INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Assignment N^o2

Electrónica 3 - 2018

Group 2:
Díaz Ian Cruz
Lin Benjamín
Müller Malena
Oh Victor

Professors:
Dewald Kevin
Wundes Pablo Enrique

November 15, 2018

1 Exercise 1: Tank Simulation

1.1 Moore's Finite State Machine

Moore's Finite State Machine (FSM) follows a model where the next state of the machine is determined by the current inputs and its current state, while the output is determined by the current state of the machine, following the designed combinational logic.

1.1.1 Design

Given the specifications required of the pump controller, the functionality of the fsm was represented on Figure 1.1 and on Table 1.1, where "LnA" stands for "Last not Activated".

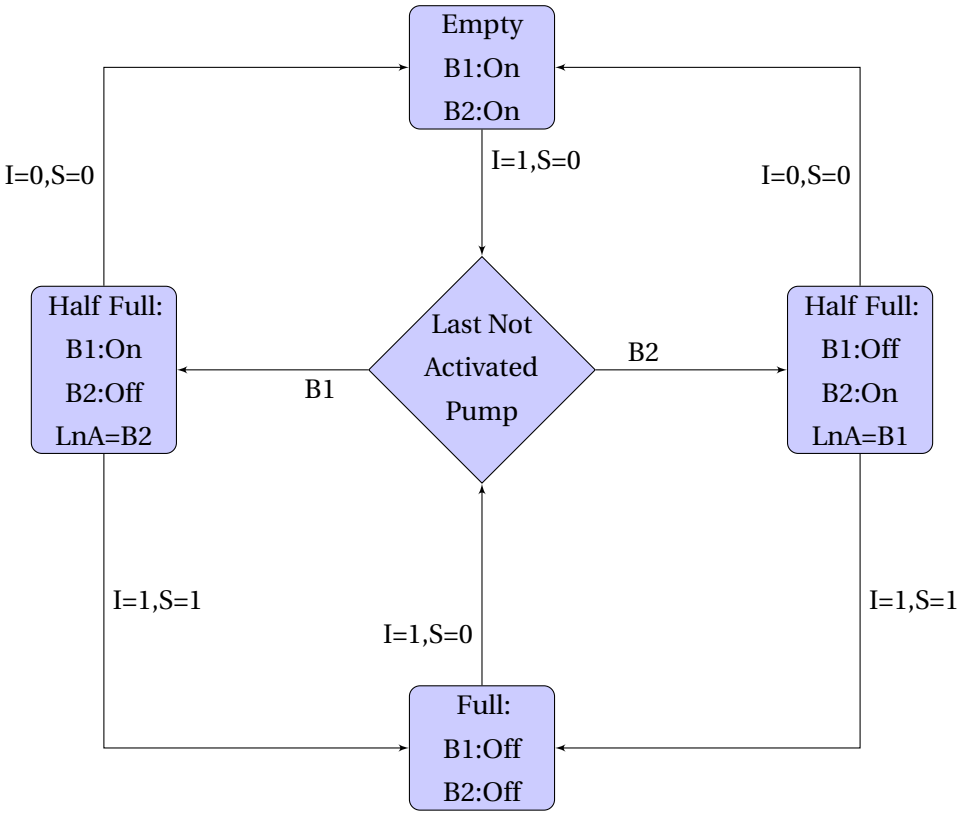

Figure 1.1: Flow Diagram for Moore's Machine

Current State	Next State				Output	
	S I 0 0	S I 0 1	S I 1 0	S I 1 1	B1	B2
Empty (0 0)	Empty	Half Full	Error	Full	1	1
Half Full (0 1)	Empty	Half Full	Error	Full	~LnA	LnA
Error (1 0)	Empty	Half Full	Error	Full	0	0
Full (1 1)	Empty	Half Full	Error	Full	0	0

Table 1.1: State Transition Table

It can be observed that an error state was added. This was to cover every possible case of inputs. It was

decided that in case a "strange" input was received (Superior sensor activated while the Inferior sensor is not), both pumps should stop working, as that seemed to be the safer decision, as this seemed to simply be a replenishment system and an empty tank would alert the user of the error.
After deducing the combinational logic behind the next state assignment and the corresponding outputs for each state, and taking into consideration the available components in the laboratory, the circuit in Figure 1.2 was obtained.

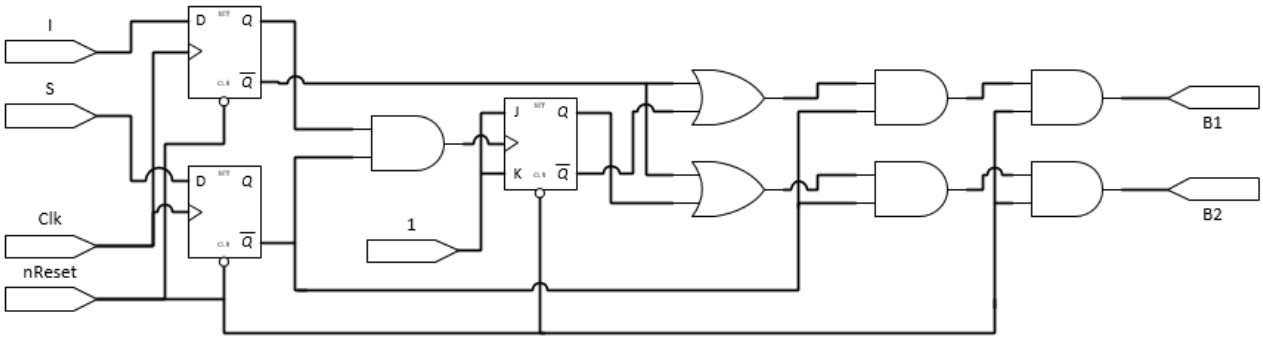

Figure 1.2: Resulting circuit logic diagram

1.1.2 Simulation

Before physically implementing the FSM, first it was simulated in Verilog and ran through a testbench, where the waveform in Figure 1.3 was obtained with gtkwave.

