputs corresponding to 1,2,4 and 8 are taken to block 1 and we mark only one switch is on and its either SW1, SW2, SW3 or SW4. Then, when we have two ones, i.e. 0011, 0101, 0110 ,1001, 1010, 1100 the outputs corresponding to 3, 5, 6, 9, 10 and 12 are taken to block 2. Same for three ones input and four ones. It is as follows:

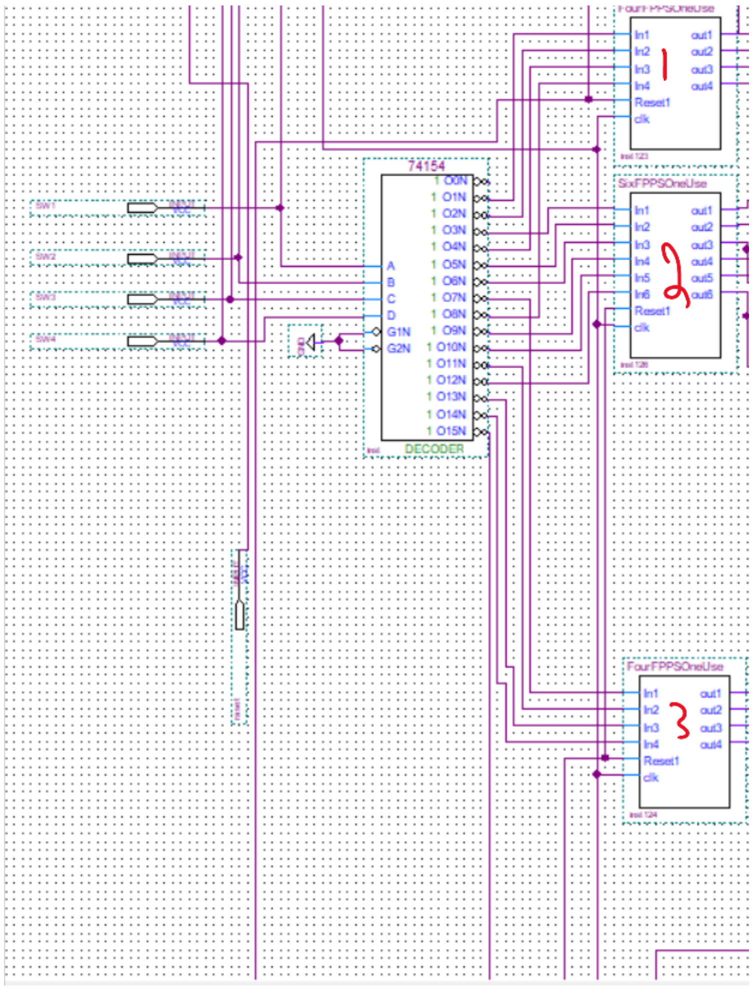


Figure 3 - Quartus design for the outputs of decoder

As seen above, we will be now referring to these blocks to know the switches behind lighting LEDs. However, we only have 3 blocks since the input of four ones 1111 has only one output which is 15, so there is no need for any blocks. Let’s take a deeper look into what is these blocks made up of:

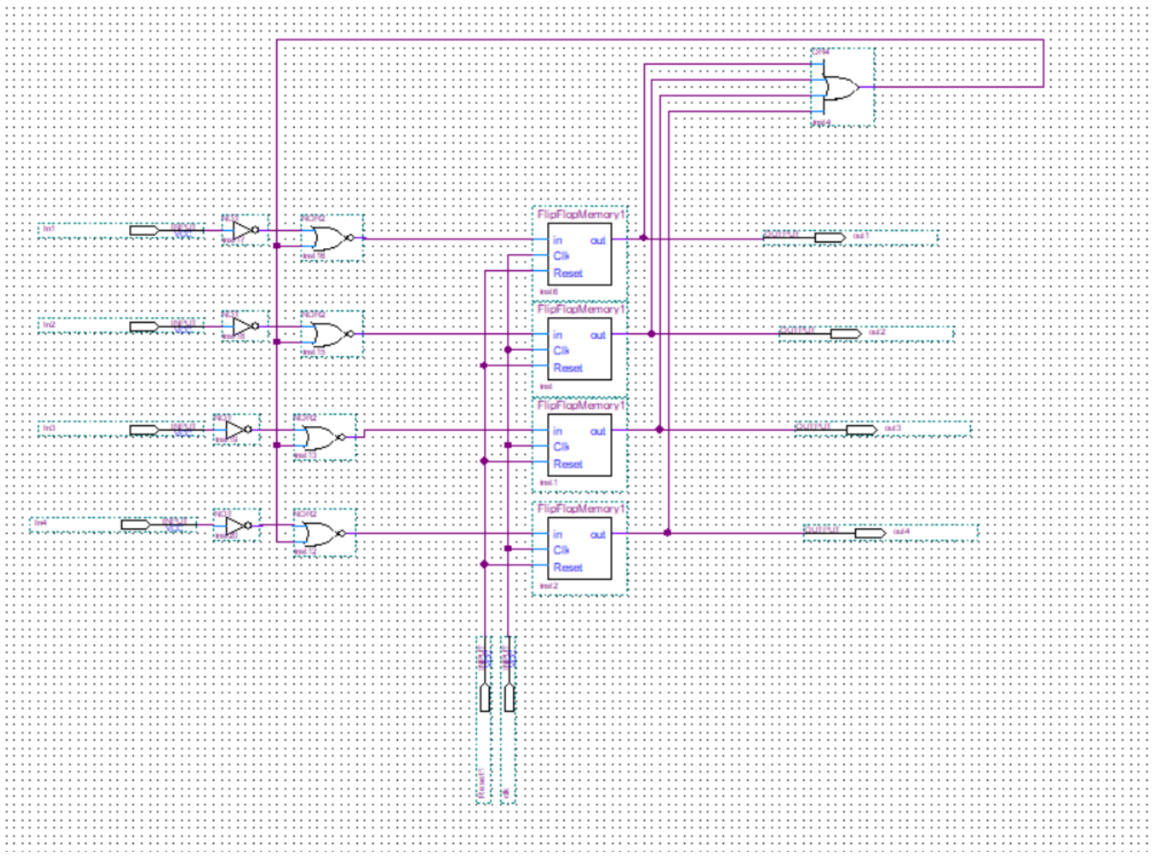


Figure 4 - Blocks receiving decoders output form inside.

