Polytechnic University of Puerto Rico

Department Electric, Computer Engineering and Computer Science

FALL 2021

Design Project Ver 1.0

Jessica N. Quintana Rivera #ID: 105416

Sebastian Castro #ID: 107094

Savier Carranza Vélez #ID: 114119

Gabriel García Torres #ID: 107298

Michael E. Negrón Labrador #ID 116073

Juan G. Torres Feliciano #ID 113238

COE 2300 Sec. T1

Tuesdays

4:30 a 6:30 p.m.

Profe. Marvi Texeira

Polytechnic University of Puerto Rico
ECECS Department
COE 2300 Logic Circuits / CS 2302 Digital Logic / Secc: M2, T1, 80, 87

Design Project FA21 Ver 1.0

Issued: September 7- 2021

Due: By the date established in your Blackboard schedule.

COE2300	secc M2	Class Hours	See Schedule
COE2300	secc T1	Class Hours	See Schedule
CS2302	secc 80	TBA	
COE2300	secc 80	TBA	

Submission of the Project Report and submission of the Logisim or Digital file will be by email prior to the presentation.

Presentation of your work (using your PDF technical report), followed by a simulation/demonstration using Logisim or Digital and VHDL (bonus work). Questions by the instructor at the Project Presentation will be through a Collaborate Session at the above designated dates.

Teams of Five Students (or less)

Each student will identify his concentration: COE EE CS **and section:** M2 T1 80 87

Design Specifications: Design a digital system with the following functionality:

Use the Logisim Keyboard (which interfaces with your laptop keyboard) to write text. Text will be displayed in two TTY Logisim displays. **See demonstration videos provided for the project.**

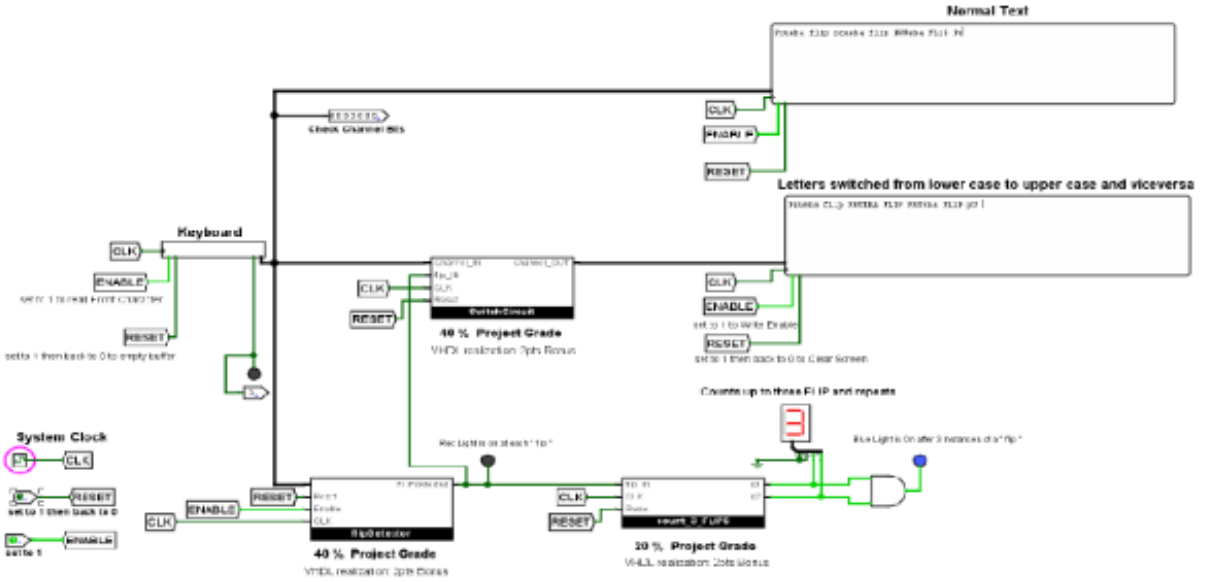
The text typed in the keyboard will appear in both TTY displays. After the string **flip** is typed, the second screen will start displaying the lower case letters as upper case letters and vice-versa. In this case only letter-characters will be allowed. If the string **flip** is typed again the text in the second screen will be displayed normally. When displaying text normally, as it was written, any character is allowed. The cycle repeats after **flip** is typed again. The system should accept the string flip in any case combination, for example: flip , FLIP , Flip , FliP, FLip , etc. It is required that the string flip be preceded by a space and be followed by a space (space bar click).

A **red light** will be ON each time that the flip string was typed and detected. Then it will revert to OFF.

A **blue light** will be ON each time that there has been three instances (consecutive or not) of the string flip. Then it will revert to OFF until three more instances occur.

Optional VHDL Bonus: A working VHDL realization, in addition to a Logisim implementation, is worth up to 6 bonus points.

A possible architecture of your system, using different subsystems, is depicted in the **next page.**



The word flip (with spaces before and after and in any case combination) makes switching each letter pressed in the keyboard from lower case to upper case. Use the word flip again and goes back to writing normally. Only letter characters are allowed when switching case.

You can be penalized:

If the system only partially works.

If you cannot explain what you did, or can't explain your logic, during the oral session of questions at the project presentation.

If you did not follow the technical report instructions given in the following pages.

If your report is sloppy.

If you do not provide a pdf of your project technical report or the project computer files.

If your design team reports that you did not participated.

If you do not form part of a group in Blackboard.

Concepts Needed:

Sequence Detectors, Shift Registers, Flip Flops, Combinational Logic, Counters, VHDL (to gain bonus points)

Formal Report Requirements:

- Describe, at a system block-level, the general design architecture that gives the desired functionality.
- Before performing the actual circuit synthesis, describe the functionality of every subsystem used in your design.
- Perform the circuit synthesis for each subsystem.
- Use **LOGISIM** (required) to run a simulation of the overall design and **verify** that your design logic works as intended.
- Try to conceptualize possible alternative designs and their capacity to meet the design specifications and constraints.
- Describe functional groups within your team (Overall Design Architecture, Circuit Synthesis, Simulation and Testing, etc). Students can participate, if desired, in all functional groups with different roles and levels of responsibility.
- Describe team member organization: Hierarchical (Project Leader, Senior Designer, etc) or Democratic, etc.
- Describe number of hours dedicated to the project and you project workflow.
- Bibliography.
- Turn in a **formal technical report.** The design documentation must include all the above parts. Also include a brief introduction and conclusion. Do not include theory. **Report Format** can be found at the end of this document.
- **Due date as shown in Bb in the schedule of exams, homework and projects**.
- Demonstration to the instructor is required: You will answer questions regarding the report and your design solution. You will run a simulation to test the design.

Groups of 5 Students (Same Section Students) Design Team Names and Tasks

Include course/section and whether you are COE, EE or CS.

1)

2)

3)

4)

5)

Report Format:

A. **Cover Page**

B. **After cover page include the Original Project as issued:** All Original pages handed by the instructor (this document exactly as it is, complete, for reference)

C. **Index**

D. **Introduction**: Summarize the desired functionality of your design. This part is basically given at the top of this document.

E. **System Architecture:** Describe, at a system block-level, the chosen design architecture that gives the desired functionality. (Also given in this case)

F. **Subsystem Functionality:** Before performing the actual circuit synthesis, describe the functionality of every subsystem used in your design.

G. **Subsystem Circuit Synthesis:** Include synthesis procedure for each subsystem circuit, including tables, maps ,etc, and circuit schematics in the report.

H. **Design Validation and Circuit Verification:** Use **LOGISIM** to run a **simulation** and **verify** that your design logic works as intended. Include your test campaign in the report (at least show the tested input streams versus output stream and light activation)

I. **Timing Problems:** Discuss whether you foresee any timing problems due to signal delay, or tested any, and solutions to these possible problems.

J. **Alternative Designs:** Try to conceptualize possible alternative designs and their capacity to meet the design specifications and constraints (if any). State advantages or disadvantages.

K. **Conclusions**

L. **Bibliography**

M. **Functional Groups:** Describe **functional groups** within your team (Overall Design Architecture, Circuit Synthesis, Simulation and Testing, etc). Students can participate, if desired, in all functional groups with different roles and levels of responsibility.

N. **Team Organizational Structure:** Describe whether it is Democratic or a Hierarchical (Project Leader, Senior Designer 1, Senior Designer 2, Validation Engineer, etc) organizational structure.

O. **Provide Team Design Rules:** For example, meeting frequency (biweekly, etc), meeting modality (in-person, on line, etc) , rules for members absences, rules for members not providing their fair share of work, etc.

P. **Project Management:** Describe number of hours dedicated to the project and completion of week by week tasks (projected versus actual).

Milestones (I to III):

i) TBA at class hours (M2, T1).

- Show the instructor a hardcopy of your project skeleton following the above Report Format guidelines. Leave sections in blank except for the sections: **A, B, C, L, M, N,** where you should include the requested information including team members.
- Questions regarding the project.

ii) TBA at class hours (M2, T1, T2).

- Show the instructor the project skeleton now incorporating additional parts D and L.
- Show any changes made to previous parts.
- Questions regarding the project.

iii) Project Presentation in Collaborate for secc CS2302-80 and COE2300-80 (TBA)

Project Presentation in Collaborate is at class hours for M2, T1, on the dates specified in the Schedule.

- Turn-in Final Project Report and simulation files in a zip-folder by email before your presentation.
- Demonstrate your Design Solution and Simulation to the Instructor.
- Answer any questions as formulated by the instructor during your presentation.

Milestone Compliance (instructor use)

Milestone/Level	Not Completed	Partially Completed	Completed with Corrections	Fully Completed
i				
ii				
iii				

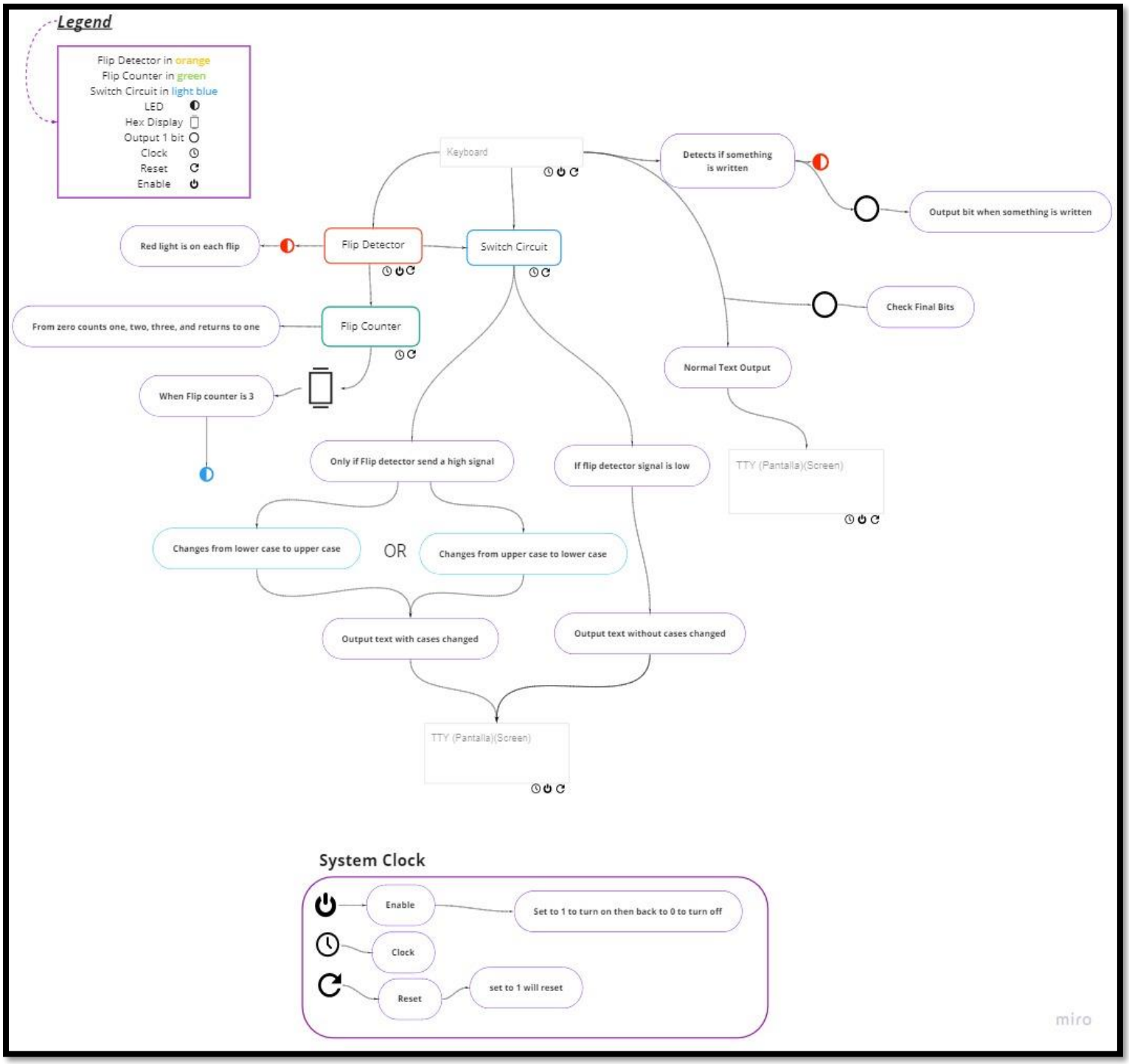
Index

Introduction 1
System Architecture 2
Subsystem Functionality 3
Subsystem Circuit Synthesis 5
Design Validation and Circuit Verification 25
Timing Problems 34
Alternative Designs 36
Conclusions 38
Bibliography 40
Functional Groups 41
Team Organizational Structure 43
Team Design Rules 44
Project Management 45

Introduction

Through this document the team will report the work put into the design specified by the professor. Explanations regarding the final design and how the product eventually met the requirements are also included. The main goal was to create and design a system that could flip the capitalization of letters entered via the LogiSim keyboard from on command while in synchronization with a clock. The required functionality for one part of the system was the detection of a code term “ Flip ”, regardless of case-sensitivity, in order to enable activation of other subsystems and notify such signals with a red light. It's important to mention that the requirements dictated for a space before as well as after the word FLIP. Then, the activation would enable the case switching circuit to change the text entered until the code word was entered again and so on. The detector circuit must signal to other sub-circuits tasked with counting the amount of times the code word was entered as well as warn every third codeword detection with a blue light followed by a reset of the count. Students had to analyze each functionality to create the integrated blocks with the function circuit inside. In the switching block, the string entered is to be changed only if the switching is enabled. Else, the string passes straight to the TTY display unchanged. In the case of the circuit being enabled, the system should distinguish between the lower- and upper-case letters to be the only ones that are changed. Special characters are passed straight to display instead. The result of the circuit is featured in two displays, one passing the string from keyboard input to the TTY display unchanged. The other passing the input through code word detection and switching boxes to be processed for display as they are processed. As an optional bonus, a VHDL implementation was implemented in individual boxes for case switching, " Flip " detection and three-time detection counter to replace the integrated circuits for programable circuits.

System Architecture



Subsystem Functionality

Every single subsystem composed for the wider circuit had multiple iterations, each using different principles given in class. Earlier versions are much cruder due to the limited subjects and topics covered in class up to that moment. Later versions implement different components and are much simpler and elegant by comparison. The earliest versions feature basic counters, logic gate arrays, multiplexers, and AND gate filters.

For the " flip " detector, the design had to send a positive signal as its output whenever it detected the exact string of characters " flip ". Once triggered, the signal from this subsystem would be used to activate the Switch Circuit and to add to the Flip Counter once. Methods for filtering out the characters and spaces could be the same across different designs. The processing of each character could vary greatly. Multi input AND gate filters would be the most likely solution to detect each character, given that the characters all have their own 7-bit identifier. Flip counters, on the other hand, should be a very easy design, as it would merely keep track of the positive signals passed on by the detector. This circuit should implement binary counters, the scale of which would depend on the maximum amount of flips expected to be used. As binary counters were discussed heavily in the laboratory, we could make the flip counter by taking queues from the designs presented and made in the lab, this should count from 1 to 3 once detected a flip, it will start on cero.

The " flip " detector was merely the first step in the entire puzzle for this grand circuit. Once we had a circuit that reliably sent out a signal each time the trigger was detected, it was time to work on the circuit that did the heavy lifting, so to speak. This brings us to the Switch Circuit. In essence, this circuit takes every letter and changes its case. That is to say, lowercases are turned into uppercases and vice versa. One important note was the difference in values between the lowercase and uppercase letters. Both groups, of course, followed the same sequence. What was noted was that the difference between each version of the same letter was always 32. Much like the " flip " detector, the Switch Circuit would need to pass through every entry made via the keyboard. Thus, the Switch Circuit would require a similar filtration system as the previous detector but not as strict. This Switch Circuit would only need to verify that the current entry is a letter that needs to be worked on, or a different character to be passed on without any modification.

The design for this Switch Circuit should resemble the following description. The circuit works in sync with the clock, as such the circuit only activates according with the clock and the flip detector. Multiplexers are used in order to allow for the circuit to be controlled through these variables. The next step of control in the circuit depends entirely on the Channel_IN input. The bits are all passed through a comparator system which verify that the bits correspond to a letter regardless of capitalization. This array of comparators is also responsible for the enable for the final Multiplexer, which decides to allow the modified values to pass or the unmodified values to remain unchanged and pass through. If the detected characters are letters, the circuit passes the data on to a different set of comparators, which add or subtract the aforementioned 32 to the value.

Thus the capitalization of the letter is effectively swapped. The resulting value determined by the arrays of comparators is finally fed out through the output.

The Switch Circuit should feature many comparators to make sure that the character being passed through was a letter, regardless of case sensitivity. Multiplexers would be used in order to maintain the circuit active or passive depending on the signals passed on by the FLIP detector. While inactive, the Switch Circuit should simply pass on the data for all characters. Once activated, the Switch Circuit needed to act upon all letters that passed through it, while bypassing any other characters it did not need to change.