These blocks contain specific number of flip flops each saves one and only one switch and holds it forever or at least for when we reset them. For example, if switch one is turned on, initially all inputs are zeroes since one nor zero is zero. However, when SW1 enters as 1, it is inverted to 0 so that 0 nor 0 will output one and flip flop one will save SW1 is the first switch turned on i.e. SW1 will be mapped to LED1.

Now, we will proceed to the second switch, so will have -for example- input 0101 to the decoder. So, switches SW1 and SW3 are the first two switches turned on, but without clue on whether the second is SW1 or SW3. To solve this, we implement the following:

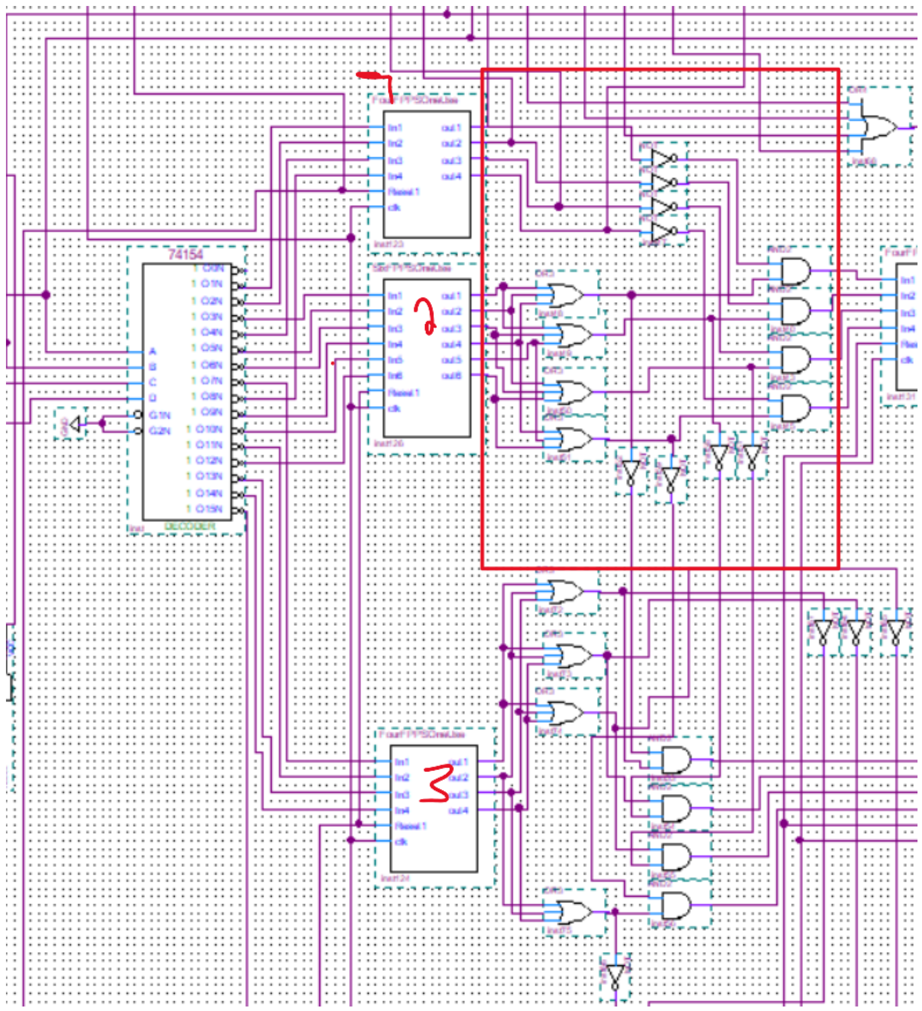


Figure 5 - Detecting the order of switches.

As noticed in the red box, we are taking all the possibilities resulting in switch one 0001 being secondly turned on and not first. The OR will account for 1001 0101 and 0011 meaning that we have LSB a one. Also, we are negating that SW1 is what lit LED1 by the not coming from the first Block. Now, we are sure which switch was turned on in order. The same is done for Block three and output fifteen where we detect the switches turned on iteratively.

Now, we know which switch turned on which LED and saving them in flip flops until reset. To light the first LED, at least one output of each block must be one (one input at a time but we donot which one so we OR all the outputs of the blocks).

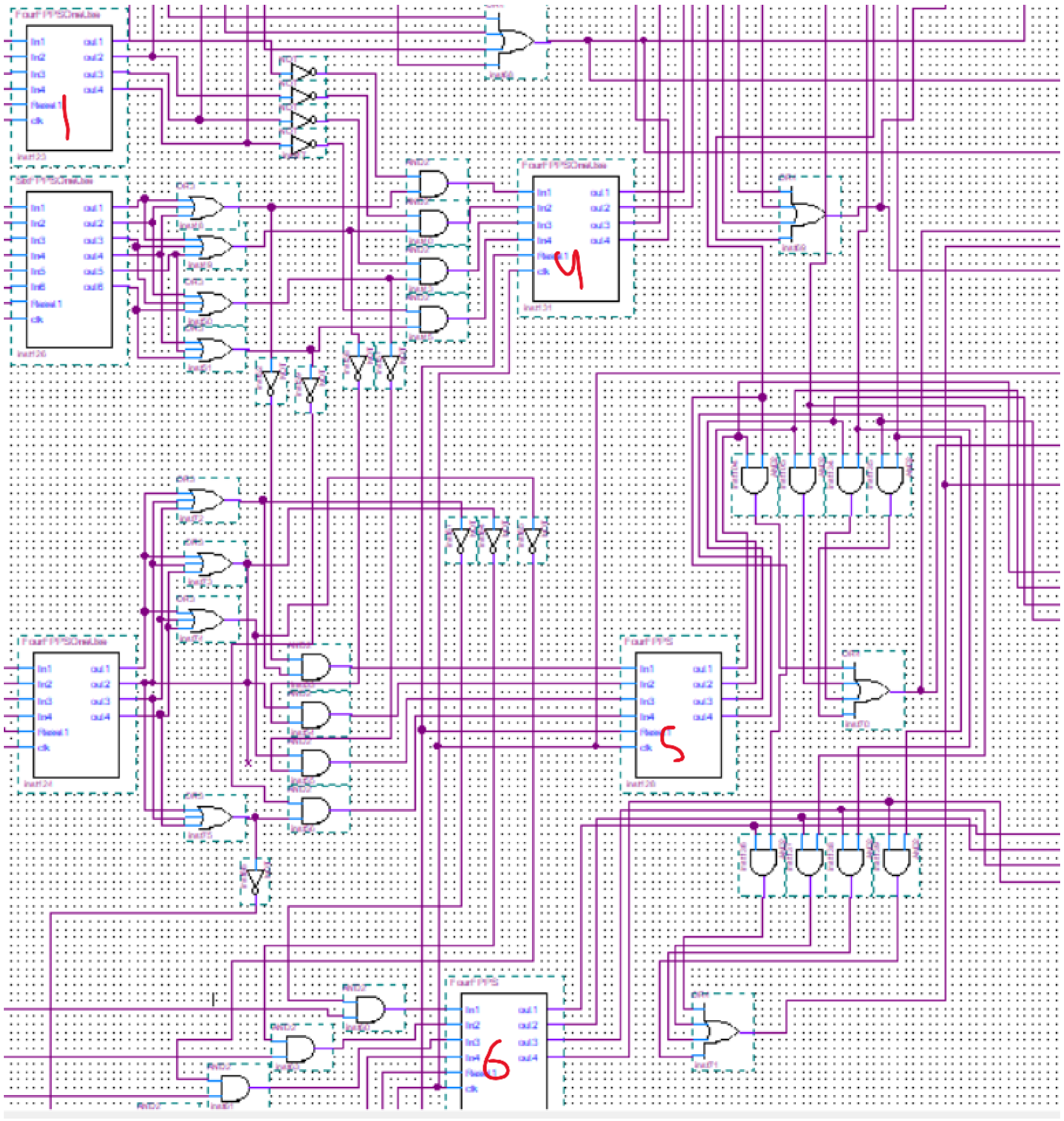


Figure 6 - Blocks outputs illustrated

For example, the output of block five which corresponds to LED 3 is being entered to a four input OR to light LED 3. But, before that, we must account for when turning of the switch to turn off the LED, so the output is first entered to an AND gate with the switch itself so that when the switch is turned off, the LED will as well. Out 1 corresponds to SW1 and out 2 for SW2 and so on, hence we AND them accordingly. The same is done with each of the four blocks.

Now, we have completed the default sequence.

Before advancing further, lets just explain the reset where it happens after four seconds of turning off all switches. In Quartus we replaced the four seconds with a less time in microseconds, however, in reality the reset works exactly as we implemented but with a larger scale of time. When this duration passes, we clear all the mapped sequence to save new one.

What we will do now is that we will take a four input OR of the four switches and invert the output so we can know when all switches are turned off, we end up with a 1. Also, we will take a four input AND of the output of the previous four input OR coming from the blocks illustrated above. The purpose of this AND is to give one when all LEDS are turned on. This value when all switches are turned on is saved inside a Flip Flop to mark that we have -at some point- lighted all the LEDs. Now, we must take the AND of these two outputs i.e. when all LEDs are turned on then all off to an AND gate and let's call it NEXT.

Now, we must count four seconds when NEXT is one to proceed, else we stay in the same sequence. To do that, we will be using a 555 timer. Note that in Quartus we used regular clock for that, however for breadboard, it is crucial to have a 555 timer. The timer is implemented in the paper design section. Now, after the four complete seconds while NEXT is true, we proceed to the next sequence.

Note the following to illustrate how sequence detecting works:

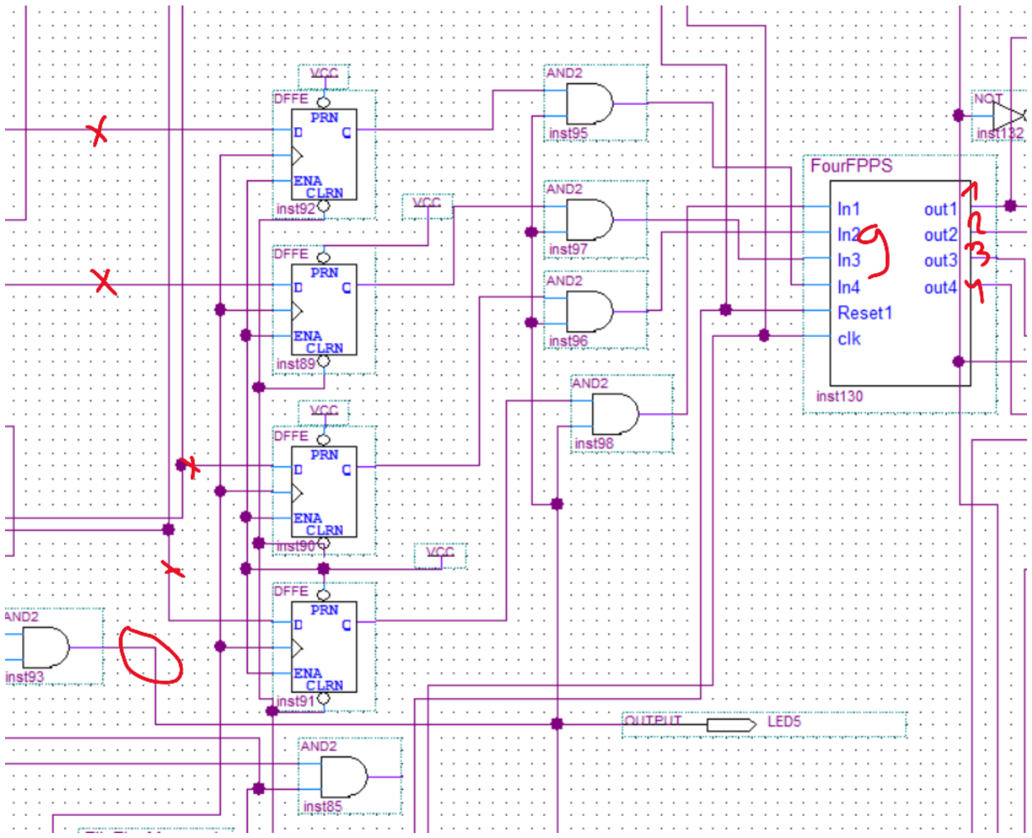


Figure 7 - Sequence Detection

The circled AND is NEXT showing the moment when all LEDs are turned off after they all have already turned on. The four flip flops are each taking input from the switches themselves. So, initially when LEDs are on, Q output is one, however, block nine is not working since gate NEXT did not yet output a one. When switches start turning off, the Q output is becoming 0. When the last switch is turned off, and at that moment, NEXT is one and the flip flop corresponding that switch still has output Q of 1, so the block nine will save the last switch turned off. Please note that block nine is four flip flops that saves the output.