Subsystem Circuit Synthesis

Prototype Original Flip Detector & Counter : made by Sebastian

The basic functionality of this subsystem involved AND filters to detect the 7-bit values of each letter of the word regardless of capitalization or lack thereof. Space characters were also necessary to count, both the front and back ones. Initial designs utilized only the letters for detection and spaces (or other characters not in the keyword) as a reset in case the necessary letters were not fully counted. These early designs are compensated for as both spaces needed to be accounted for when designing the detector. In one of the prototypes, a counter was set up for both the letters (regardless of capitalization) and another for both spaces. The letter counter reset at 4; the space counter at 2. A reset from each counter counted as a "point" which triggered the final detector counter, which also counted to two. If an unwanted character was ever detected, all counters were universally reset to 0. Each reset from the letter or space counters counted as a "point" for the final confirming counter. The initial space would be detected, counting for 1 in spaces. The four letters in " flip " would soon follow suit, eventually resetting the letter counter and sending the first confirm signal to the final counter. Immediately after, the last space would be detected, raising the space counter to 2 and resetting it. This would in turn give the final counter its second signal, causing it to reset and sending a signal that a " flip " had been detected.

The later versions for the " flip " detector feature a much more efficient approach through the use of more elaborate components and their properties. These versions utilize digital registers in order to store multiple bits at once instead of staying with the same one-bit flip flop counters. Multiple registers can be placed in series in order to store a sequence of multiple bits over the course of multiple ticks. The function of this version is much more simple compared to the early versions. Seven registers are placed in series, each should store the appropriate bits as the " flip " is entered. Each stage of the series of registers features a filter, which detects if the appropriate characters are in place. Once the " flip " is entered and enough ticks have passed, each character and space will be stored in their designated registers. The filters will all return a 1 which lead into the final AND gate. This finally results in the entire detector returning a 1.

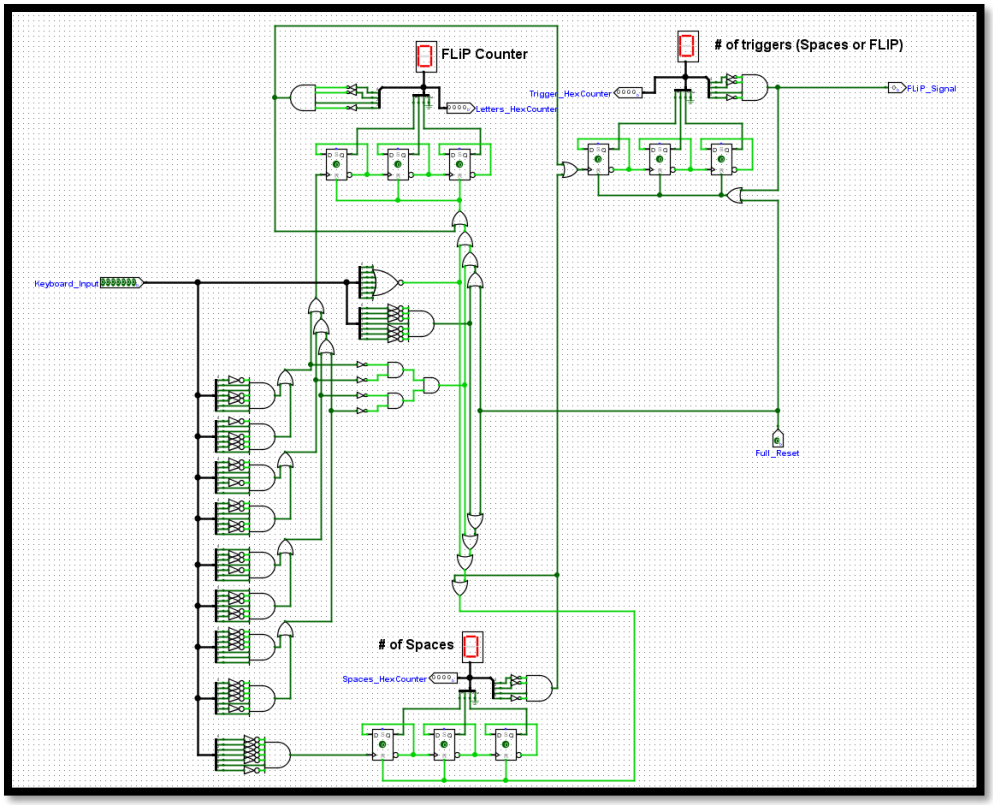


Figure 1: Original Flip Detector.

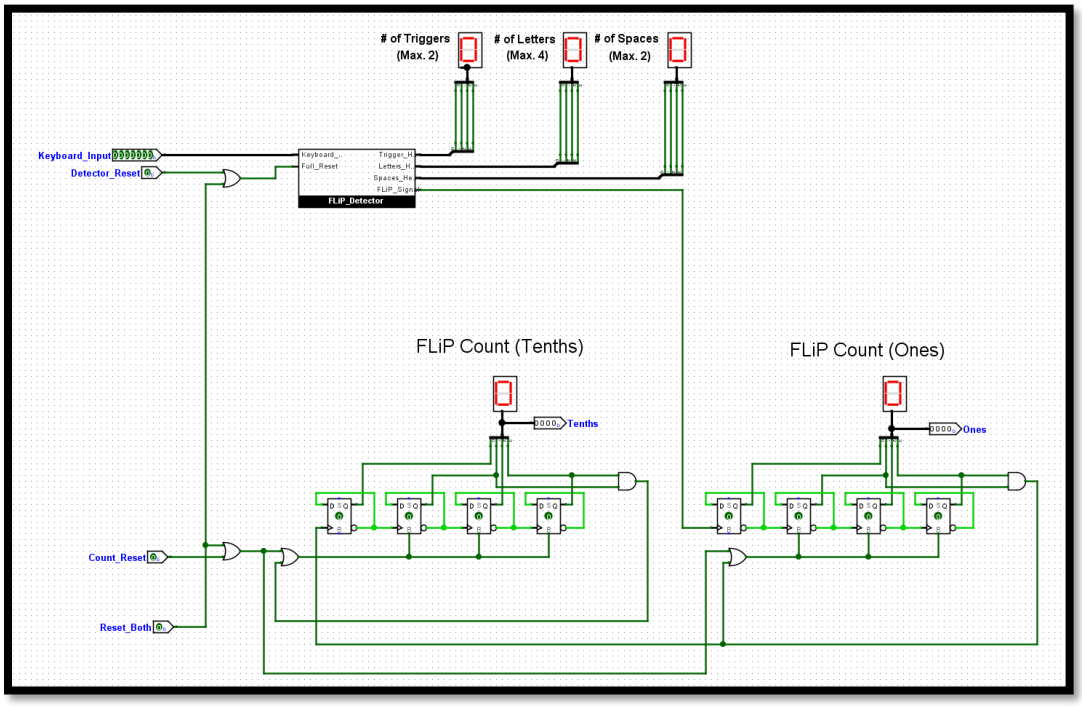


Figure 2: Original Flip Counter.

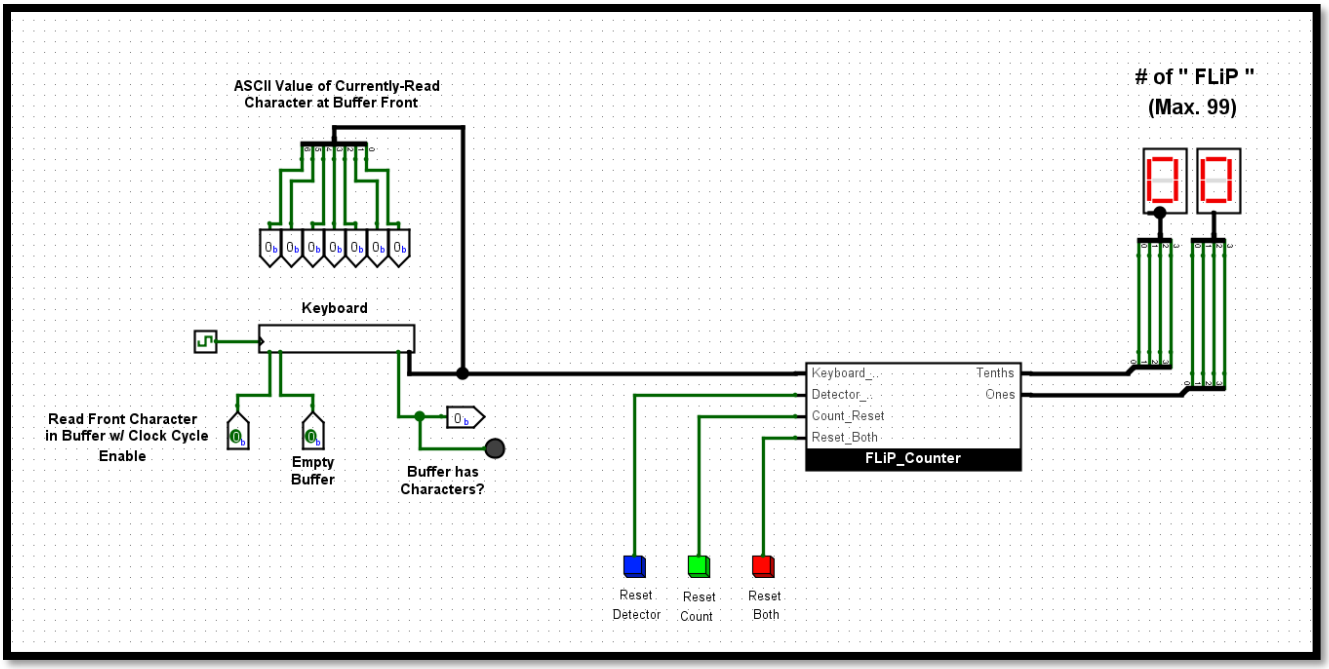


Figure 3: Flip counter that counts to 99.

First Prototype for Switch Circuit Version 2: made by Jessica

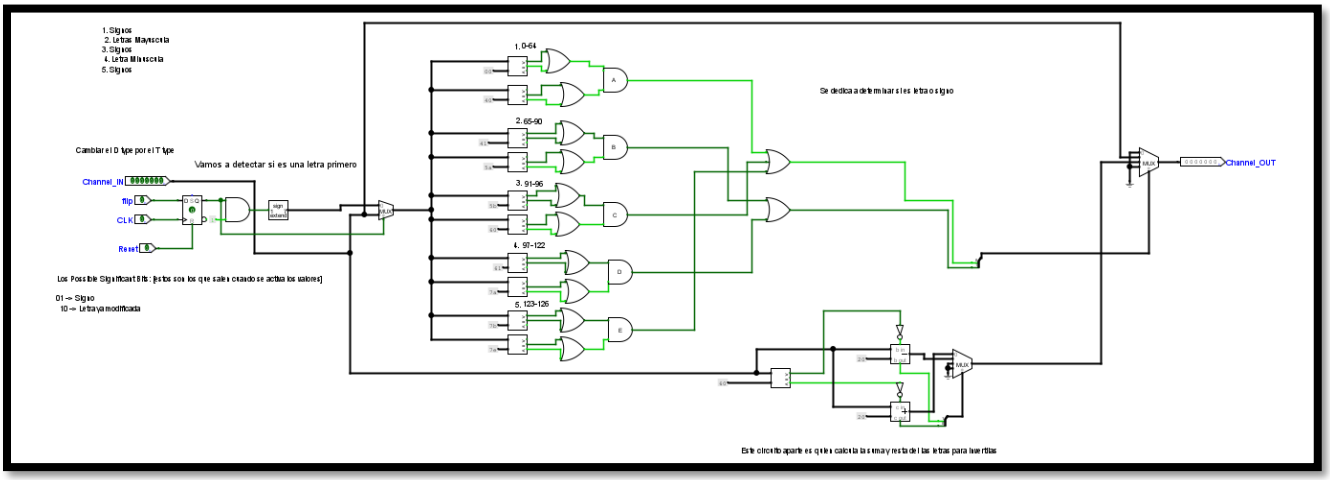


Figure 4: This is the first prototype for switch circuit for version 2, in which it includes the new feature to pass the signs that are entered by the user and switch the letter from upper case to lower case or lower case to upper case.

Looking at the schema given by the professor, the first part of the block is included. This is the Channel_In, meaning each of the 7-bit letters or signs that will be processed, flip that enters

a single bit that detects with 1 when it is found the word flip with spaces, the clock and reset. Once the input values are clear, the output values must be one that the Channel_out will call this output is 7 bits that consists of leaving the letter already modified or the sign unmodified, at the same time keeping clear that the letter will not be modified when the flip is not detected or if it is detected a second time.

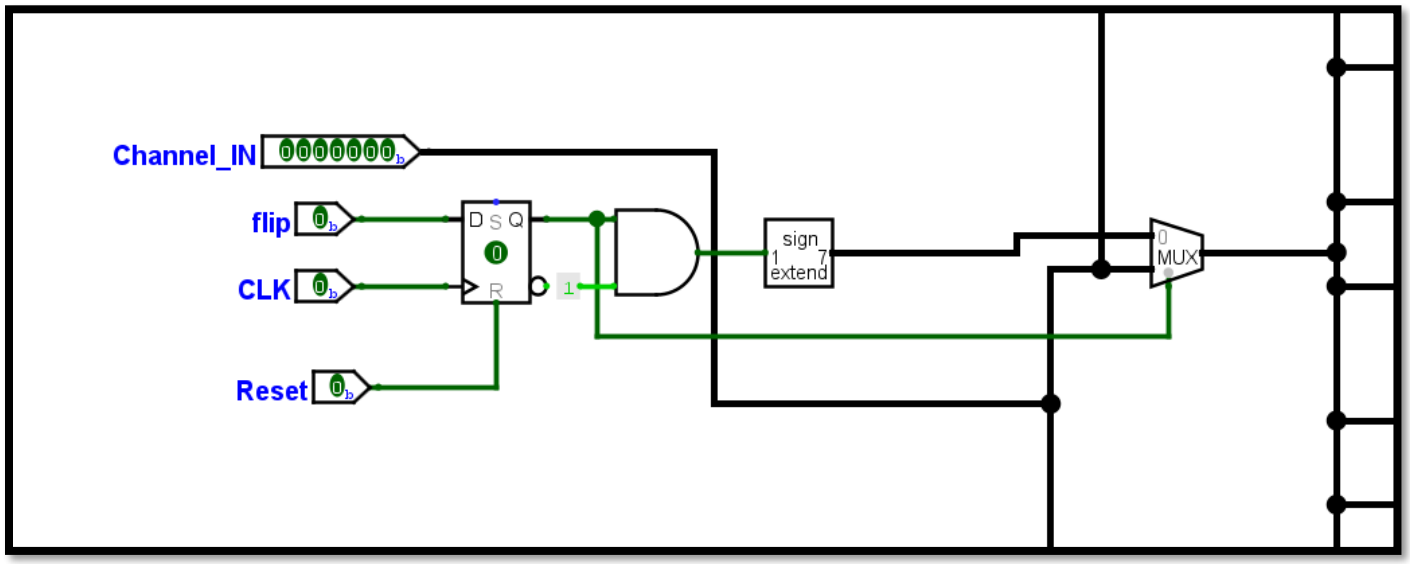


Figure 5: Image shows the input values in use and the type of flip flop, as well how the system of activation of the circuit mas made in more detail.

Starting with the type of flip flop on use, we thought that it was better to use the D type flip flop, since according to the truth table that same value entered will be its output, making it easier to deal with (this plan backfired since it will affect the second flip once it was activated and the output will continue to switch the letter it's not the most effective for the project). Then passing the value of the flip flop to the AND gate (to which this contains a constant one), once detected the value passed will be entered through a bit extender, this block will take the AND gate value to pass it as a 7 bit so then the output will the output value of AND. The multiplexer is a one selected bit, decides to either pass the Channel_In value or pass the value of the AND gate, it's selector will be the output of the D type flip flop that will help to identify the which input will pass as the output of the multiplexer.

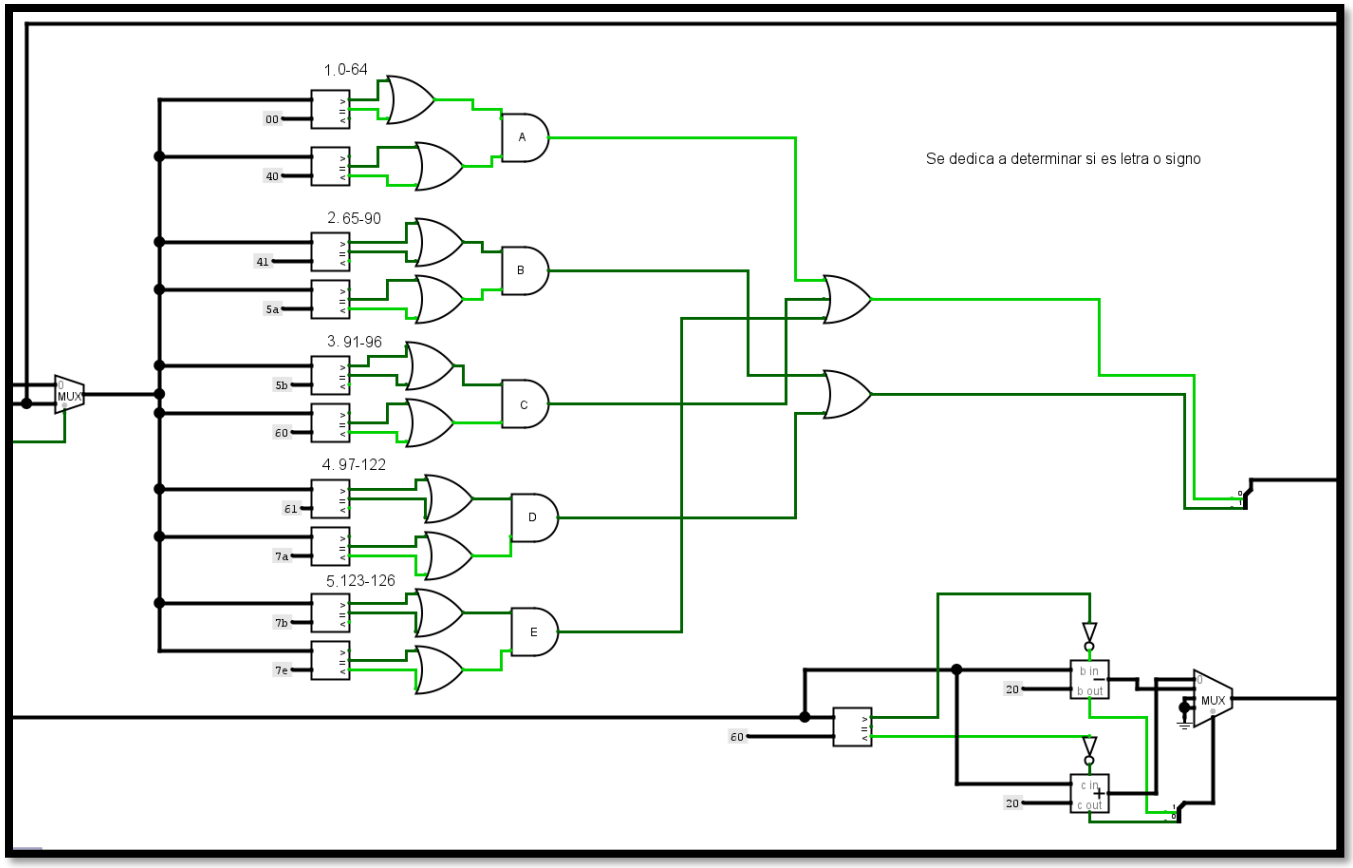


Figure 6: The part contains the comparison of the sign and letter and will be passed to the multiplexer. Meanwhile on the lower part of the comparison there is circuit made for the modification of the letters. 0-64, 91-96, 123 - 126 symbolizes the sign and 65-90, 97-122 symbolizes the letters.