Now, we have the last switch turned off saved and can implement the sequence we want using logic gates.

Let us proceed and implement the seven-segment display that shows the sequence we are in.

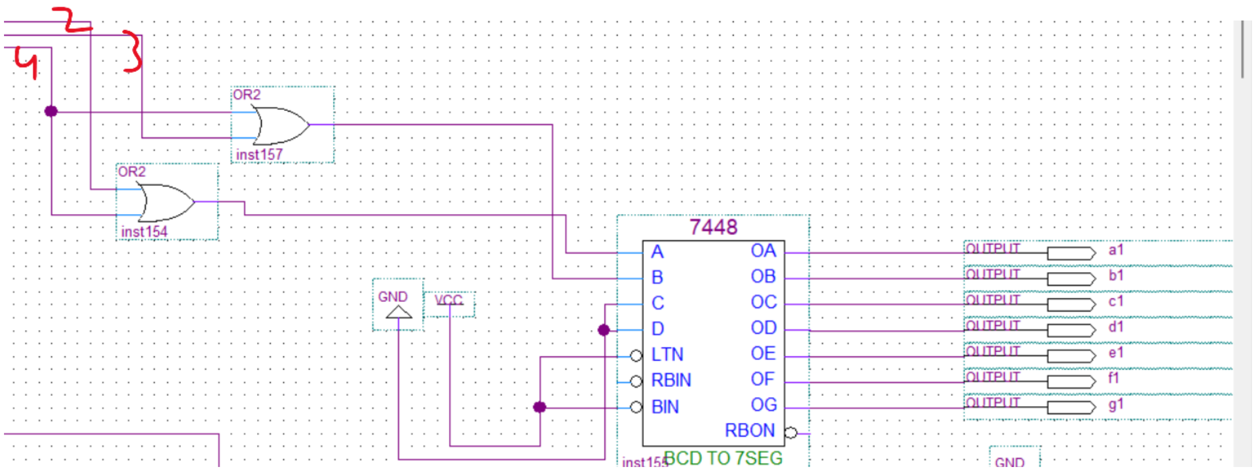


Figure 8 - Seven-Segment display showing the sequence we are in

The inputs are taken from block nine that holds the last switch tuned of i.e., the next sequence. The inputs are B and A as MSB and LSB, respectively. MSB is one when we are in sequence two or four. LSB is one when we are in sequence four or three. Initially in the default state, the inputs are 0 0 meaning sequence zero. The outputs are taken to seven-segment display of parts a, b, c, d, e, f, and g.

Finally, for the locked state, we are using a flip flop to save the second output from block 1 to mark that the first switch turned on is SW2. Then we implemented a 4 to 1 MUX that has switch lanes A, the output of the flip flop and Lane B connected to the NEXT saved in flip flop. Input 1 is taken showing locked mode state where A=1 and B=0 i.e., we are in the default sequence, and we have lit SW2 first. Locked mode corresponds to each switch turning on the corresponding LED as follows SW1-LED1, SW2-LED2, SW3-LED3 and SW4-LED4.

Here is the finalized block of the design:

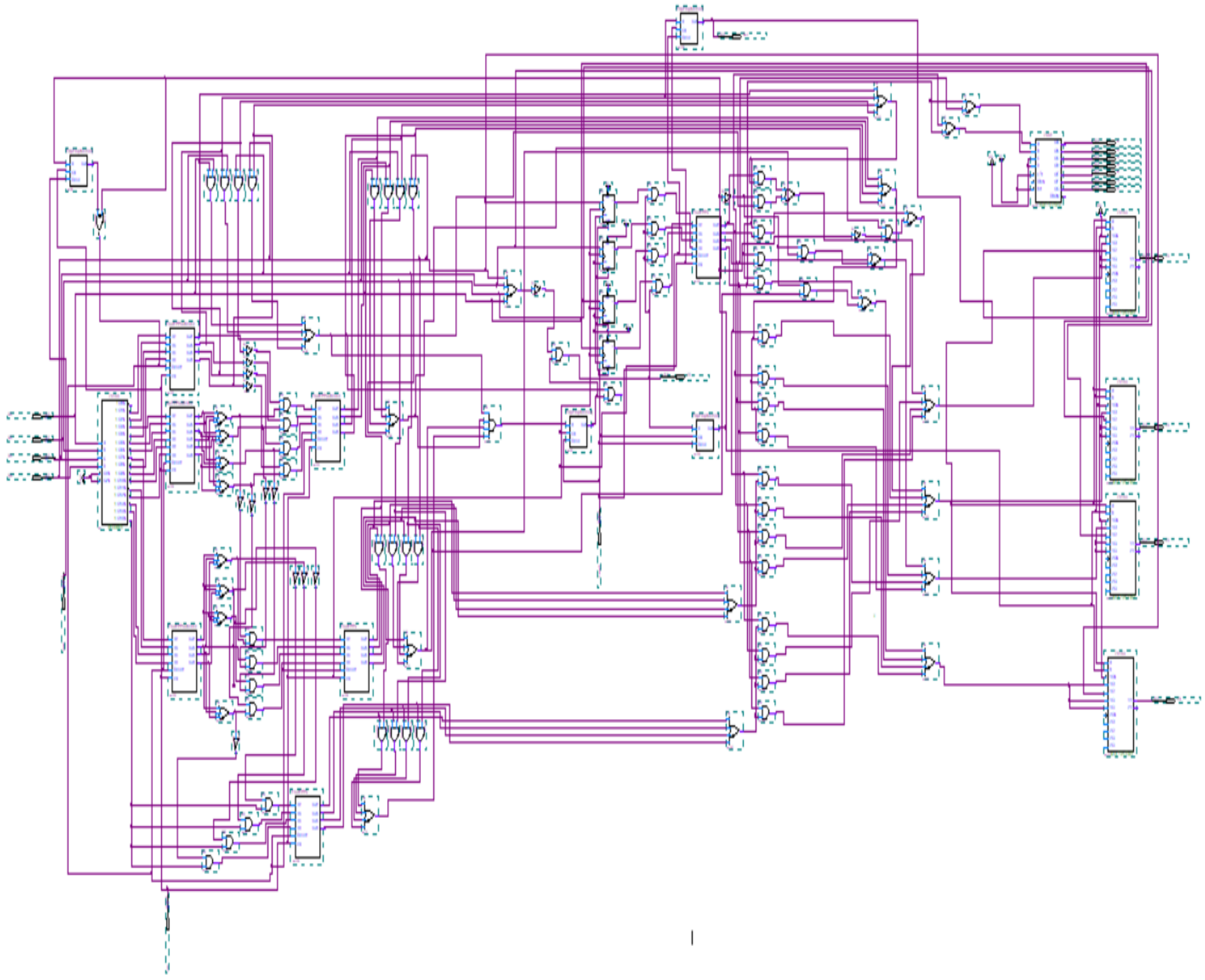
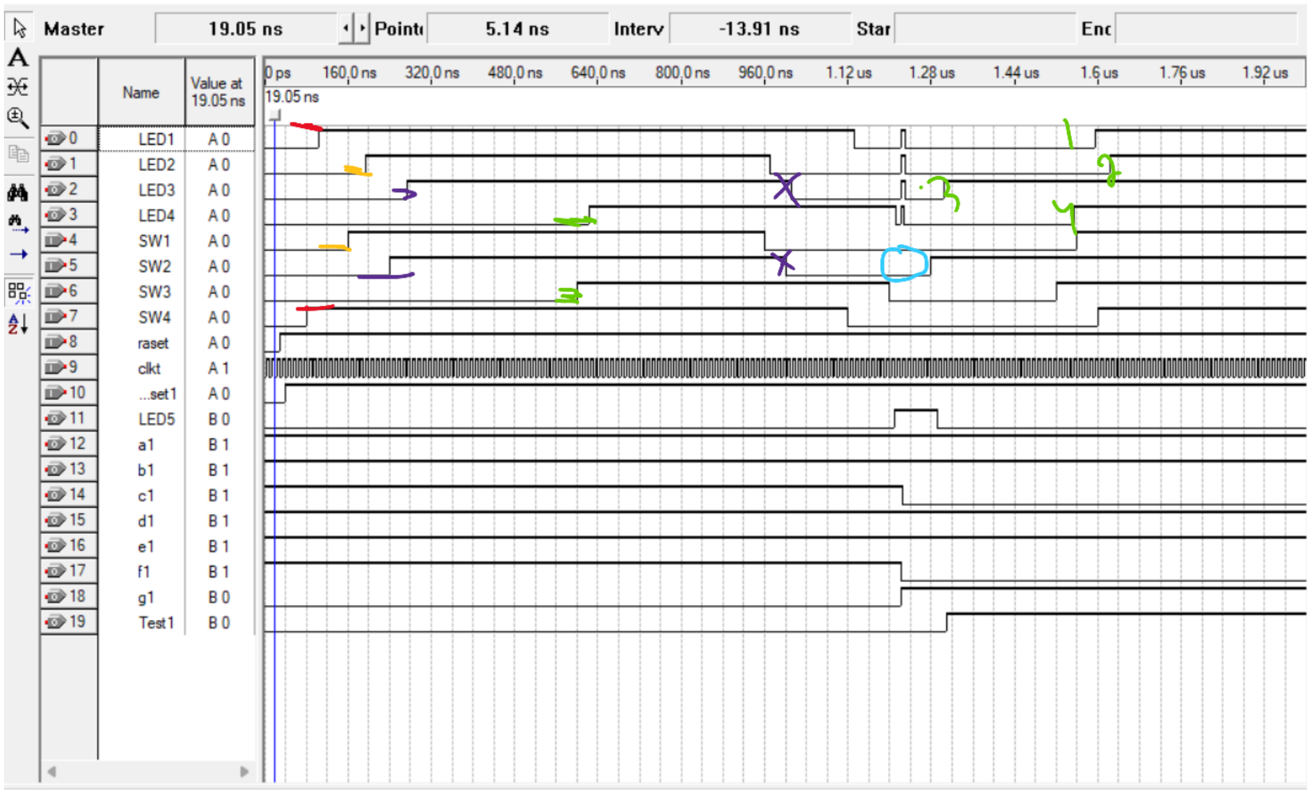


Figure 9 - Final Quartus Design Block

Now, let us discuss the different simulations:

Simulation one:



```
Master   19.05 ns   Pointe  5.14 ns   Interv  -13.91 ns   Star   Enc
```

	Name	Value at 19.05 ns
0	LED1	A 0
1	LED2	A 0
2	LED3	A 0
3	LED4	A 0
4	SW1	A 0
5	SW2	A 0
6	SW3	A 0
7	SW4	A 0
8	raset	A 0
9	clkt	A 1
10	...set1	A 0
11	LED5	B 0
12	a1	B 1
13	b1	B 1
14	c1	B 1
15	d1	B 1
16	e1	B 1
17	f1	B 1
18	g1	B 0
19	Test1	B 0

Figure 10 - Quartus Simulation One

It is clear from the simulation that regardless of which switch is turned on first, LEDs are turned on in sequence 1-2-3-4 showing the default sequence. Switches turned on in order of SW4-SW1-SW2-SW3 and output LEDs still turned on from 1 to 4. Now, if one can notice the blue circle indicating that all switches are turned off and the last one to be turned off is SW3. This means we need to enter sequence 3 and that is exactly what has happened. Regardless of the switch order, the switches turned on in sequence of 3-4-1-2. Let just infer one thing about the duration of the circle, it must be around 4 sec, but in Quartus we implemented the same logic with lower reset in micro and nano seconds. Also, note that after the mapping that happened when SW2 turned off, LED three was turned off showing a perfect mapping. Regarding the delays, we will discuss it is the delay section. But overall, it is due to the propagation delay of the logical ICs that causes such glitches and delays.