Once the output of the multiplexer is in use, meaning that it already passed the stage of which input will be the output, each box will take two inputs. One will be the value coming out of the multiplexer and the other will be the number placed on top as comments that are in decimal, will be converted as a hexadecimal number that will be constant. This will occur to each of the boxes with the respective constant number to use such as 0, 64, 65, 90, 91, 96, 97, 122, 123, and 126. The main purpose of this comparators is to identify if the input value is in the range of the selected constant values, taking for example the first box: "if input value is greater or equal to zero and if the value is less or equal to 64". This exact pseudo logic was used in each one the boxes (there will be two boxes to complete the requirement). Once the values are passed the output of the AND gates that identify sign and letter will grouped with an OR gate to pass to the selector of the 2 selected bit multiplexer. This is because the value to be entered is to be a sign the selector should pick 10 and if is a letter will be 01 to pass the letter modified. The comparator part to identify the letter and sign comes from the ascii table the decimal numbers.

In the lower part of the picture there is another comparator box with the constant number of 96 in hexadecimal. There are two more boxes: one is for addition, the other is for subtraction. The comparator will take the constant value. The channel_In value is passed through and will identify if the letter is greater than 96, if it is, then it will be classified as a lower-case letter. If the letter is less than 96 then it will be classified as an upper-case letter. This idea began by looking at the ascii table. While looking and analyzing the difference of range of the selected upper case and lower-case letters, uppercase being the lowest value such as 65 to 90 and lower case is from 97 to 122. By taking the letter 96 is the in between value of the range of upper and lower case in which will identify correctly each value entered. For the addition and subtraction boxes have another constant input that is a 32-hexadecimal value. Following the example below.

Table 1: ASCII Table

Characters	Decimal Number	Binary Number
A	65	1000001
a	97	1100001
Space	32	0100000

Calculations made:

97 - 65 = 32

65 + 32 = 97

97 - 32 = 65

As seen on the table below the letter A has the number 65 and a has the number 97 in the ascii table. When taking these two numbers there is a difference of 32 that is the space character in the table. Taking these 32 decimal numbers and add it to the 65 decimal number of A it will give a result of 97, meaning that is the letter a in lowercase, this will be switch of upper case to lower case and vice versa. Now this is the theory side of why and how it will be working on the subtraction and addition boxes, but instead of using the decimal number this will be working with binary numbers. In more detail the box contains a carry in and a carry out, the carry in is when the adding of a single bit and there is a residue will be place on top of the next digit to come, once that each bit is added then the residue of the bit that surpass the 7 bits total will be the carry out. Also, always the second bit going from right to left is the one to change, it may change to 0 to upper case or 1 to be lower case.

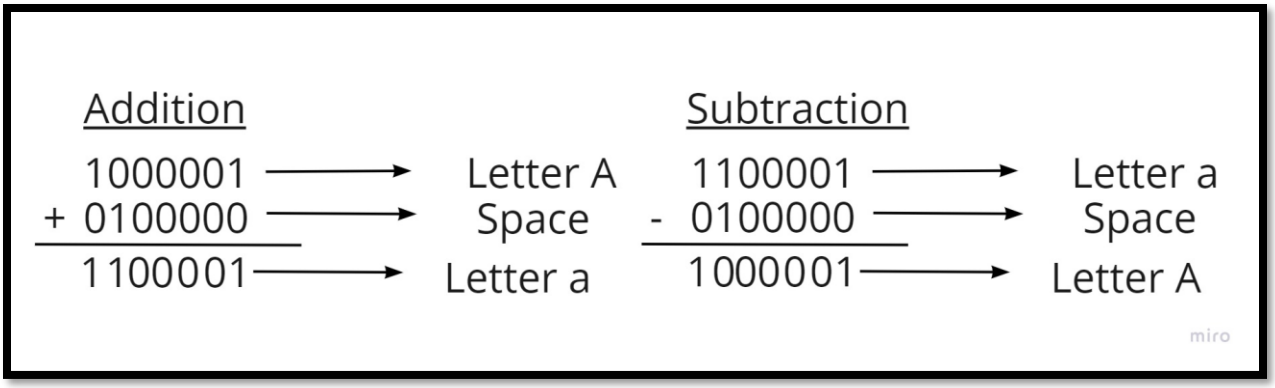


Figure 7: Seen in the picture this is how the switch of upper case and lower case and vice versa will be done for the switch circuit.

Each box of comparator, addition and subtraction will take 7 bits and will be unsigned. The comparator will activate when the value entered to compare is either greater or less than 96. For example, if the Channel_In is the letter A, then the box will throw a one to less than, for that the one will be negated and connected to the carry in of the addition box, it will be negated for the simple reason that when carry in is cero won't affect the result and the carry out will be cero as well meaning no single bit residue. This is done as well for the greater than or in other words for a lower-case letter, but instead it will subtract.

The carry out will determine the activation selector for the two-bit selected bit multiplexer. Once it determines the value will automatically pass for the next multiplexer that will either pass a sign or a letter that has been changed. Utilizing the result of the sign and letter detector will help to pass the output needed.

Second Prototype for Switch Circuit Version 2: made by Jessica

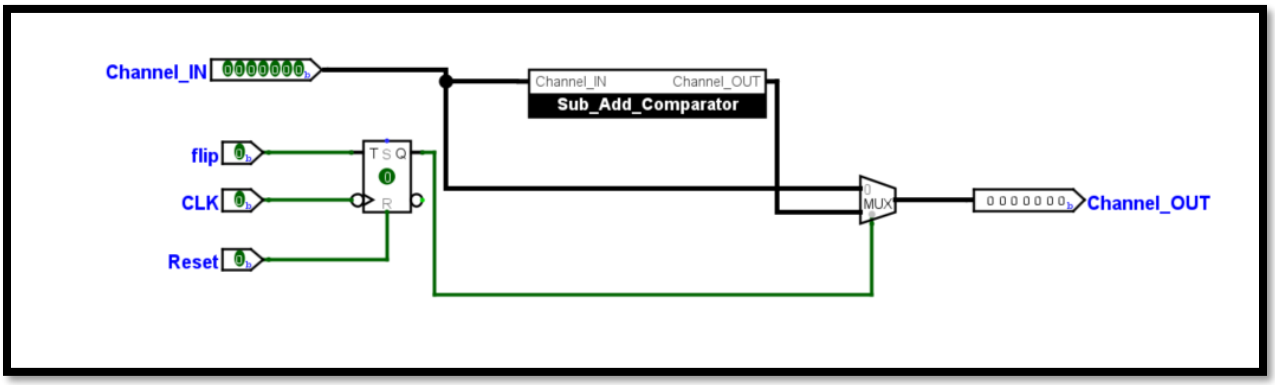


Figure 8: Second prototype for the Switch Circuit, but with extreme changes. Still contains all the features of the beginning established.

The change made for this prototype was the flip flop used. The way the Sub_Add_Comparator was built to establish the comparison of the letter or sign, the final multiplexer changed to a one selected bit instead or the two selected bit make it more compacted choice. Starting with the T type flip flop used for the switch circuit. The change was made since the other D type flip flop that was used didn't provide the change in the moment needed and not all letters changed when flipped was detected, for that the T type flip flop worked the best for the result wanted.

T	q	q*	Operation
0	0	0	q*= q No Change
0	1	1	
1	0	1	q*= q' Toggle
1	1	0	

Figure 9: T type flip flop truth table taken from Blackboard module.

By looking at the truth table given for the T type flip flop, when the flip gives an input of cero the state won't change meaning it will not pass the changed letter, instead any symbol or letter entered will pass as if nothing changed. Once the flip gives an input of one then it will activate in the multiplexer to pass the output of the Sub_Add_Comparator box that is the letter changed from upper to lower case and vice versa. The benefit of the T type flip flop is that once it enters an input of cero after the flip was detected then by the truth it won't change and it will stay with the previous present state. As well, since flip was detected once the second time it's called will toggle meaning that in the first flip detected will give a present state of one, meanwhile in the second flip will be a cero.

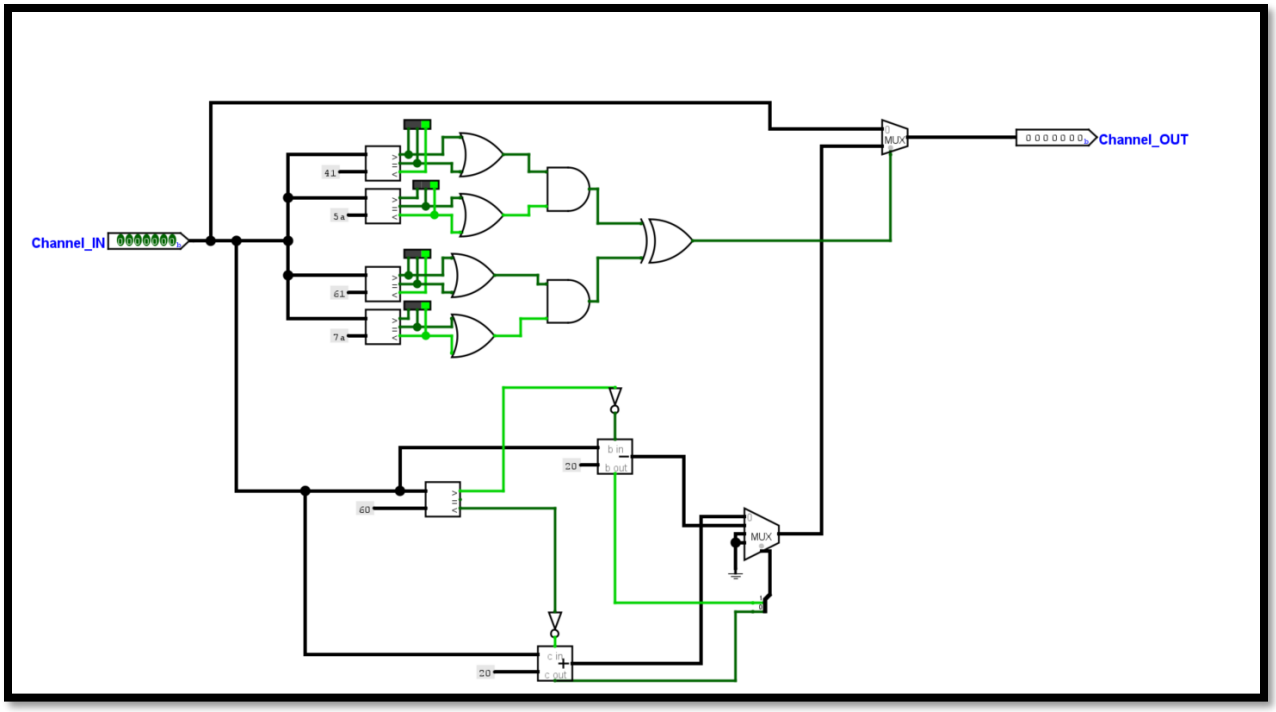


Figure 10: Second Prototype for the Switch Circuit, this is the Sub_Add_Comparator.

The only input needed is the channel_in that is 7 bits, this input value will be first compared if it is a letter that is in between the range of "if 65 is greater or equal and if 90 is less or equal" this is for the first two comparator boxes that will identify the upper-case letter. As for the second two comparator boxes will go in a range of "if 97 is greater or equal and if 122 is less or equal". Then the two OR gate will connect to the AND gate, this is for both comparison boxes. Once connected to the AND gate, both AND will connect to the XOR gate, if both values are the same from the AND gate it will give a cero, if different then a one will be the output of the XOR gate. This gate would determine the selected input for the multiplexer, if value selected is one the changed letter will pass, if not the letter not changed or other sign will pass if the selected input is cero. The same structure of the circuit from first prototype to change the letter for lower case to upper case and vice versa. The output will be passed later the Switch Circuit box. This was simulated and tested, it managed to work after a delay was fixed.

Switch Circuit Version 2: made by Jessica

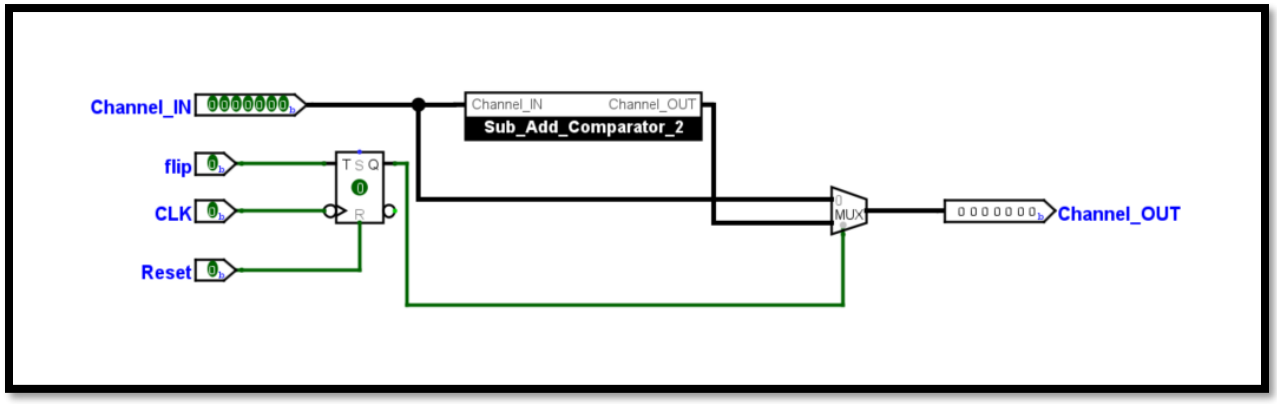


Figure 11: Final Switch Circuit version it stayed the same structure for the main part, the change is based on the Sub_Add_Comparator_2

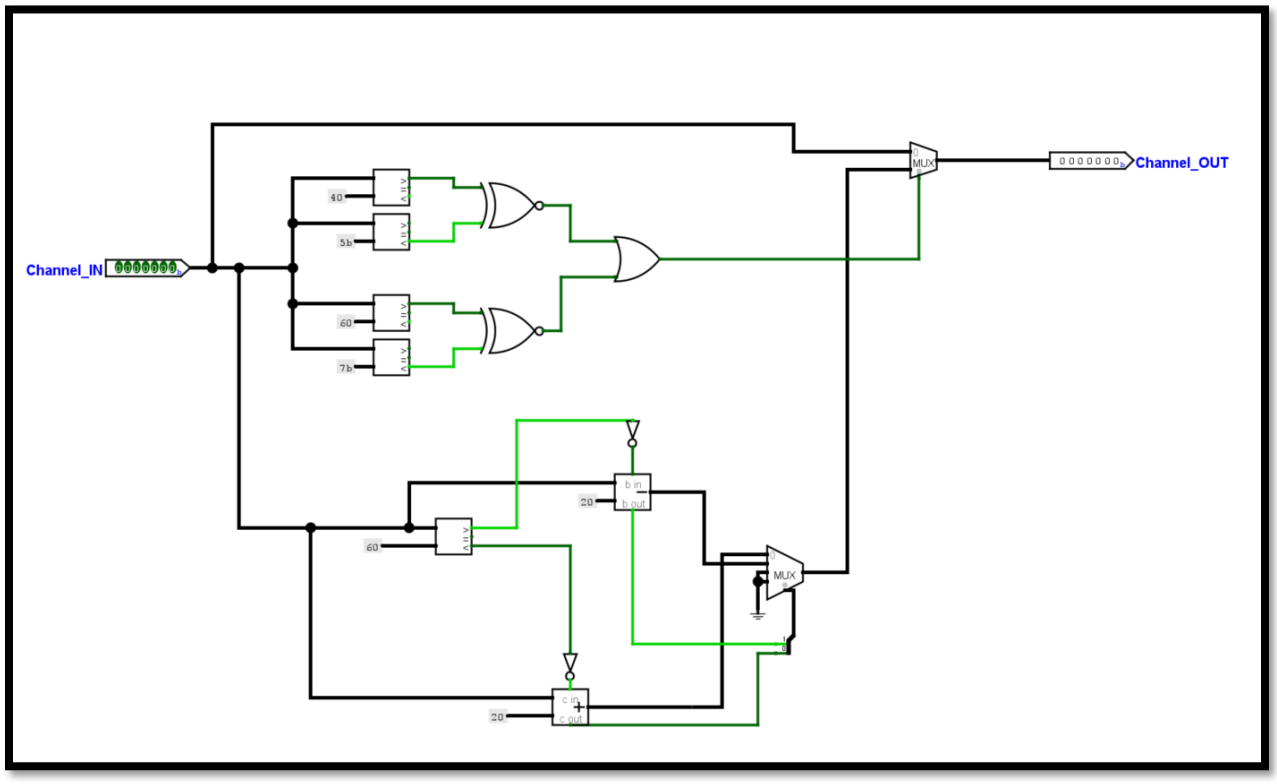


Figure 12: This is the Sub_Add_Comparator_2 with its final change.

This is the same second prototype, but with a change on the Sub_Add_Comparator_2, provided a change on the gates used for the comparator, this time the range goes in both comparators for the upper case as "if 64 greater than or 91 less than " and for the second boxes for lower case as "if 96 greater than or 123 less than". This will cover 65 to 90 and 97 to 122 for both type of cases. What changes are the number of gates being in use, instead of having four OR

gates, two AND gates and one XOR gate. It was reduced to two XNOR gates and one OR gates, in which this is better, meaning it requires less material and reaches to the outcome wanted. The following XNOR gates will provide a one if both inputs are the same, and a cero if they are different, once passed both output values of the XNOR gate to the OR gate this will indicate the selected input value to pass.

This is the final version that is based on both prototypes provided, this version is the most efficient type for the Switch Circuit, that provides the feature of passing the signs or letter when detected. It covers what the project indication asks and more.