Simulation 2:

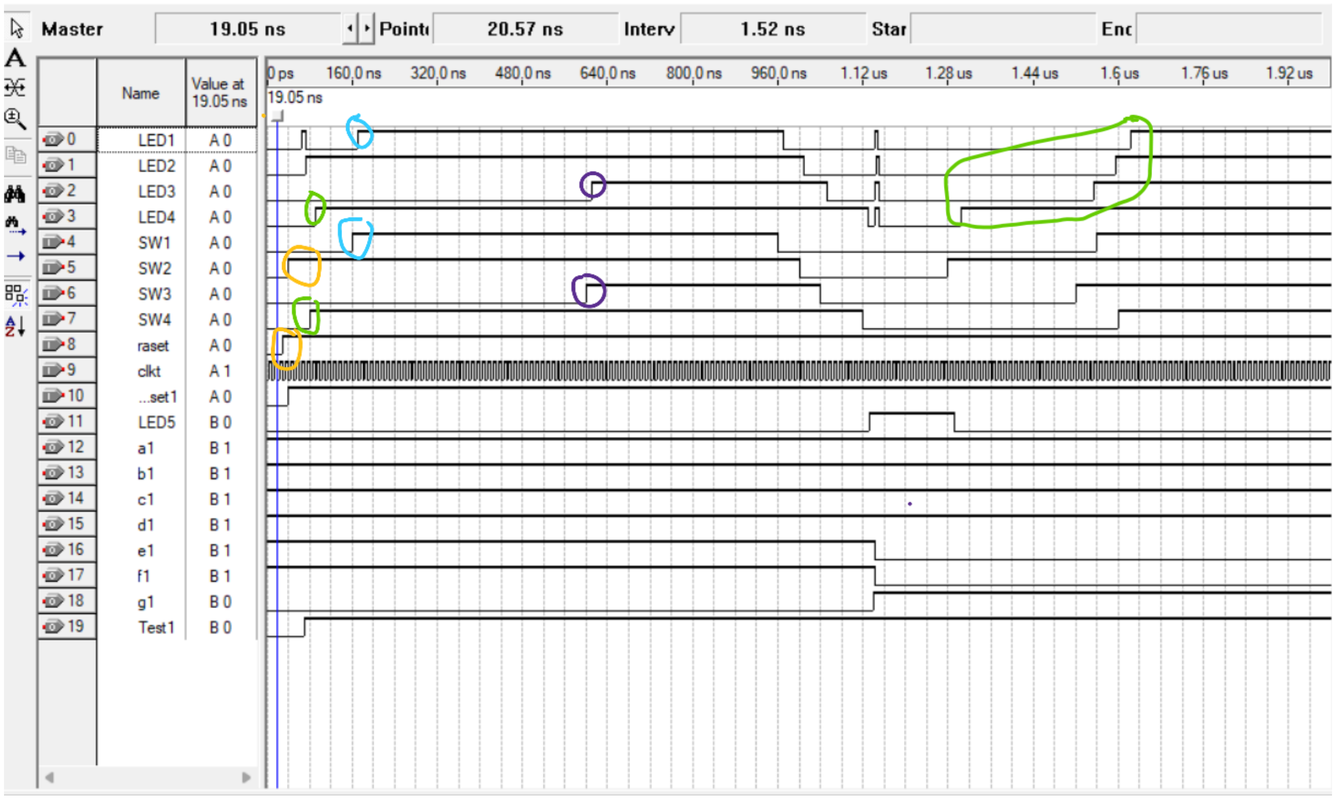


Figure 11 - Locked Mode Simulation

We can clearly notice that when switch two is the first lit, we have entered locked mode and as explained in the analysis, each switch lights the corresponding LED. As noticed in the simulation, SW2 turns LED2, SW1 turns LED1, SW4 turns LED4 and SW3 turn LED3.

Also, since we turned off SW4 lastly notice how the sequence becomes 4-3-2-1 regardless of switch turning on.

Financial study, focusing on minimizing the cost of the design

An important way of reducing both the bill of materials and assembly complexity is to take advantage of universal gates which are NAND and NOR. Because either of these gate types alone can be wired to implement any Boolean function, you can often replace a series of different gate-type ICs (AND, OR, NOT, XOR, etc.) with one or two part numbers.

Let us give an example of how costs differ:

Table 1 - Cost table of some ICs

Part Number	Function	Typical Unit Cost $	Quantity Used	Subtotal ($)
74LS00	Quad NAND	0.6	4	2.4
74LS08	Quad AND	0.45	2	0.9
74LS32	Quad OR	0.45	2	0.9
74LS04	Hex Inverter	0.50	1	0.5
Total				4.7

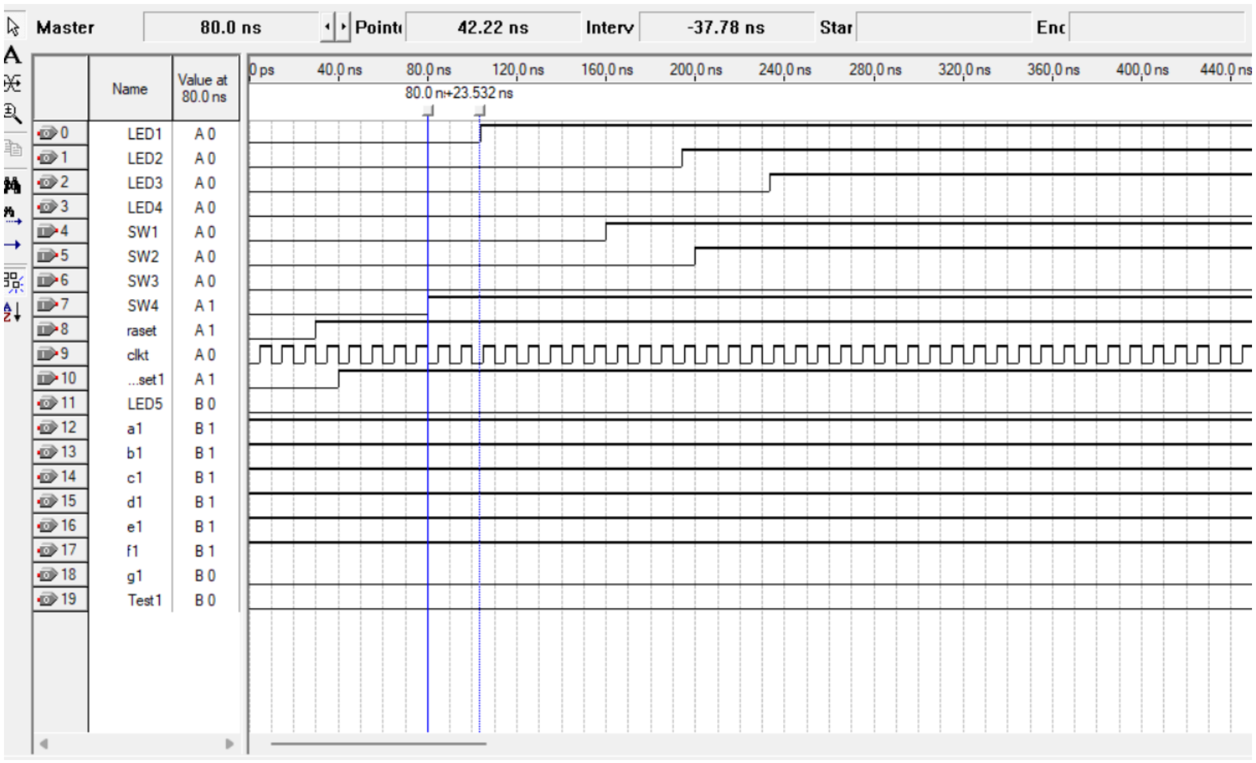
You save $1.70 (a 36 % reduction) in gate IC costs alone, and you only need to carry one part number instead of four.