Prototype Scheme of Circuit: made by Juan

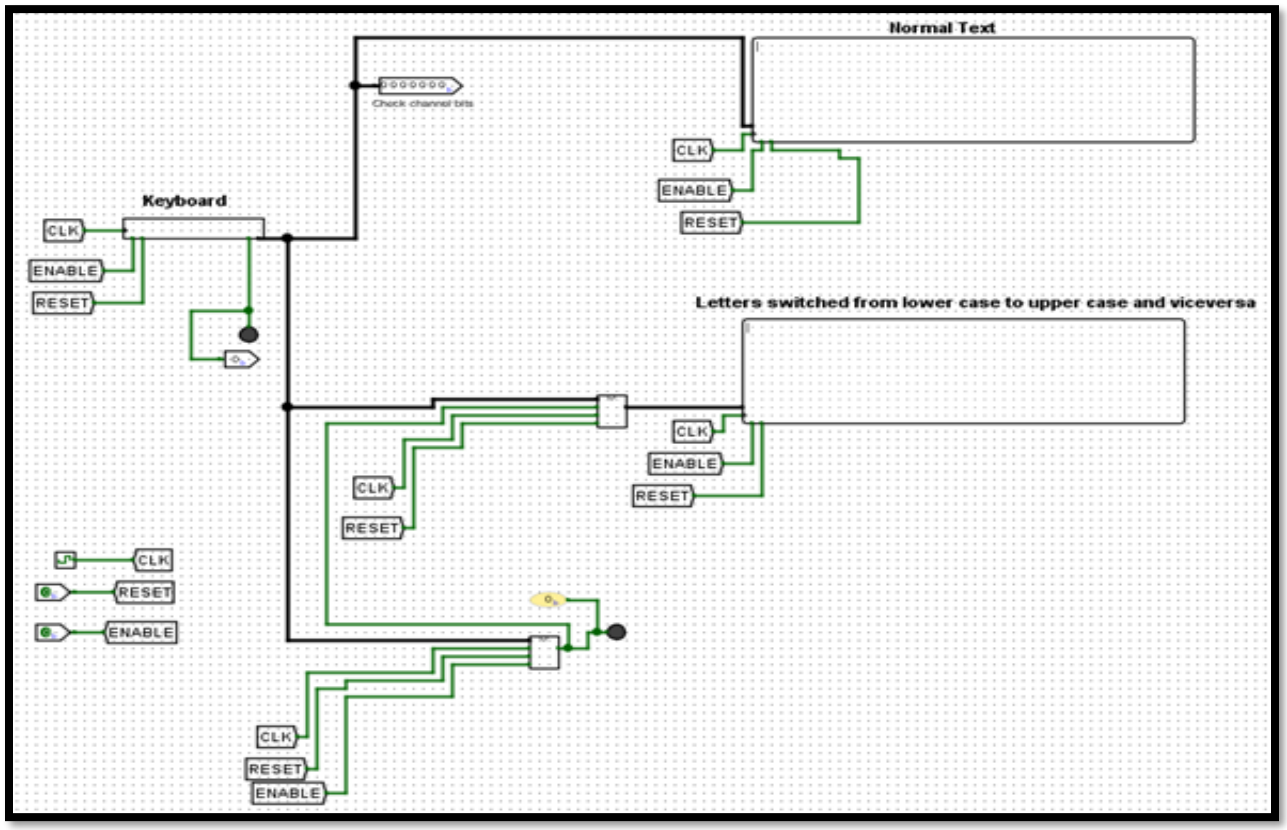


Figure 13: A Proof of concept for the circuit

This is a prototype of how the complete circuit should work and how it is installed. In which this is counted as a prototype because it did not work because it is assumed that after the flip there is a space, and it begins to change from uppercase to lowercase and from lowercase to higher. But in this one he does it after two spaces after the flip.

Prototype Flip Detector for Second Version: made by Gabriel

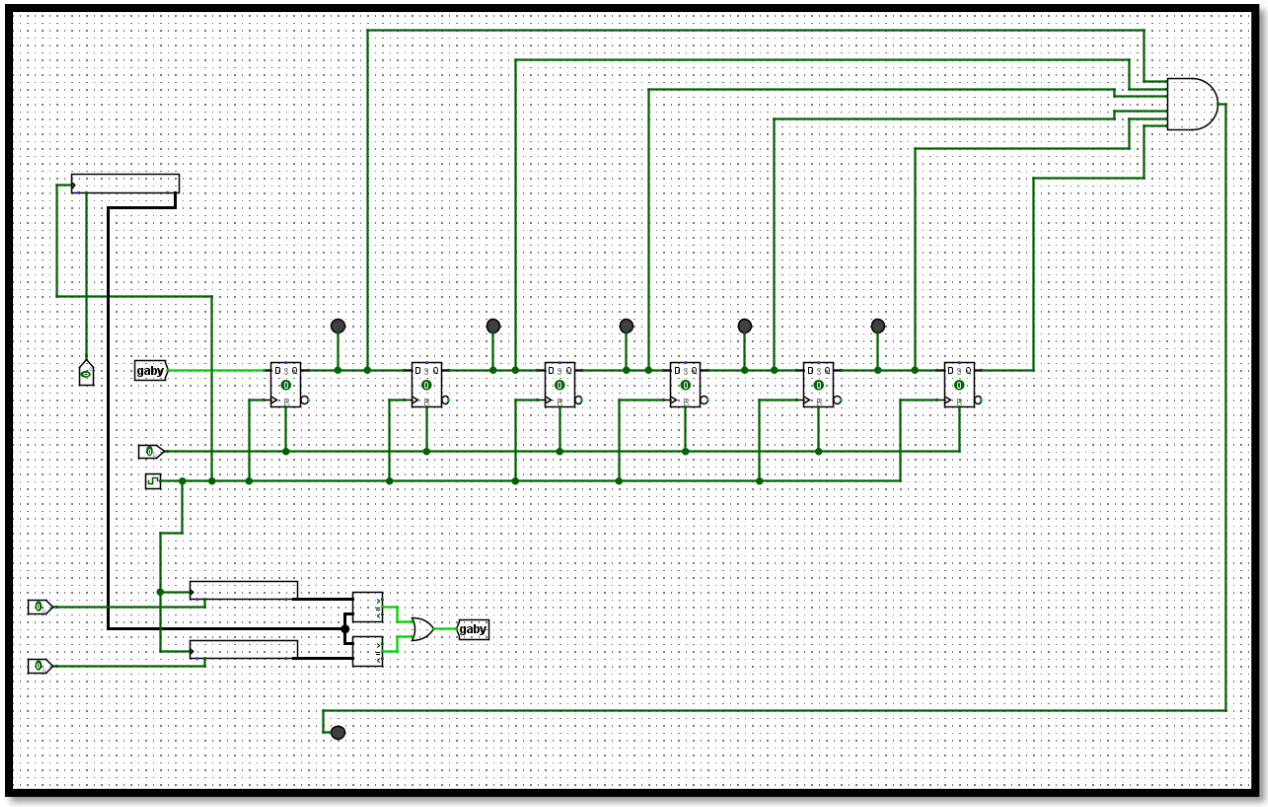


Figure 14: This is the Sub_Add_Comparator_2 with its final change.

Detector first version was a prototype to have Flip as a keyword using the keyboard inputs to compare each symbol and report detection to the single shift register. After notifying detection of the same character, it was saved in the first Flip flop.

Steps of operation consists of the following processes:

First the system enters Keyboard character by clock rising edge activation, so the character is compared with the ones in the manually preset letters at the comparator Keyboards. Then comparators send one if the character entering is equal to the preset and is sent to the first Flip Flop on the shift register chain for sequence detection. As part of the clock use nature, at the next Clock rising the previous steps are repeated, and the bits saved in the shift register move to the right so the next 1 or 0 can come in without trashing the first bit. The detection for sequence detection is achieved when 6 consecutive comparations result in 1 for equal characters at their clock rising so they are sent to the shift register and the and is then logically having a 1 output, so detection is noticed.

Defects of the Design:

The comparators have a limited comparison base and cannot repeat the before compared reference letters. If words are not well aligned, the word for activation may be entered and not be detected, and consequently it doesn't work if you are not purposely entering the same letters in the same order. Also, detection of the first character may not be detected.

Prototype Flip Detector 2 for Second Version: made by Gabriel

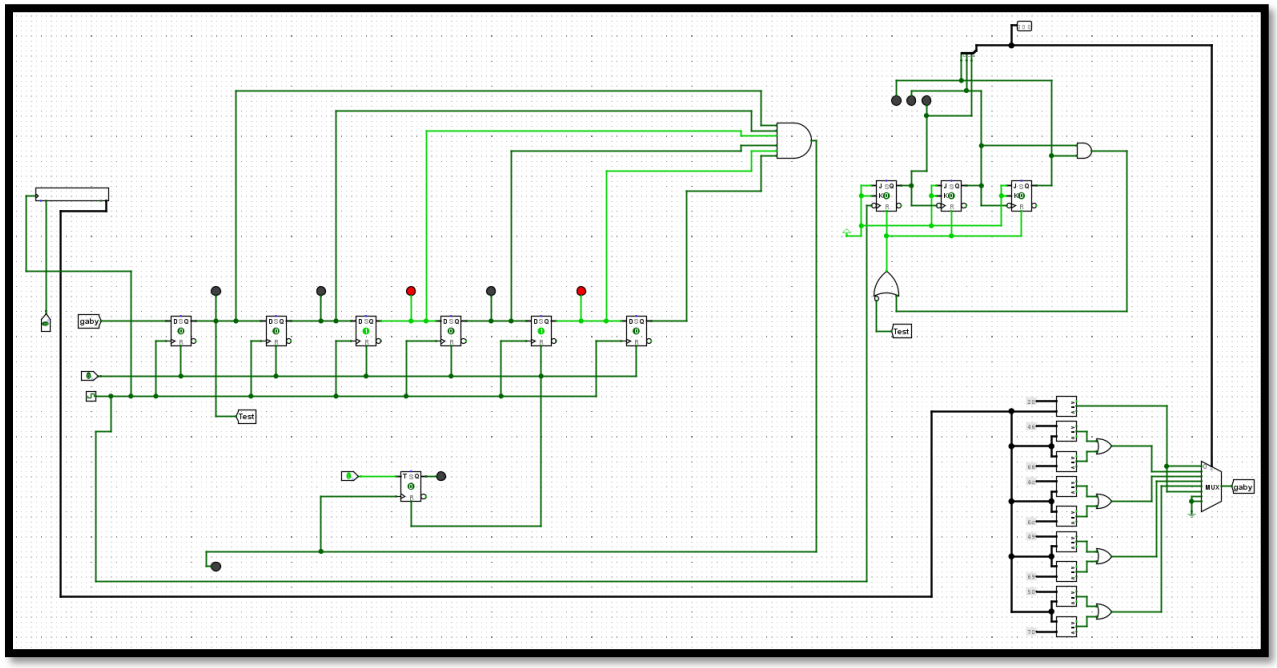


Figure 15: A second itineration of the Flip Detector, making use of comparators and binary counters

Steps of operation idea: Version 2

System enters Keyboard character by clock rising edge activation, so the character is compared with the hexadecimal values put in at the comparators to upgrade from a letter to letter to a 7bit to hex number. Constant value numbers for comparison can be used for multiple times of detection. The counter is also activated by clock but on the falling edge to count and move as selector bits though the MUX options of the multiplexer comparison result, only keeps comparing if the character on entry matches the constants. Comparators send one if the character entering is equal to the preset constants and is sent to the first Flip Flop on the shift register chain for sequence detection. At the next Clock rising, the previous steps are repeated, and the bits saved in the shift register move to the right so the next 1 or 0 can come in without trashing the first bit. Detection is sequence detection is achieved when 6 consecutive comparations result in 1 for equal characters at their clock rising so they are sent to the shift register and then logically having a 1 output, so detection is noticed. Detection also means the counter went from 0 to 5 without a mismatch between keyboard values and comparator constants. If not, the counter resets to 0 and starts up again.

Defects of the Design:

Counter timing difference is too much, and this causes use of the wrong MUX selected pin which provokes a reset on the clock so it ends up stuck or always resetting to 0. The delay causes misalignment of correct MUX pin with words even if the code word enters, the word for activation may be entered and not be detected. Circuit is too dependent on the counter for results to go on the shift register which affects detection of the first character being the only one detected.

Flip Detector for Second Version Final: made by Gabriel

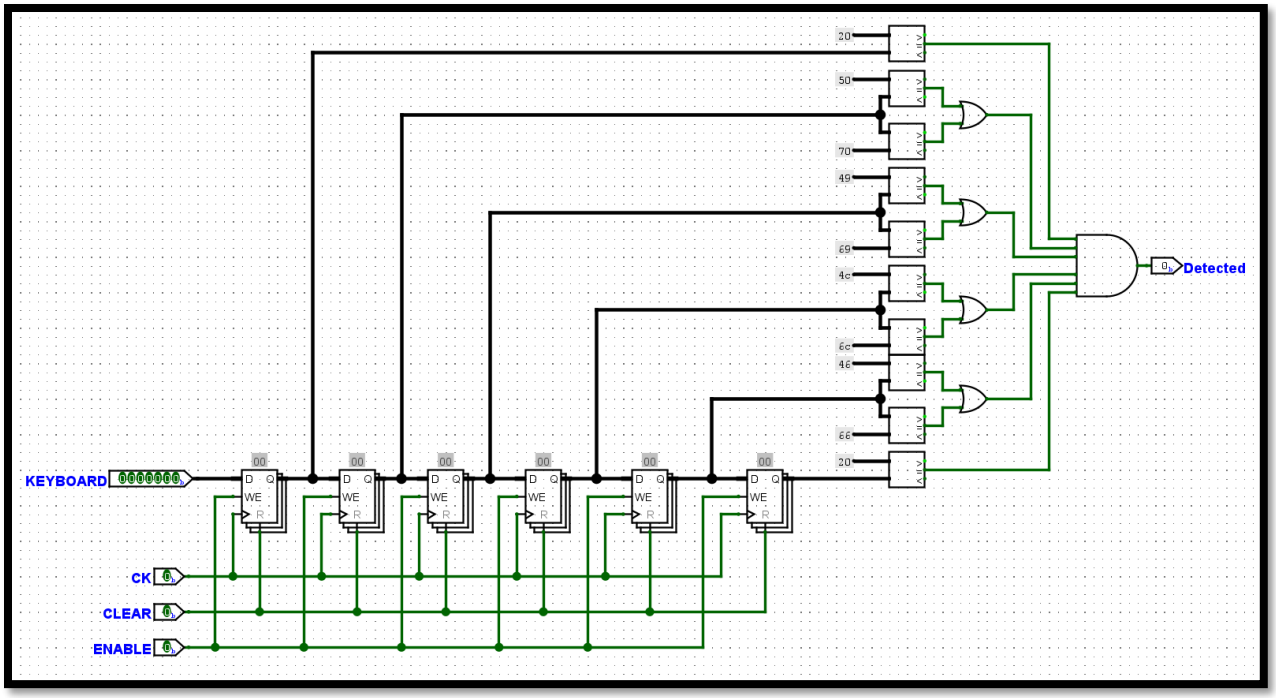


Figure 16: Flip Detector with comparators, it detects if is in the range of the space and each individual letter of the flip.

Steps of operation:

The circuit receives the keyboard character at the clock rising edge activation, so the character enters the first shift register of the chain holding 7 bits at a time from Keyboard. Character asci value being on hold is compared with the Hex value as a constant at the comparators. Comparators send one if the character entering is equal to the preset and is sent to the output AND for sequence detection contribution. At the next Clock rising, the previous steps are repeated, and the bits saved in the shift register move to the right so the next asci binary value can come in without trashing the first 7 bit. Detection is sequence detection is achieved when all 6 comparators get 1 as a result for the AND to produce an activation notice thanks to the one.

Defects of the Design: None after the work done in past days.

Flip Counter for both Circuit Version: made by Savier

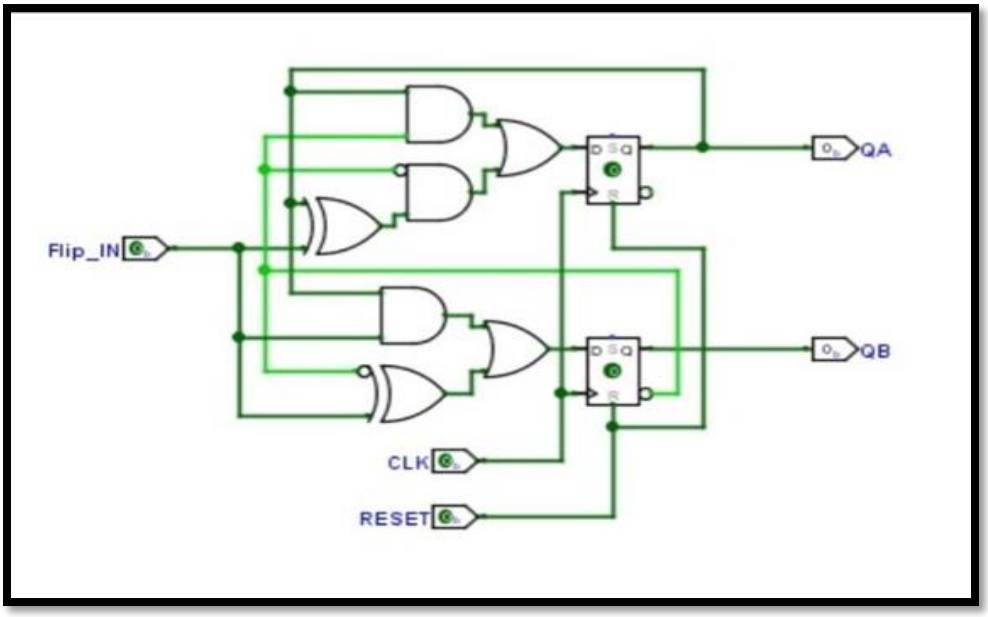


Figure 17: Flip counter made from D type flip flops. It will count 0, 1, 2, 3 and returns to 1 and continues the count.

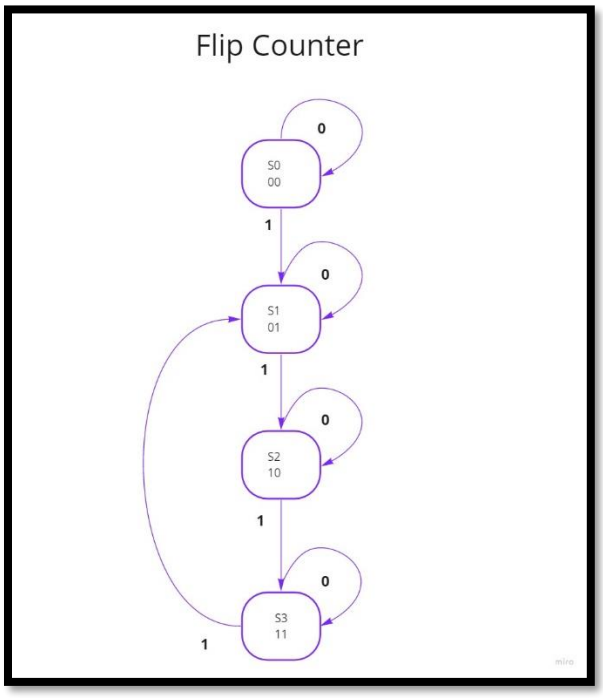


Figure 18: State Diagram of the Flip Counter

TABLA DE ESTADO

	Estado Presente		Input	Flip Flop		Estado Futuro	
	QA	QB	X	DA	DB	QA+	QB+
0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	1
2	0	1	0	0	1	0	1
3	0	1	1	1	0	1	0
4	1	0	0	1	0	1	0
5	1	0	1	1	1	1	1
6	1	1	0	1	1	1	1
7	1	1	1	0	1	0	1

Figure 19: State Table of the flip counter.

K-MAPAS

QAQB/X	00	01	11	10
0	0	0	1	1
1	0	1	0	1

DA= QAQB' +QAQBX' + QA'QBX

QAQB' + QB(QA ⊕ X)

QAQB/X	00	01	11	10
0	0	1	1	0
1	1	0	1	1

DB= QAX + QBX' + QB'X

QAX + (QB ⊕ X)

Figure 20: Input K-Maps of the flip counter.

This circuit would be a normal counter that counts from 0, 1, 2, 3 and returns to 1. In doing this, a state diagram had to be made to help form the circuit. Then it is created with the table that is used in excel. This counter is done using the D type Flip Flop but it can also be done using any flip flop if using a state table and the flip flop excitation tables.

Prototype Flip Detector First Version: made by Savier

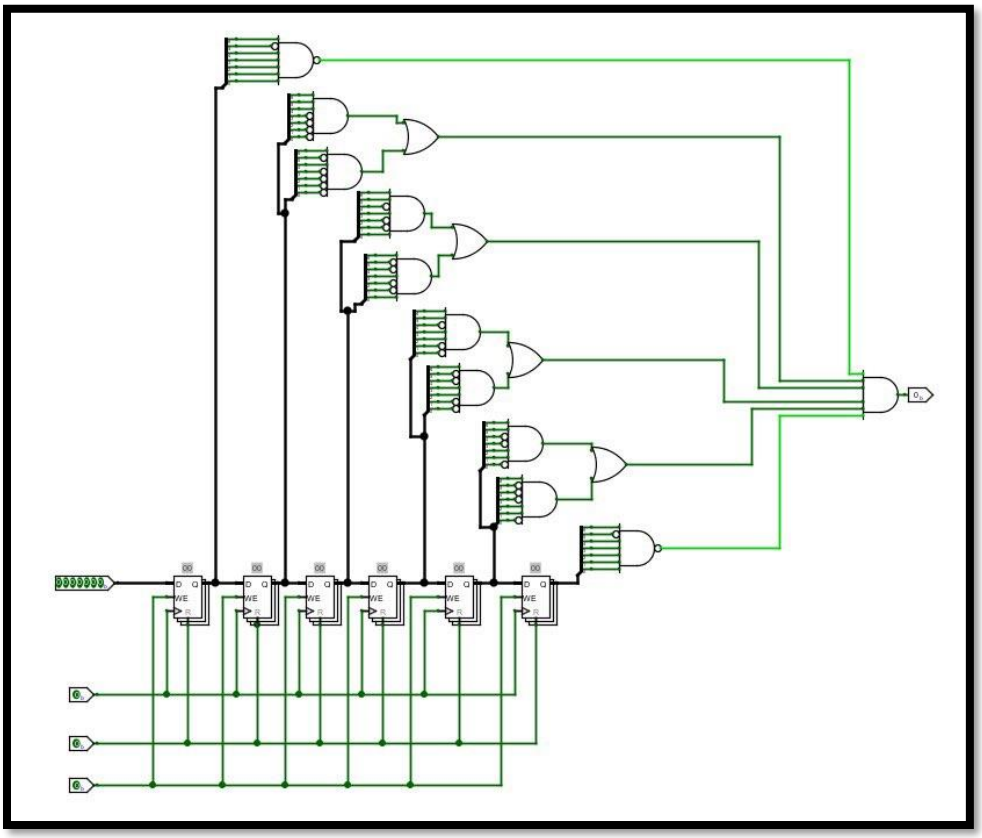


Figure 21: A Flip Detector utilizing sequential Shift Registers and bit filters.

With the flip detector first version prototype we can see that we have some registers connected to each other. This lets us do the flip flops work but with more than one bit. By having these signals, we can introduce two splitters in the output of each register. This allows us to know which is the uppercase and which is the lowercase so that we get the "flip" signal. This will then be input to an AND output to send a signal that a flip was detected.

Flip Detector Final First Version: made by Savier

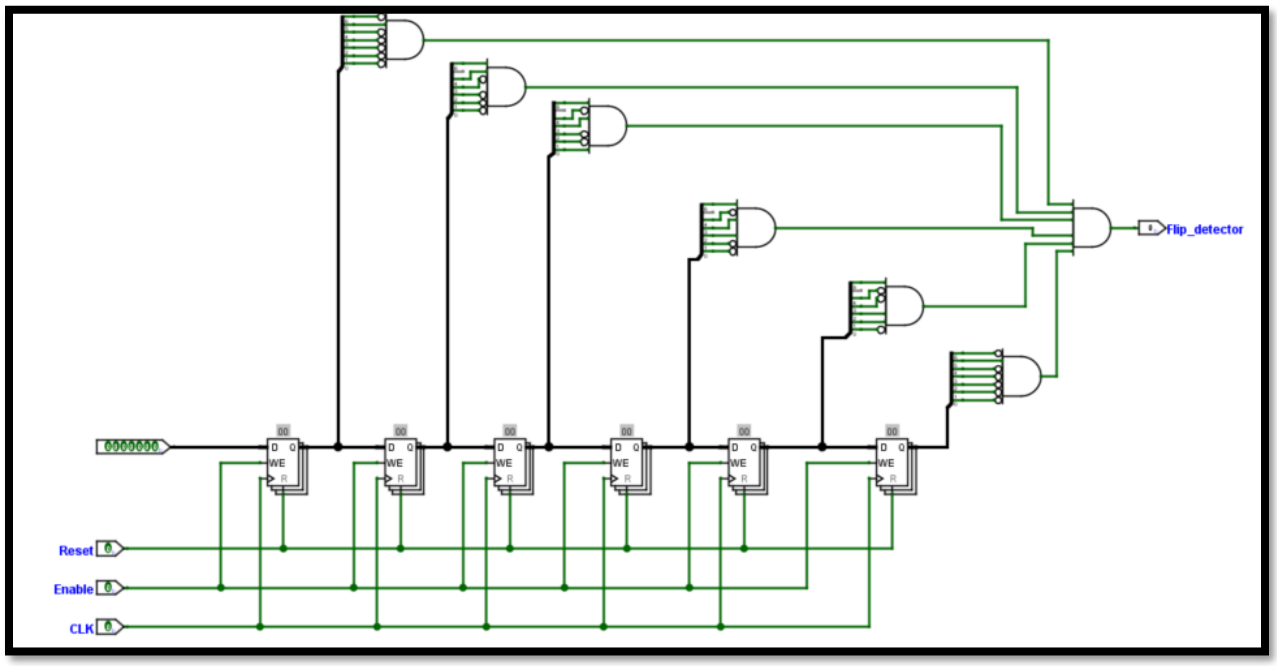


Figure 22: This is the final first version of the flip detector.

In this final version of the same first prototype the only change made is the last two gates that contained a NAND gate for detecting the space, was replaced with the AND gates, but instead it will be negating everything in other words it will be occurring the opposite of the negated NAND gate.

Switch Circuit First Version: made by Michael

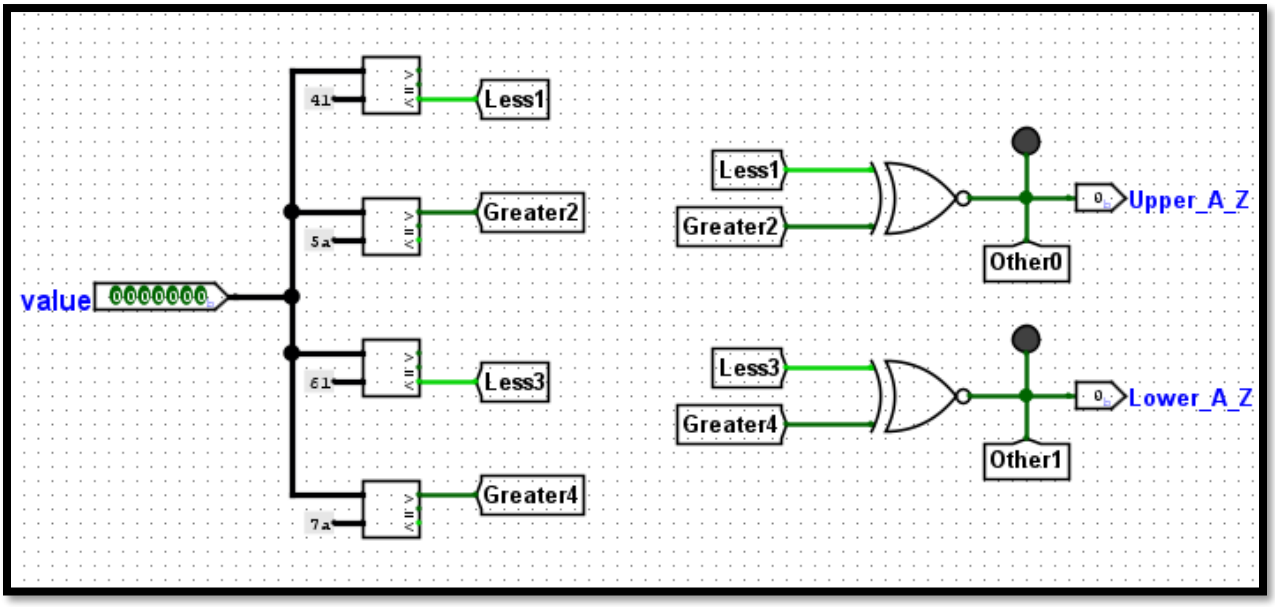


Figure 23: Final version of the range comparators for A to Z and a to z.

Most of the work here was done by observation. I started observing the behavior of a comparators. I knew I wanted a behavior like a<x<b in mathematics. So, I started by probing what designated if I was inside or outside of a range. After simulating variations of my idea, I realized that outside and inside are in a way the same destination. In other words, if the number (x) is within the range I want, it will always result in two high signals. If a low signal were to ever occur in either range, it means its outside its bound. Then it was just a matter of adding an XNOR as a way of simplifying the output. The XNOR was chosen since it only allows equal input signals to exit as a high output signal.

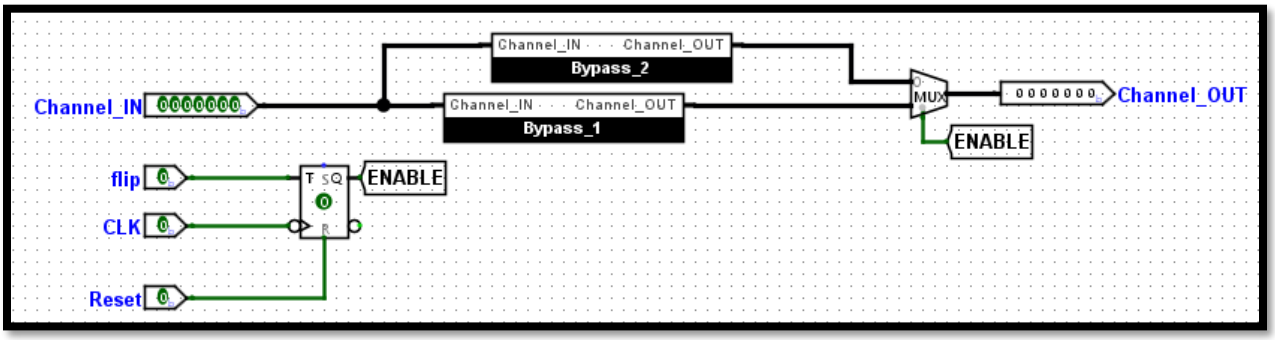


Figure 24: This is the Final Switch Circuit for the first version.

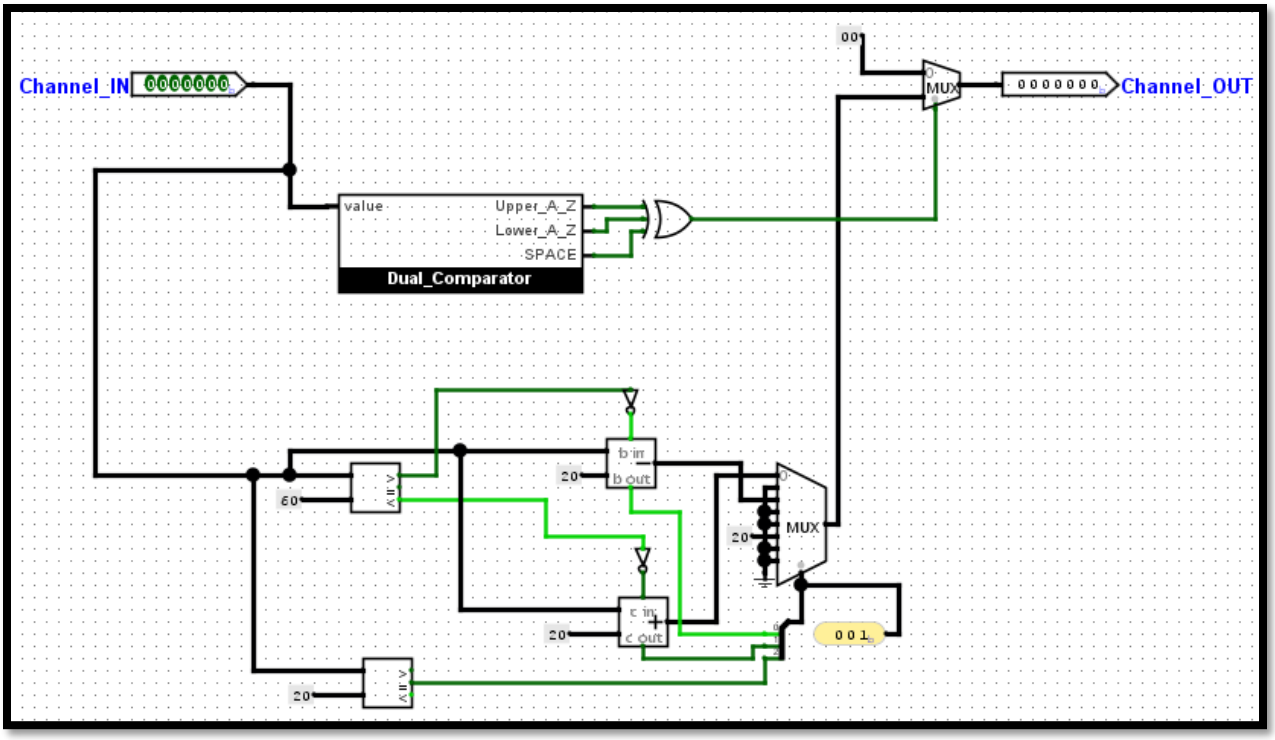


Figure 25: This is the circuit inside of the bypass_1 that gives the modification of the letters and let's only space and no sign to pass.

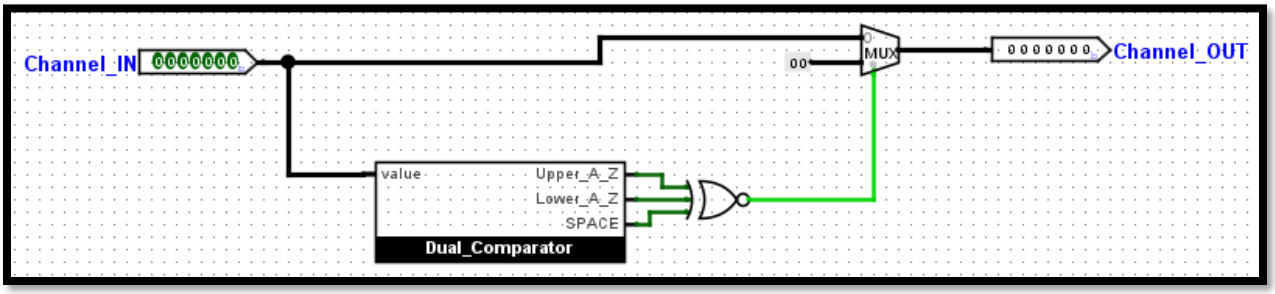


Figure 26: This is the inside of the box of bypass_2, this is the circuit that will be passing all the values of the entered except the invalid signs such as a number or a enter key etc.

First version of the switch circuit uses 2 bypasses to create a null (00 in 7-bit hex effect). To determine what character is changed to null, we use a custom comparator that determines if the input is within a specified range. If it happens to be outside of the range, it gets turned to null. The only exception for this rule is the value for space while the allowed values are A-Z and a-z. Bypass 1 contains the mechanism to switch the case of the letter by adding or subtracting 32 (base 10) to the ASCII value of the input. Bypass 2 acts as a filter where the null substitution occurs. We used a T-Type flip flop to control the mux enable since it allowed us to keep the enable on until another sequence is detected to turn off the switch circuit.