It is also simpler to iterate on the design with universal gates. When you need to change the logic (add a new feature or fix a bug), you can redeploy an unused NAND gate rather than having to source and stock a different type of gate. All this flexibility translates into reduced "design-change" costs over the product's life.

Briefly, using extensively NAND or NOR chips not only reduces the number of ICs you buy—it also reduces the number of SKUs, inventory overhead, and simplifies both power and layout problems. Each of these factors together can place total component cost below the $20 target even more easily than by pure Boolean minimization alone.

Delay Calculation

Delay is a normal thing when it comes to large circuit where the propagation delay over and over results in some delays in the outputs. Let us see the delays and calculate them for our simulation.



Master 80.0 ns | Point 42.22 ns | Interv -37.78 ns | Star | Enc

	Name	Value at 80.0 ns
0	LED1	A 0
1	LED2	A 0
2	LED3	A 0
3	LED4	A 0
4	SW1	A 0
5	SW2	A 0
6	SW3	A 0
7	SW4	A 1
8	raset	A 1
9	clkt	A 0
10	...set1	A 1
11	LED5	B 0
12	a1	B 1
13	b1	B 1
14	c1	B 1
15	d1	B 1
16	e1	B 1
17	f1	B 1
18	g1	B 0
19	Test1	B 0

Figure 12 - Delay of simulation

As we can notice from the simulation, we have a delay of 23 ns between pressing switch 4 and turning on of LED 1. This is due to the propagation delay of the logic gates and ICs where the time needed to pass from one level to another is not neglected but clearly seen.

NE555-emulated clock also has rise/fall times where no square wave is ever truly vertical. When we do that in HDL (as a pulse generator with rise/fall delays), the D-flip-flops see slightly different arrival times on their clock pins than their data pins. That skew can gnaw at your setup/hold margins, resulting in simulation-observed violations if the delays add up.

We have a 4 s reset from the NE555 cap timer. In simulation, representing that as a clean level signal doesn't simulate the infinitesimally small RC-charging curve, but if we insert a real model, we will see the reset pulse rise slowly over the TTL threshold (~1.6 V) on the order of microseconds. That window of time, your 74LS components may have undefined behavior. In a time-annotated sim, we can verify that both reset desertion and assertion occur off critical clock edges.

Power consumption analysis

The below table shows a summary about the approximate power dissipation for each IC type at $V_1 = 5$ V, using typical datasheet values:

Table 2 - Power consumption of some ICs

IC Type	Qty	ICC	P per IC	Total P
74LS08	11	ICC = 4 mA	5 V × 4 mA = 20 mW	11 × 20 mW = 220 mW
74LS74	22	ICC = 4 mA	5 V × 4 mA = 20 mW	22 × 20 mW = 440 mW
74LS153	2	ICC = 10 mA	5 V × 10 mA = 50 mW	2 × 50 mW = 100 mW
74LS154	1		45 mW	45 mW
74LS48	1		35 mW	35 mW
74LS04	5	ICC = 1.6 mA	5 V × 1.6 mA = 8 mW	5 × 8 mW = 40 mW
Total Power				880mW

0.88 W is a substantial but not negligible draw: it determines your power-supply design, thermal requirements, and, if you do end up battery-powered or energy-constrained, pretty much necessitates low-power CMOS solutions.

TTL devices are easy to connect but rather "power thirsty."

Employing CMOS logic (74HC/HCT) would reduce static drain to a matter of tens of milliwatts total—reducing heat by leaps and bounds and increasing battery life at a negligible parts cost penalty.

Problems Faced During the Project

The problems faced were many, so for clarity and simplicity, they will be divided into categories as follows:

Syntax Problems:

-These problems happened in running Quartus and in making if the blocks where not saving some times or overlapping of two wire without noticing will take really a lot of time to notice such small mistakes.

-The naming of components inside Quartus has resulted in some problems and took quiet a time to figure the best way of naming possible

Logical Problems:

- These problems happened when the file compiled but we get a different result than expected.
- At first, when turning on one switch, all LEDs were turning on as well, so we solved this problem by a very good mapping system.
- Another problem was turning off of a switch mot resulting in turning off the LED. Since we are dealing with flip flops that saves the input and keeps it one as long as it isn't reset, the LED would never go off. Solving this problem was not that hard, where we implemented a set of AND gates that take the output itself with the switch so the AND results in zero when switch is off.
- In sequence detector, the problem was in the output sequences where I pushed the sequences for the four outputs and that resulted in a sequence of 4-1-2-3 in sequence four. After some debugging and noticing how the circuit works, I was able to notice the pattern and reconnect few things and the problem was finally sorted.

Approach Problems:

At first, we thought of the project as one block and started working on it to find an output after doing the whole project at one. Obviously this approach was 100% wrong and very difficult. Then, we approached the project from another point of view emphasizing of divide and conquer algorithm where we solved the project tasks one by one. This method was clearly better and more logical while dealing with different tasks.

Key design points that present advantages over alternative designs

This design at hand utilizes the use of universal gates which will for sure minimize the cost and lower the power consumption and hence boosting advantages.