Design Validation and Circuit Verification

Circuit Version 1:

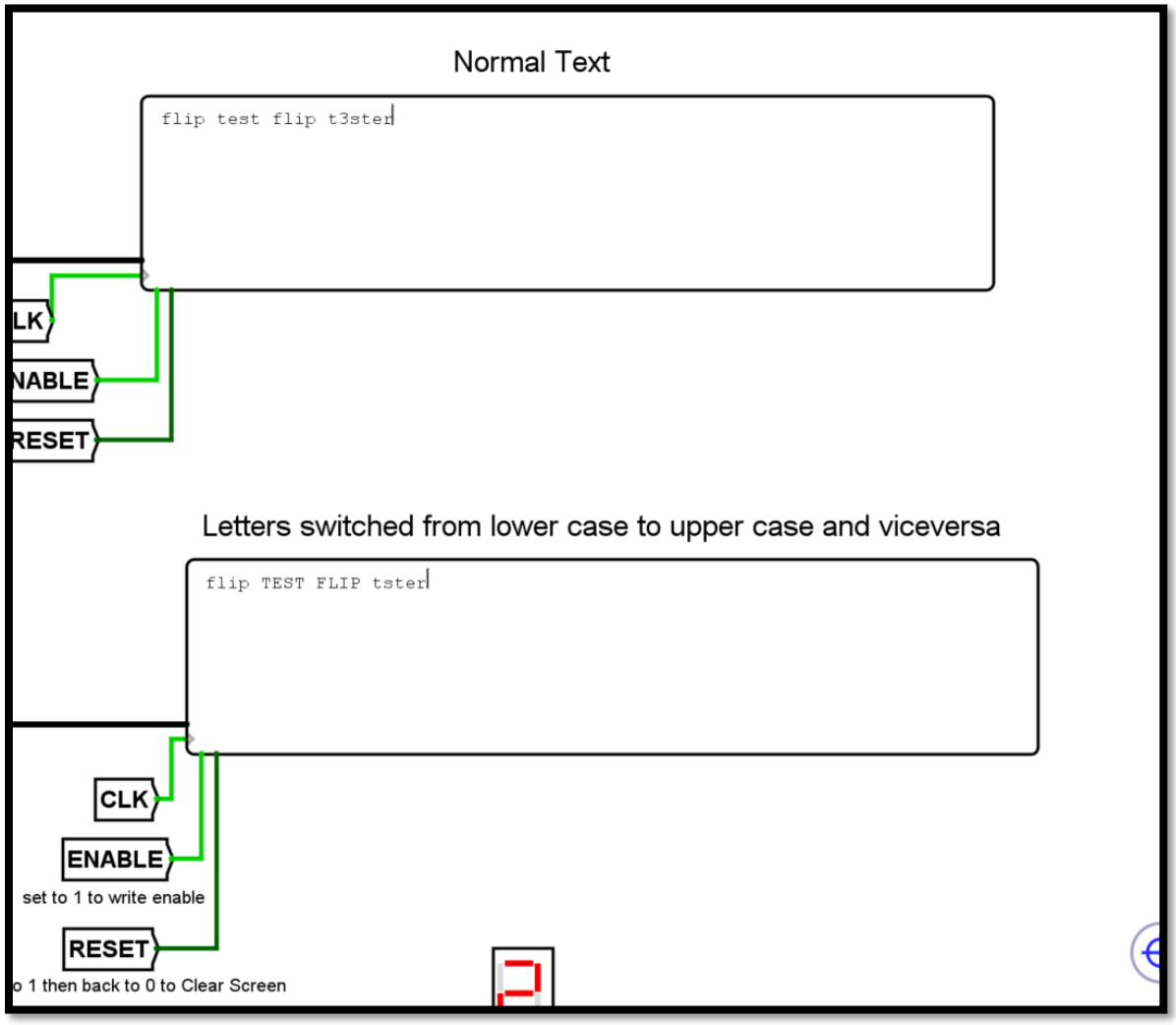


Figure 27: This is the output of the first version of the circuit, the number 3 from the word tester has been replaced with a null, that means it will not move the cursor instead will be eliminated the value.

Circuit Version 2:

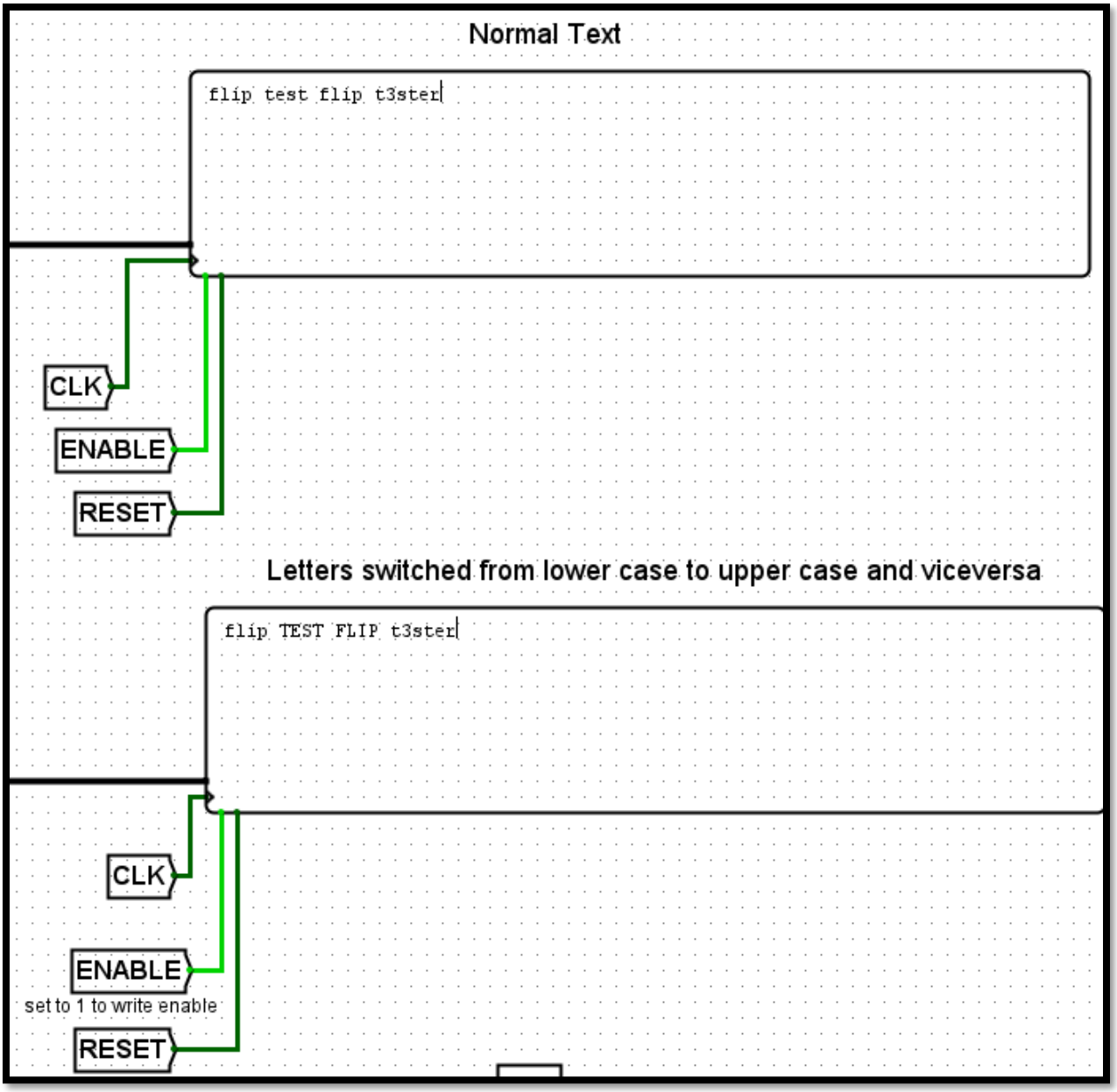


Figure 28: This is the output of the second version of the circuit and as well the word tester that contains a number 3 as a e has not been replace instead it just passed through, that's the difference of this circuit of the first version.

Switch Circuit Version 1:

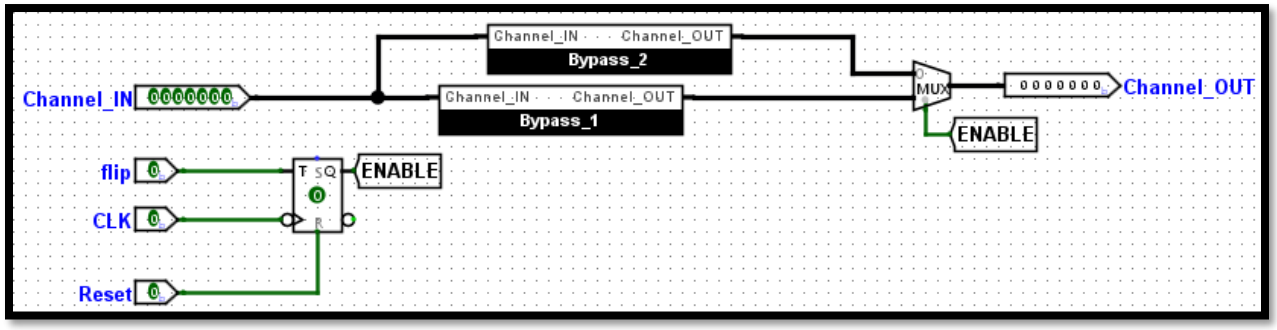


Figure 29: This is the Final Switch Circuit for the first version.

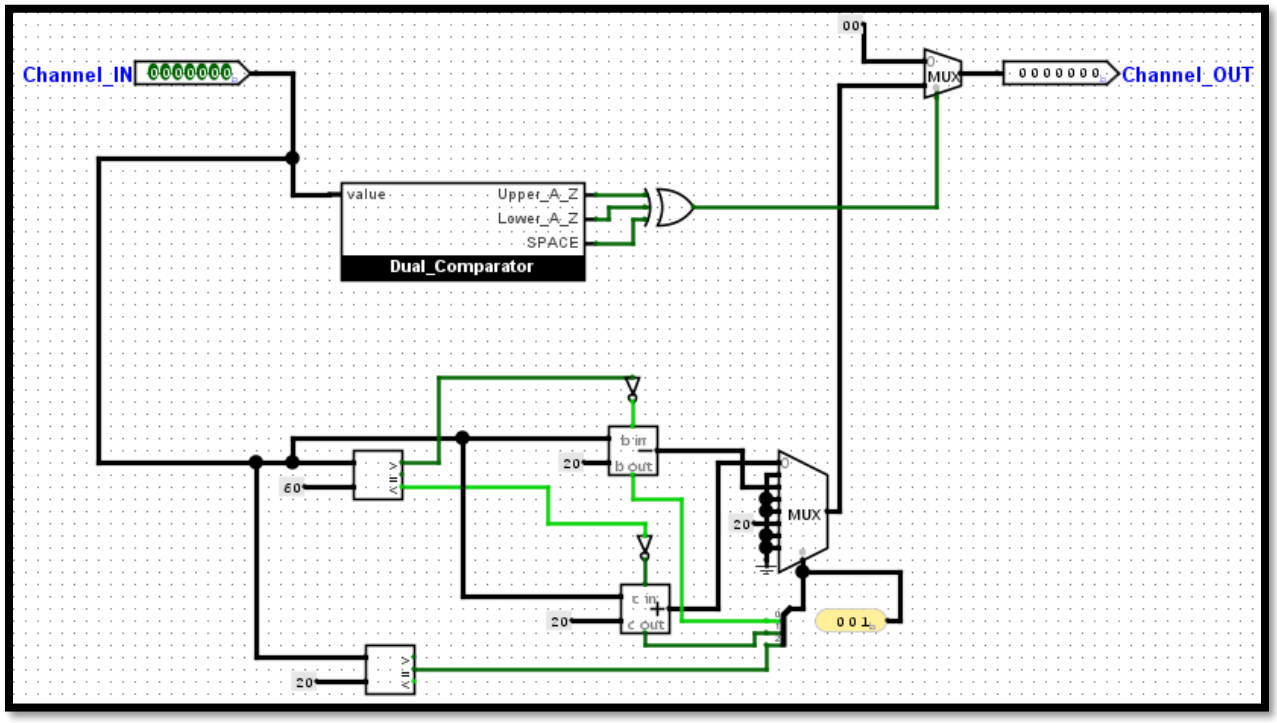


Figure 30: This is the circuit inside of the bypass_1 that gives the modification of the letters and let's only space and no sign to pass.

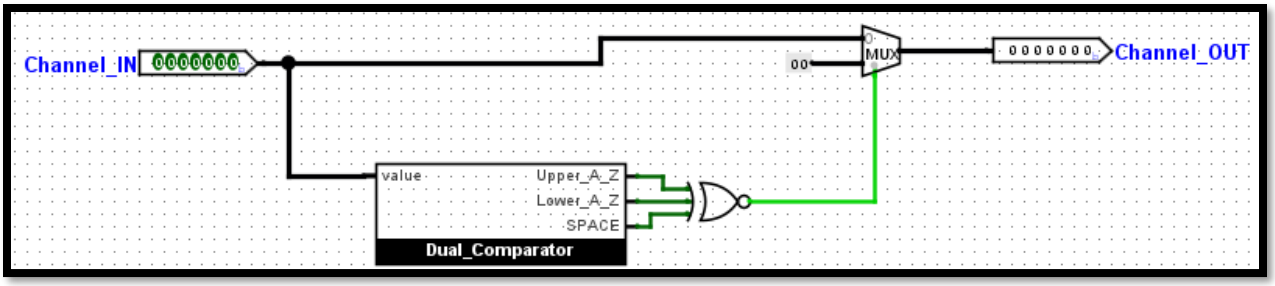


Figure 31: This is the inside of the box of bypass_2, this is the circuit that will be passing all the values of the entered except the invalid signs such as a number or a enter key etc.

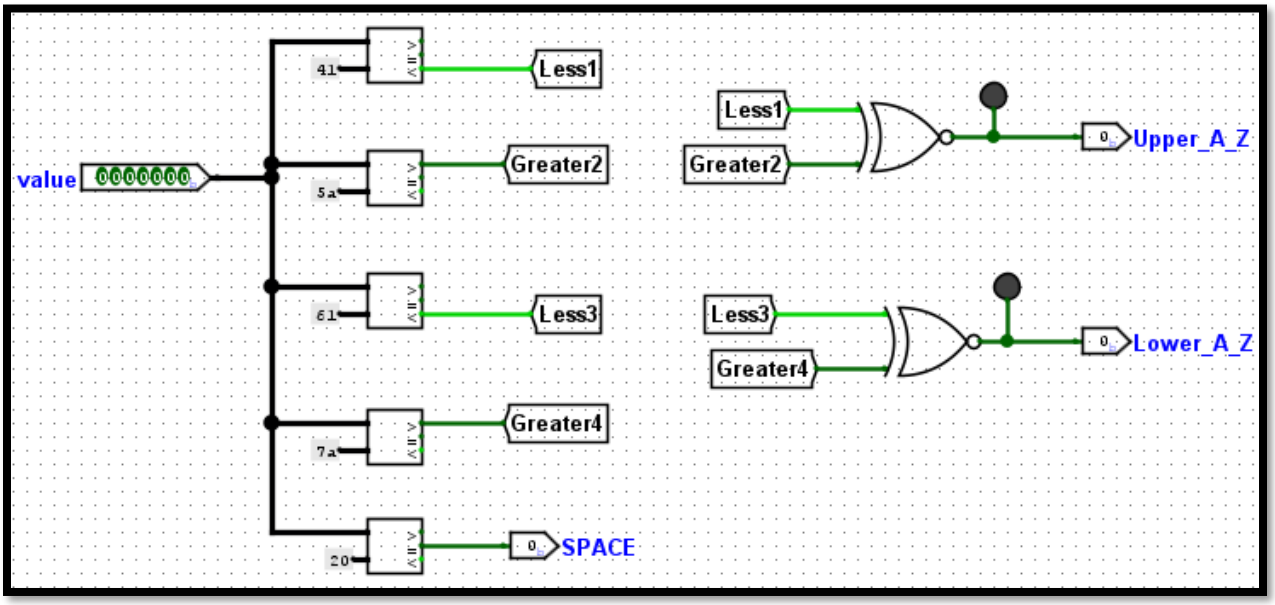


Figure 32: Final version of the range comparators for A to Z and a to z, using the comparators and the EXNOR gate.

Switch Circuit Version 2:

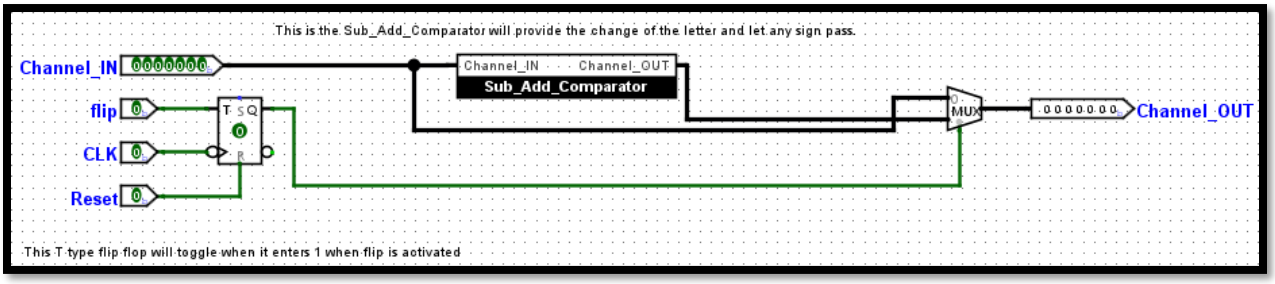


Figure 33: This is the inside circuit of the switch circuit, it contains the sub_add_comporator that will be passing the letters edited from upper to lowercase and vice versa.

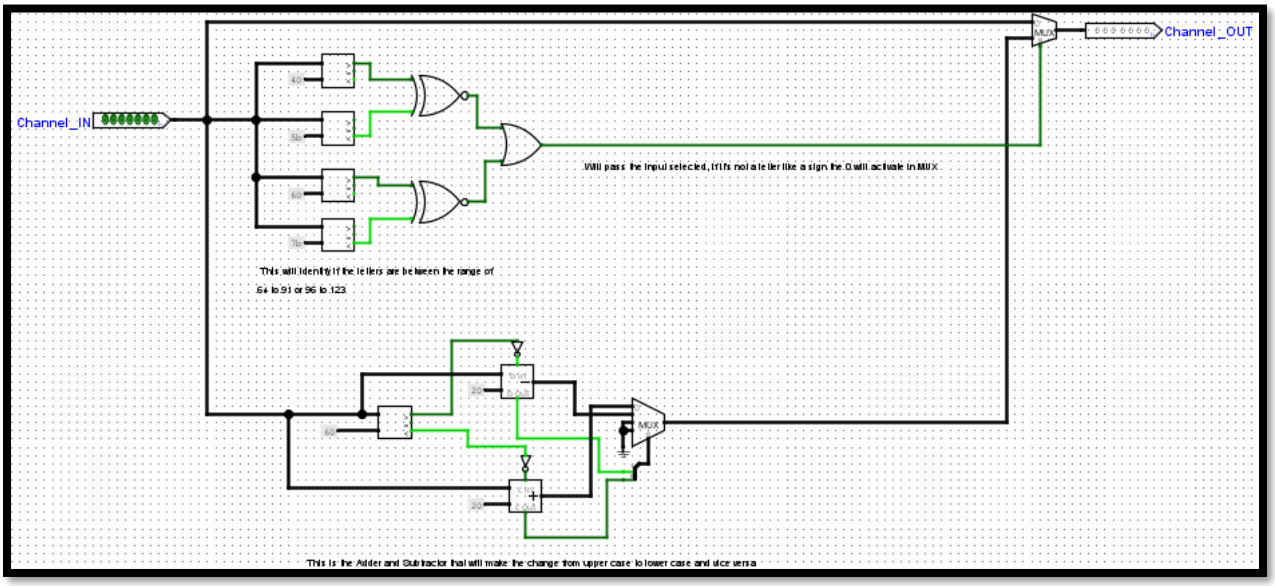


Figure 34: This is the sub_add_comporator that cointans the mux connected with the gates and comporators that will dictate the letter if is detected, once detected will be replaced with the circuit with the adder and subtractur of the 7 bits.

Summary for Circuit Version 1 & 2, and Switch Circuit 1 & 2:

The following images (Circuit Version 1 & Circuit Version 2) demonstrates all sub-systems working in unison. Though, the important part to point out is how the Switch Circuits handle their input and what they output. In Version 1, the switch circuit will make every "unknown" input into a "null" space and removes it completely. This was achieved by having two bypasses and making a custom comparator to determine if a letter is within a specified range. If the input is determined to be out of range, it gets "removed".

Two aspects from version one of the switch circuit that made it to version 2 is the way we change letter cases and the fact that we have a custom comparator. In Version 2, instead of doing a double bypass and substituting "unknown" characters "null", it simply just passes the along the

character. As in, it will go directly to the output. Which has the advantage of maintaining the structure of the input.

Flip Detector Version 1 & Version 2:

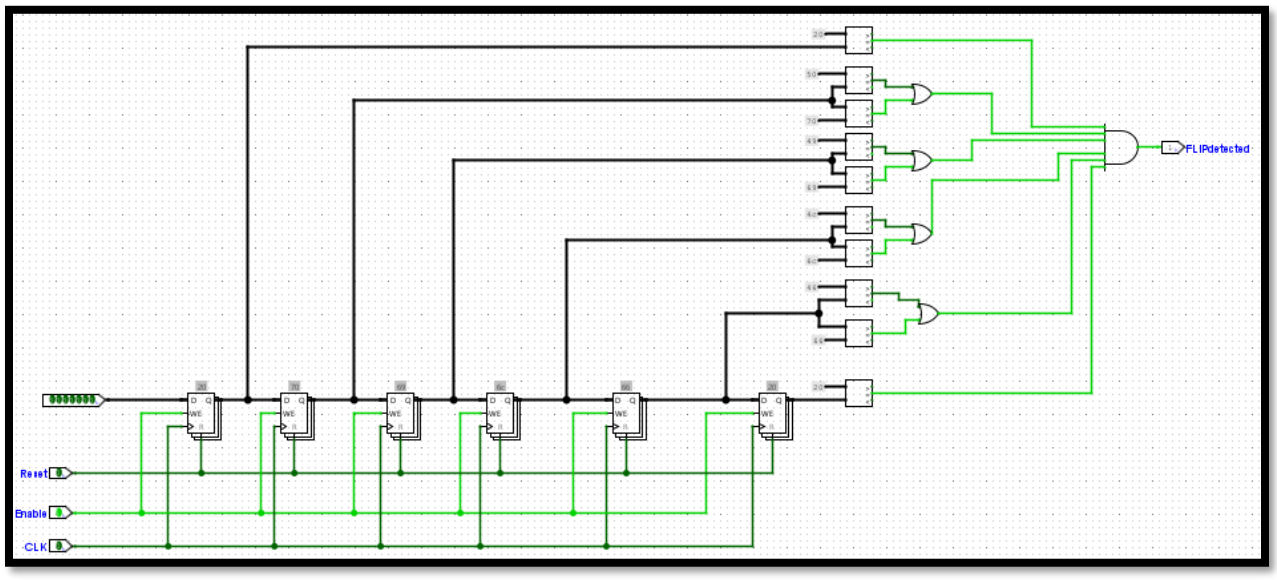


Figure 35: Flip detector for the second version.

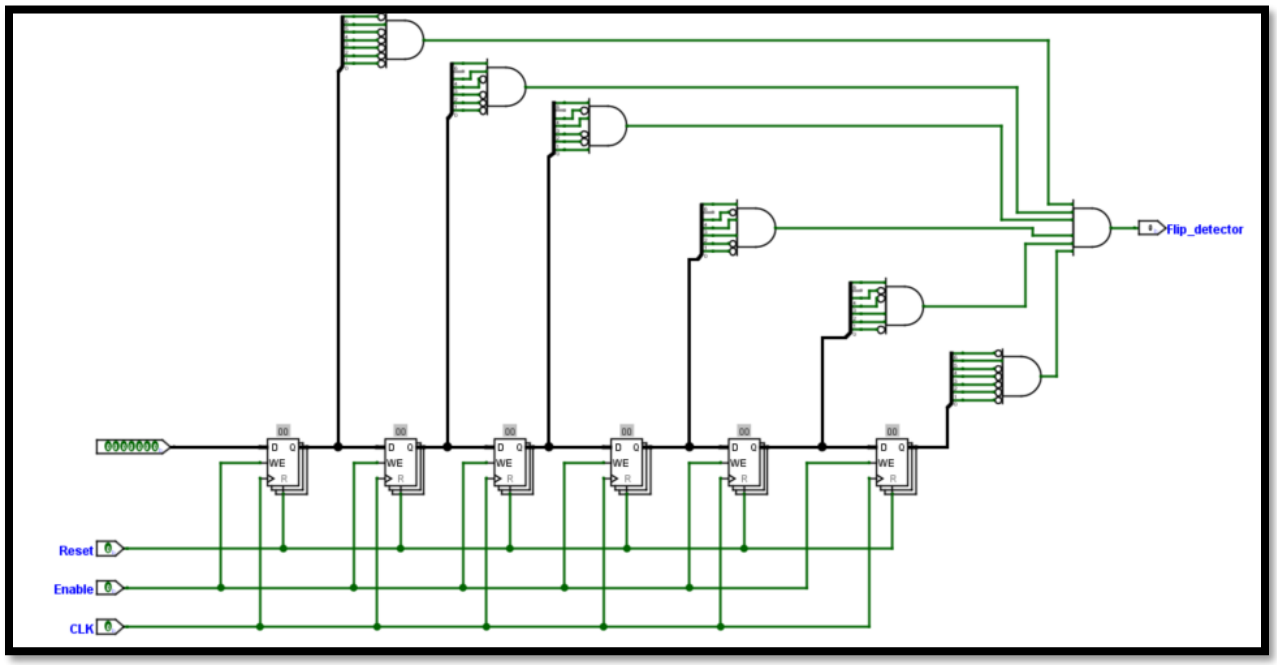


Figure 36: Flip detector of the first version.

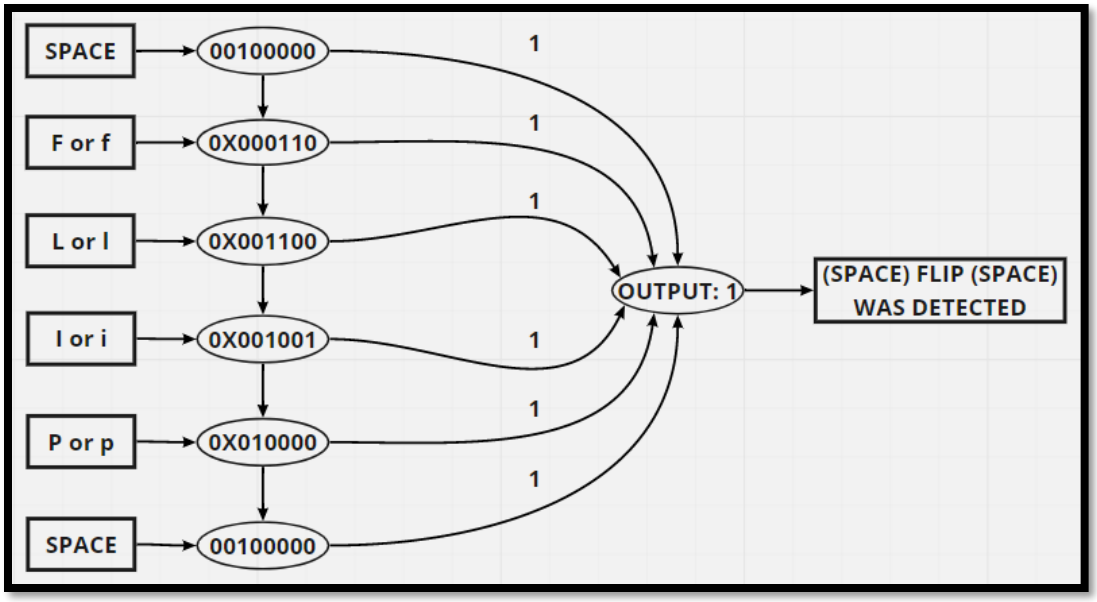


Figure 37: Diagram of the flip detector.

Summary for Flip Detector 1 & 2:

To detect the sequence "[space] F (f) L (l) I (i) P (p) [space]" we used 6 shift-register with SISO (Serial – In/ Serial- Out, shifts to the right) behavior. Since we can only send one letter at a time, the SISO behavior is important. If the sequence is detected, we will receive a high signal at the output. The diagram above demonstrates the described behavior.

Flip Counter Version 1 & Version 2:

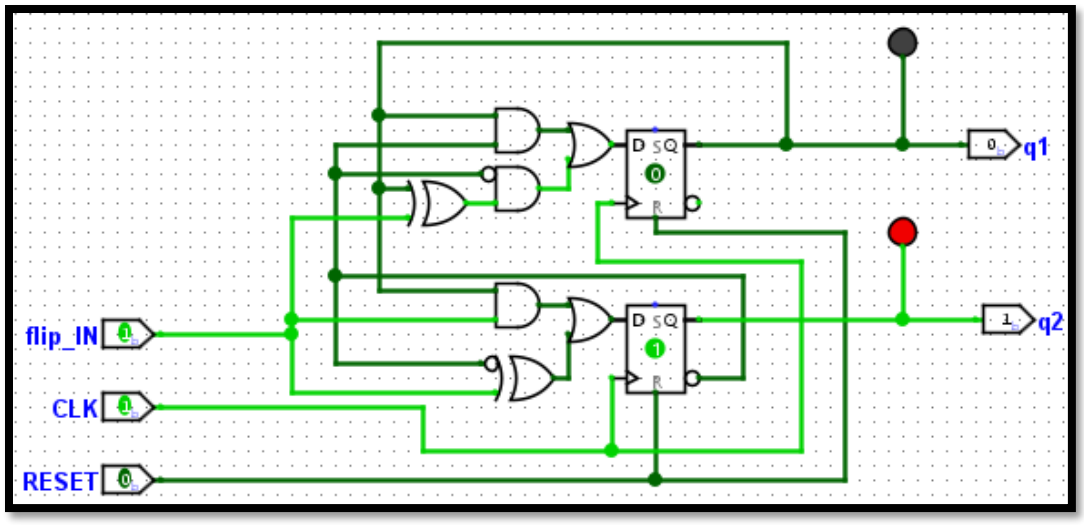


Figure 38: Flip counter giving the value of 1.

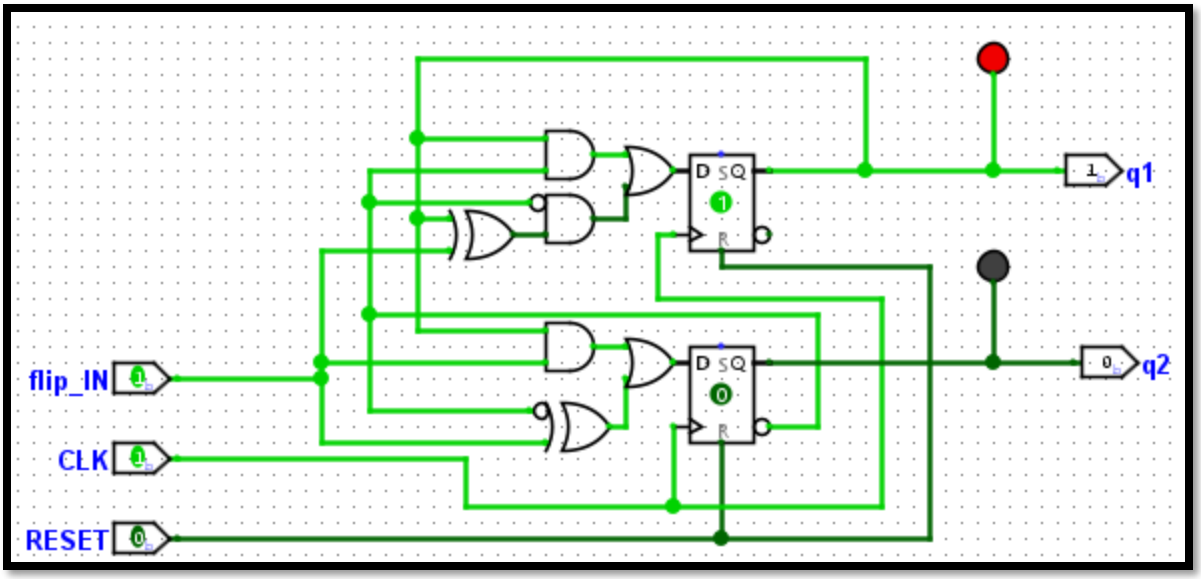


Figure 39: Flip counter giving the value of 2.

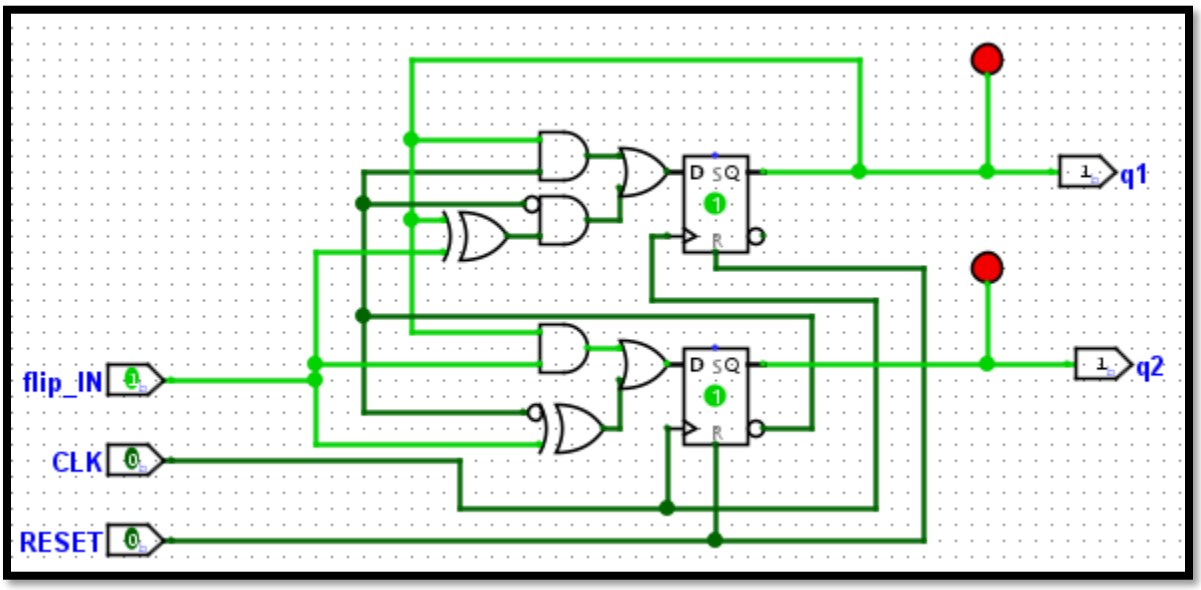


Figure 40: Flip counter giving the value of 3.

Estado Presente		Input	Flip Flop		Estado Futuro	
QA	QB	X	DA	DB	QA+	QB+
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

Figure 41: State Table of the flip counter

Summary of Flip Counter Version 1 & Version 2:

The implemented counter counts from 1 to 3 then resets to 0. The Truth table and images above demonstrate the suggested behavior, as well in figure 19 is the diagram state table.

Timing Problems

Second Version Circuit

The problem situation is when the word flip is detected, the characters that come after only the first one or second does not manage to make the change, this change refers from uppercase to lowercase or vice versa. It not only happens when the flip is detected the first time but also the second time as well, the only way it will work is by making a double space after the word flip.

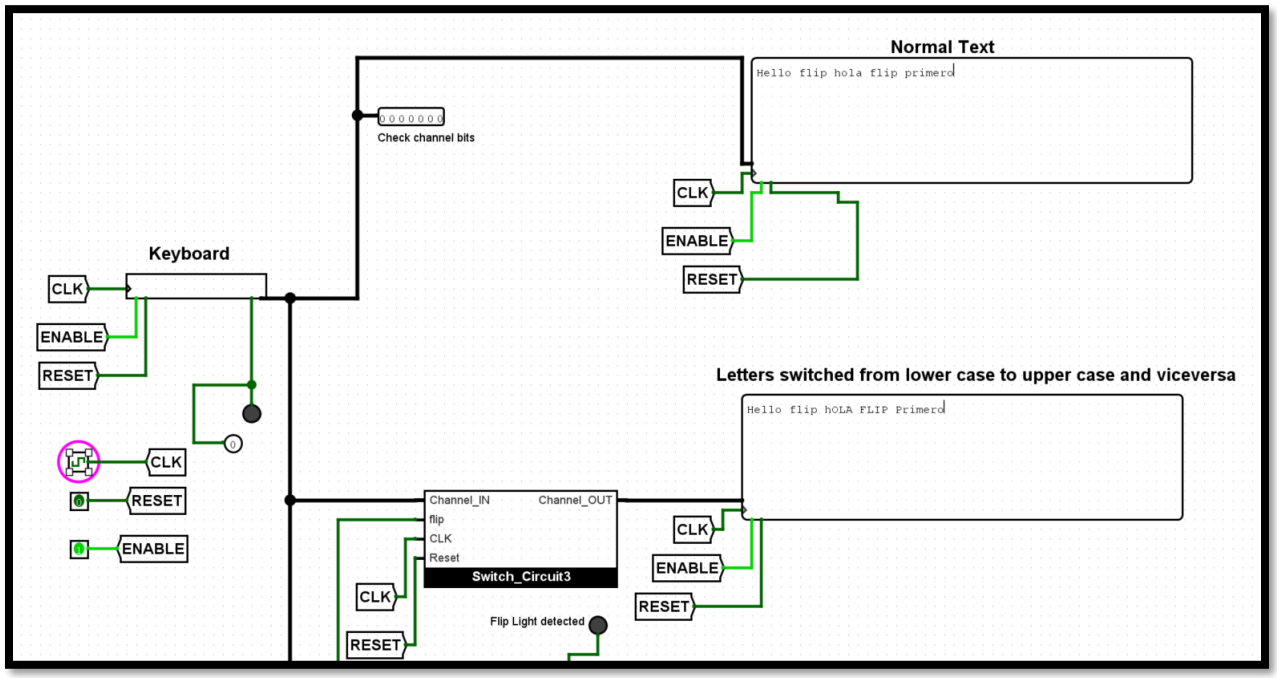


Figure 42: In the Normal Text is shown what was written in the keyboard, once it starts to tick the clock the letter in the second screen after the word flip is detected is supposed to change any word in the transit period in between of the two words flip, in which it did no occur, creating a delay in the clock cycle.