The design is divided into clearly defined functional blocks (7-segment driver, reset control, sequence detection, switch-to-LED mapping), each isolated by registers or combinational interfaces.

Benefit: Allows for development and debugging—each block can be separately designed, simulated, and tested—and promotes reuse in future designs.

D-flip-flops encapsulate the order in which switches are enabled initially (and eventually disabled), maintaining state until a deliberate reset.

Benefit: Provides uniform, glitch-free mapping without continuous combinational analysis; eliminates race conditions characteristic of pure combinational design.

One 7-segment driver block decodes the two-bit "current sequence" code into segments a–g. **Advantage:** Provides immediate visual display of the sequence currently active, drastically minimizing both user operation and field troubleshooting.

A small additional flip-flop and 4:1 multiplexer implements a "locked" mapping mode (each switch directly controls its own LED) when switch 2 is initially turned on.

Advantage: Implements a useful feature with minimal incremental hardware and without altering the basic FSM logic

Bonus part

Our aim in this part is to display the word logic in a way that it is shifting to the right and then entering again from the beginning.

To design its circuit, we will need a 4-Bit counter (74LS161) to count from 000 till 100, where each 3-bit binary number represents a letter. We chose 000 for letter C,001 for letter I, 010 for letter G, 011 for letter O, and 100 for letter L. After the counter we used a 3 to 8 decoder (74LS138) which has inputs as the 3-bits output of the counter. For each letter we need to find the segments that must be on to display them on the seven segments.

The segments that must be on for each letter are as follows:

(000) C: a,f,e,d

(001) I: b,c

(010) G: a,b,c,d,f,g

(011) O: a,b,c,d,e,f

(100) L: f,e,d

To determine the function of each of the segments we must see when each segment is on:

I0,I1,I2,I3,I4 are the outputs of the 3 to 8 decoder.

For a it is on when 000 or 010 or 011

Which means that a: I0+I2+I3

For b it is on when 001 or 010 or 011

Which means that a: I1+I2+I3

For c it is on when 001 or 010 or 011

Which means that a: I1+I2+I3

For d it is on when 000 or 010 or 011 or 100

Which means that a: I0+I2+I3+I4

For e it is on when 000 or 011 or 100

Which means that a: I0+I3+I4

For f it is on when 000 or 010 or 011 or 100

Which means that a: I0+I2+I3+I4

For g it is on when 010

Which means that g: I2

After we get the output of each segment, we connect them to 7 d-flipflops. Then we connect these 7 d-flipflops and we connect their output to another 7 d-flipflops. We do the same process till we have 4 levels each containing 7 d-flipflops. The output of each level is connected to one of the 7 segment display. The first level is connected to the first 7-segment display, the second level is connected to the second one, the third level is connected to the third one and the fourth level is connected to the fourth one.

We did this cascading since we needed each letter to be shifted to the right segment after each clock.

Now, for the counter to reset back to 000 and count from it again, we must make the load active when the count becomes 101 (since we need 5 letters and on the 6th iteration we need to reset). To make the load active when the count becomes 101, we need to connect QA and QC of the counter to an 'AND' gate and invert the output since the load is active low(QA is the LSB and QC is the MSB).

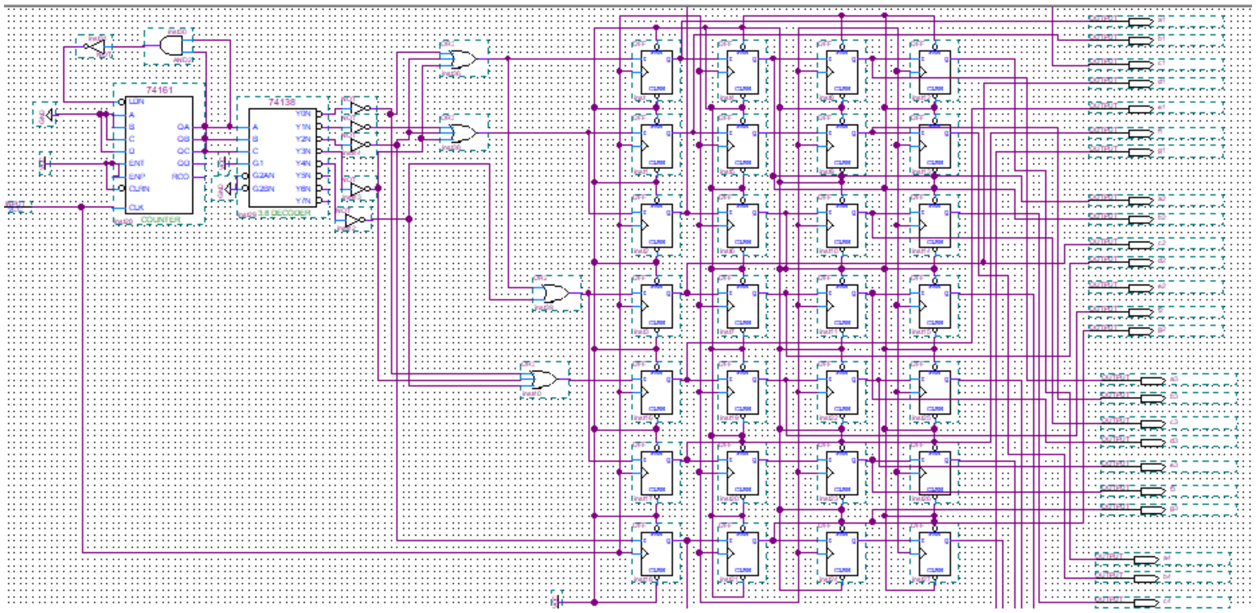


Figure 13 - This is the implementation of the circuit on Quartus II

WhatsApp Video
2025-05-14 at 2.42.40

Git Link

https://github.com/Yehya005/LogicDesign.git

Conclusion

From this project, we can better understand how digital logic can be employed to create dynamic and interactive systems. By combining combinational and sequential logic, and memory elements, we can create a system that changes its action based on user interaction. Designing the implementation using Quartus enhances our design thinking and hands-on skills. Furthermore, the project educates us on clock generation using 555 timers and introduces us to advanced features like locking mechanisms. Overall, the hands-on learning bridges the gap between theoretical concepts and real-world digital system design.