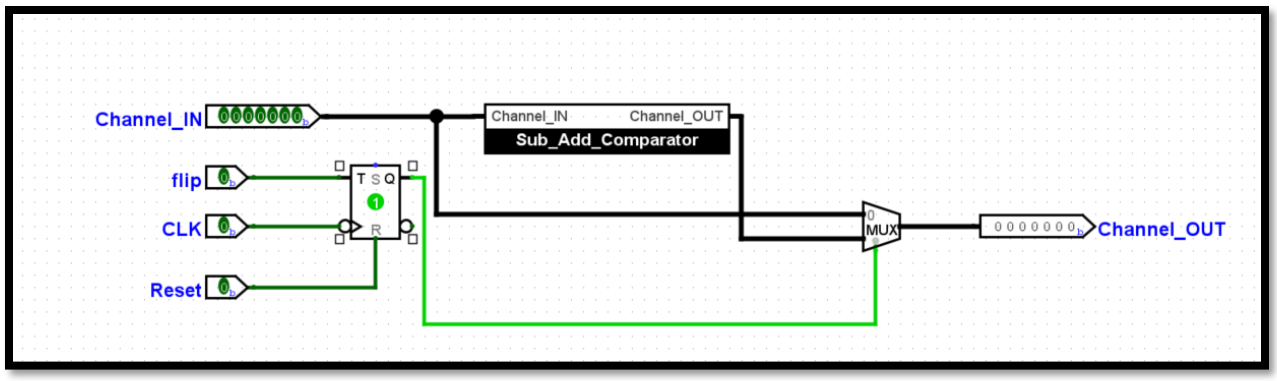


Figure 43: This is what is inside of the Switch Circuit box.

The solution of the problem was found in the T type flip flop by changing the trigger functionality of a rising edge to a falling edge. The difference of a rising edge and a falling edge trigger is that the rising edge is a low to high transition making it a positive edge, while the falling edge is a high to low transition being a negative edge.

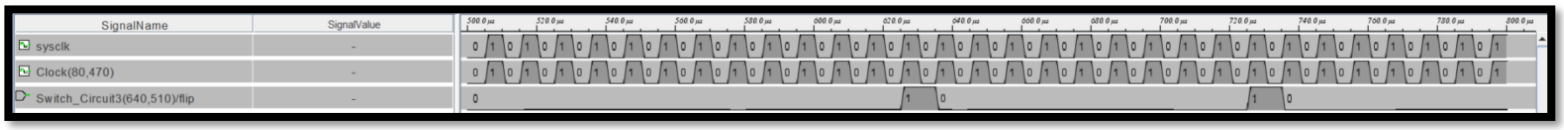


Figure 44: Timing Trace of the Switch Circuit box with the time of detecting the word flip, the channel In and clock cycle. This is when the following input was written in the keyboard "Jessica flip hola flip hello".

By seeing the timing diagram of when the flip is activated in the switch circuit box, in the rising edge there is a slight delay when the clock is 1 and the rising 1 starts later, but later the falling edge is on time without delay. Meaning that by using the T type flip flop trigger with the falling edge there won't be a delay and the circuit will work how is supposed to, the only explanation to it is that breaking the logic in parts there might be too much logic of passing the flip detection from one box to another that creates a delay on receiving the logic. This problem is called a propagation delay that is very important for sequential logic, propagation delay is the amount of time that it takes to pass a signal to the source of destination, the longer the propagation delay, the slower the clock will be able to run. The "sysclk" clock is just a simulation clock that isn't in use, while the Clock is the CLK clock in use throughout the entire circuit. This timing diagram is taken from the simulate section of LogiSim Evolution.

Alternative Designs

The functionality in the final version is the same, by means of a switch circuit, a flip detector, and a counter for 3 flips. The subsystem circuits have differences in operation and architecture. Because of these the differences, the performance of one system will be better than the other resulting in one being the main circuit and the other being the alternative circuit even if they were constructed parallel in time.

The first subsystem to be evaluated is the flip detector, which makes use of shift registers as a form to simplify and maximize the circuit's memory while using the least components possible. The part of the subsystem that changed was the method through which it detects the code word "flip", including spaces before and after it, by implementing AND logic gates in the first version and binary to hex number comparators in second version. For the first version, AND gates work with negated inputs as a SOP function, meaning that in order to detect whether the character is a match or not, the zeros of the binary ASCII code are negated while the others enter as they come. For version two, the comparators are brought into play in substitution of the AND gates in a more compact logic and wire arrangements. This version works by taking each Q output and passing them through their respective comparators which have a preset ASCII value equivalent hexadecimal number to check if the character in memory matches the one in the code word. Detection of spaces uses one value because there is only one case for spaces, but letters use two due to two possible cases. For these scenarios, an OR gate is used in case either is detected. One thing both versions have in common is that for the detector output to be one the six (6) memory output need to match the in their comparators. Another aspect taken into consideration was the delay in which the activation is made for each version alone, not as part of a final circuit whole. The results from the timing trace indicate that the subsystem from second version using the comparators has the least amount of the delay between both. The opinion is that the capacity to change the hex values for code word replacement through programming makes the second version detector the main design, making the first version the alternative design.

The second subsystem to evaluate is the switch circuit, both versions of which apply the same method of holding activation and logic to decide if the input simply passes unchanged or if it is altered as required by the flip design. The T-type flip flop was chosen because of its capability to hold memory waiting for the clock (code word detection signals in this case) along with the 2-1 MUX for its function of applying one of two paths, pass freely or pass altered. Version one uses what were labeled as Bypass_1, to flip, and Bypass_2, to pass freely. Both paths use a dual comparator subcircuit operating as identifiers of uppercase, lowercase or space. Bypass_1 takes the output of the dual comparator to pass it through one XOR in order to be used as a selector. When flip is chosen, the binary input enters the MUX with binary 32 subtracted if its upper case, added if lower case and unaltered if space to be the output. Bypass_2 takes in the dual comparator output to pass nothing (00). When the current character is one outside of the range of upper or lower case, or not a space; the character passes freely. Version two applies a simpler logic by using a 2-1 MUX that lets the characters pass freely if not enabled, or implements flip logic to letters otherwise. The switch circuit is more compact and straight to the point. The circuit used at selector 1 uses comparators to identify within the range of upper or lower case and uses the 32 subtraction or addition if true. When true, the binary input enters the MUX of that subcircuit with binary 32

subtracted if its upper case, added if lower case and unaltered if space to be the output. When not true, the special characters outside the ranges pass freely. After seen side by side, the second version design follows the requirements for letters and special characters. Meanwhile the first version covers the requirements for letters as well as the version one, but it has an extra function that make special characters not be displayed. Knowing the requirements of the project the main version must be the number 2 and the alternative is the number 1.

The third subsystem to evaluate is the counter. Both versions implement the same type circuitry due to it's reliability. The designing was done through state diagram to assure it fulfills the requirements of the project. This subsystem doesn't play a part performance differences between the two circuits.

As mentioned all throughout the report, the second version is capable of covering all the tested scenarios through the logical binary AND filters in order to target specific sequences in each of the character bits; the logically calculated state table counter design targeting the different states possible, and the specific identification and channeling of the outputs to every identification. The reason the first version was left as the alternative design lies on the underperformance compared to the second version. A noticeable characteristic that makes it the alternative design is the way it treats the special characters by not showing the on the flip screen. The second version has faster detector operation visible in the timing delay, the logic layout is simpler while getting the same work done, programmable component capabilities by the comparator use in more subsystems and less space occupancy.

Conclusions

Most classes that feature more interactive learning have a final project that seeks to encompass the knowledge acquired by the students throughout the course. These projects put students to the test, but often times they mostly emphasize the later topics of the class. Earlier topics are often relegated to be a footnote in the project or a given that students are supposed to already know and not need to prove any further. This particular final project differentiates itself greatly from other classes' projects in that regard.

The concepts, goals, and general layout for this project were given out quite early on. Even if groups wanted to begin creating designs to achieve the same goal with the given criteria, classmates would run into great obstacles since they had never tackled a project of such scale. Confounding matters, early attempts at the project were likely convoluted due to the limited knowledge at hand at the time. Sure, some classmates could go ahead and implement more elaborate concepts before their subjects were touched upon in class, but it was a bit of a risk. With these circumstances in mind, one can see how crude/rough the original prototype circuits for this project were. As time progressed and more subjects were covered in class, one can denote the simplification and refinement of the same circuits. You could say they even became much more legible as improvements were made along the way.

Everything from basic Combinational Logic and Karnaugh Maps to Sequential Logic, Comparators, and Multiplexers are seen throughout this project, serving as a compendium of the class's subjects as well as a test of student's knowledge. For instance, the initial versions for the " flip " detector were comprised of basic Combinational Logic paired with Binary Counters in order to count spaces and letters. These counters were set to send a positive signal once both counters were tripped once. The later version of the " flip " detector utilizes a much more streamlined approach with the use of shift registers. Each register was allowed to process 7 bits at a time with the idea that if a "flip "were to be entered, the characters would eventually fall into place in their respective shift register and trigger the filter attached to each register. Once all 6 characters had entered their registers, the final AND gate would trigger and send a positive signal.

As for the Switch Circuits, we can see more of the Combinational Logic but paired alongside Comparators and Multiplexers. Given the extensive use of Comparators in this subsystem, there was also heavy use of Number Systems in order to achieve the needed results. Later versions of the Switch Circuit were reduced significantly in size as the functionality could be achieved using a smaller mechanism through further analysis. The initial Flip counters featured basic Flip-Flop counters, utilizing table analysis and K maps in order to figure out the needed combinational logic circuitry for each input. Other Flip counter circuits featured greater amounts of Flip Flop components in order to achieve a two digit counter that could be connected to a Hexadecimal display.

All of this returns to how this project separates itself from the final project from other classes. Not only is this an elaborate circuit which changes capitalization of input letters at the drop of a keyword, it can also serve as a compendium of all the different subjects in class that were

seen throughout the trimester. With the improvements done to the circuit and it's subsystems, we could also see the importance of certain components as well as comprehend the purpose of design tendencies and techniques in the field and industry. The project also served as a learning experience between classmates as many classmates tackled different subsystems in their own way. Classmates could interchange ideas and demonstrate their knowledge as well as teach each other through the analysis of their fellow groupmate's designs.

Bibliography

- "What is Propagation Delay", *Nandland.com*, 2021. [Online]. Available: https://www.nandland.com/articles/propagation_delay.html. [Accessed: 05- Oct- 2021].
- "Beginner's tutorial", *Cburch.com*, 2021. [Online]. Available: http://www.cburch.com/logisim/docs/2.7/en/html/guide/tutorial/index.html. [Accessed: 23- Oct- 2021].

Functional Groups

Design Circuit Architecture	
Description of task: For this role everyone will have to design the architecture of the circuit, how it will work and what materials will be used.	
1. Gabriel	2. Michael
3. Savier	4. Juan
5. Sebastian	6. Jessica
Circuit Synthesis	
Description of task: After different versions of each part are built and working, each individual will analyze the best option for the project.	
1. Gabriel	2. Sebastian
3. Jessica	4. Savier
5. Michael	6. Juan
Simulation and Testing	
Description of task: After the project has been done it is time for testing and simulation to see if it works and it needs a change will be proceeding back to the design architecture group.	
1. Gabriel	2. Sebastian
3. Jessica	4. Savier
5. Michael	6. Juan
Documentation	
Description of task: will take notes of everything that has been taken place of the reunions and the different plans and ideas taken to account by the group	
1. Gabriel	2. Sebastian

3. Jessica	4. Juan
Software Developer	
Description of task: once the project has been finalized the main components, then is time to implement them in VHDL, a GitHub repository has been created to store all progress made of the code, once done will pass the code to the simulation and testing group.	
1. Savier	2. Michael
3. Jessica	

Team Organizational Structure

The group for this project was the largest of the groups in the class. Due to this, maintaining a coherent structure was slightly more challenging. In this group's case, the structure was a hybrid between the aforementioned Democratic and Hierarchical structures. A leader for the group was established early on, but the different subsystems and sections of the report were taken up by volunteers. Multiple members could, and often did, work on the same subsystem for the greater circuit. This led to many different designs and prototypes, which only became more numerous as more subjects were introduced and allowed for improvements to the subsystems.

Team Design Rules

1. Reunions will be on the weekdays, Saturdays at 6:00 – 8:00 p.m. and 4:30-8:30 p.m.
2. If finished everything that is supposed to be placed for the first reunion and second reunion on one day, the next reunion can have reduced hours.
3. Each member must come in time for the reunion.
4. If not coming the reunion must meet the requirements met for next meeting.
5. If any doubts after reunion days, can contact the leader through Blackboard, email, and group chat.
6. Treat each other with respect.
7. Help one another with difficult or time-consuming deliverables.
8. Play an equal role in the team by contributing equally to every task.
9. The reunions will be in Blackboard, if Blackboard has difficulties of streaming the content will move automatically to discord server created for the team.
10. The leader will give a briefing of what will be done and what will be archived during the weekend.
11. Members that don't provide their fair share of work, they would be given warnings up to the point of having their name wiped from the project if need be.
12. Any information or circuit made for this project is prohibited to pass to others that are not in this group project.

Figure 45: Evidence of the first announcement of reunions dates placed on the group chat.

Project Management

Saturdays from 6:00 – 8:00 P.M.	Sundays from 4:30 – 8:30 P.M.
September 18, 2021 assisted: • Gabriel • Jessica • Michael • Savier • Juan • Sebastian	September 19, 2021 assisted: • Gabriel • Jessica • Michael • Savier • Sebastian
Completion: Setting up the group, description of what will be done in the project and dividing report material and groups. As well starting to plan the first version of the Switch Circuit, Flip Detector and Flip Counter. **Time reunion:** 2 hours	**Completion:** Continuation of the past reunion and demonstrating the work that was done of each component made for the circuit, analyzing strategies of how to connect all of them. Already having Switch Circuit done, but the flip detector and flip counter where a 50% done. **Time reunion:** 4 hours
September 25, 2021 assisted: • Gabriel • Jessica • Michael • Savier • Sebastian	September 26, 2021 assisted: • Gabriel • Jessica • Michael • Savier • Sebastian
Completion: Presented the finished version one of the past reunions, implement testing and simulation for each and then created the first version of the project. Started designing a second version. **Time Reunion:** 2 hours	**Completion:** No reunion, it was decided to continue the progress of the second version. **Time Reunion**: 0 hours
October 2, 2021 assisted: • Gabriel • Jessica • Michael • Savier • Sebastian	October 3, 2021 assisted: • Gabriel • Jessica • Michael • Savier • Sebastian • Juan

Completion: Simulated and Tested the second version of the circuit components, discussed doubts of each part of the report, discussed for a third version of the circuit. **Time Reunion:** 2 hours	**Completion:** Shown progress of the third version of the circuit components, analysis of the design, testing and simulation was done. **Time Reunion:** 4 hours
October 9, 2021 assisted:	October 10, 2021 assisted:
• Gabriel • Jessica • Savier • Sebastian • Michael	• Jessica • **Gabriel** • **Michael** • **Juan**
Completion: Simulated and Tested the third version and the past versions into account after having the professor's reunion Monday at 5:30 p.m., deciding to change the strategy and changing to be only two versions and fixing the other boxes details, analyzed each design and circuit synthesis. As well review the document changes and what will be placed on report. Fixed coding issue of VHDL. **Time Reunion:** 2 hours	**Completion: Checked the time delay of each subsystem, passed a google forum to vote for the best subsystem for each version of the circuit, build the two versions of the circuit to work, simulation and testing was also made. Discussed for a possible a new upgrade of switch circuit to be made, implanting the idea in logiSim Evolution and verified the delay of both and talked of what will be the best version. (Ended up staying with the same switch circuit)** **Time Reunion: 4 hours**
October 16,2021	**October 17,2021**
• **Sebastian** • **Michael** • **Savier** • **Gabriel** • **Juan**	**No reunion**
Completion: Completed most de report, review some designs and VHDL code for Switch Circuit. **Time Reunion: 2 hours**	**Completion:** Dedicated to everyone to complete each part of the report. **Time Reunion:** 0 hours
October 23,2021	**October 24,2021**
• **Sebastian** • **Michael** • **Savier** • **Juan**	• **Sebastian** • **Michael** • **Gabriel** • **Juan**

• **Gabriel** • **Jessica**	• **Savier** • **Jessica**
Completion: Fixed the report, version 1, and documented one more section of the report. **Time Reunion: 2 hours**	**Completion:** Fixed every small detail needed in the report and went through the report to discuss any doubts. The VHDL code will be send in a individual file that contains all code made, since it does not work in LogiSim Evolution. **Time Reunion: 4 hours**

Note: "*Reunions will be twice a week, the first reunion consists of 2 hours, while the second one is 4 hours, having 6 hours total to discuss and implement the project as a team. The group divided in two subgroups one contains the switch circuit and full circuit schematic, meanwhile the other group will be creating the flip counter and flip detector. Any member that couldn't finish the progress to be made for the reunion time will continue during the week and will be informed by the leader what happened on the reunion."*