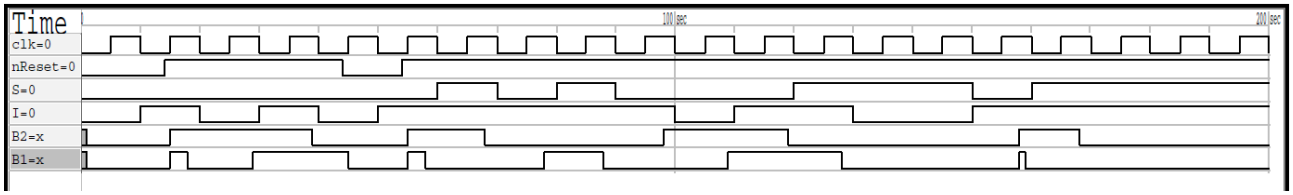

Figure 1.3: Simulation results from gtkwave

It needs to be mentioned that in the design, a method of resetting the machine was included in case it was needed. This reset case will set the LnA variable to 0 (which arms B1 for use) and deactivate both pumps until it is no longer set.

1.1.3 Physical Implementation

After confirming the correct behaviour of the device, it was implemented on a Printed Circuit Board with the schematic on Figure 1.4.

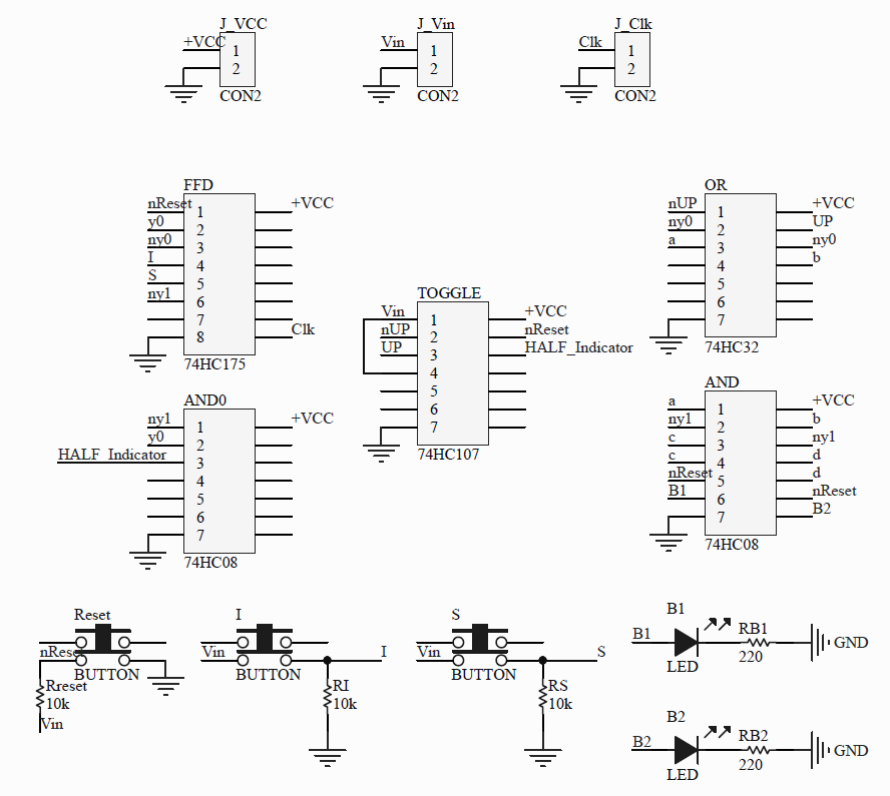

Figure 1.4: Device Schematic

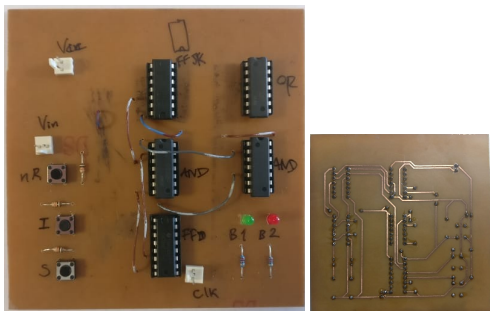

Figure 1.5: Final Physical device (front/back)

After it was fully implemented and tested, several measurements were taken to figure out its characteristics. In Figures 1.6 and 1.7 both the functionality and the delay between the output and the reset input can be seen and were measured to be 6.6ns.

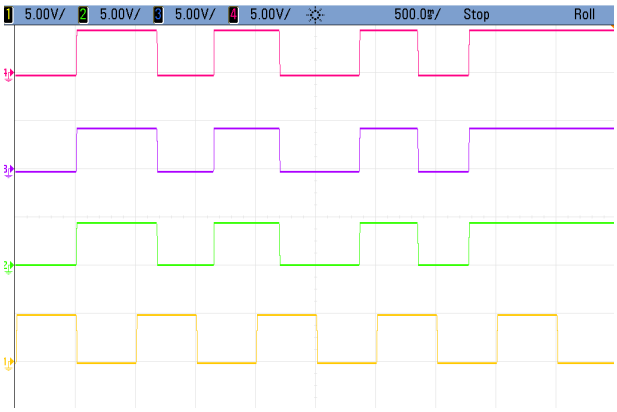

Figure 1.6: Reset (pink) functionality testing

Figure 1.7: Reset (pink) delay measurements with B1(green) and B2(blue)

The delay between the clock's positive edge and the output signals was also measured to be around 30ns in Figure 1.8.

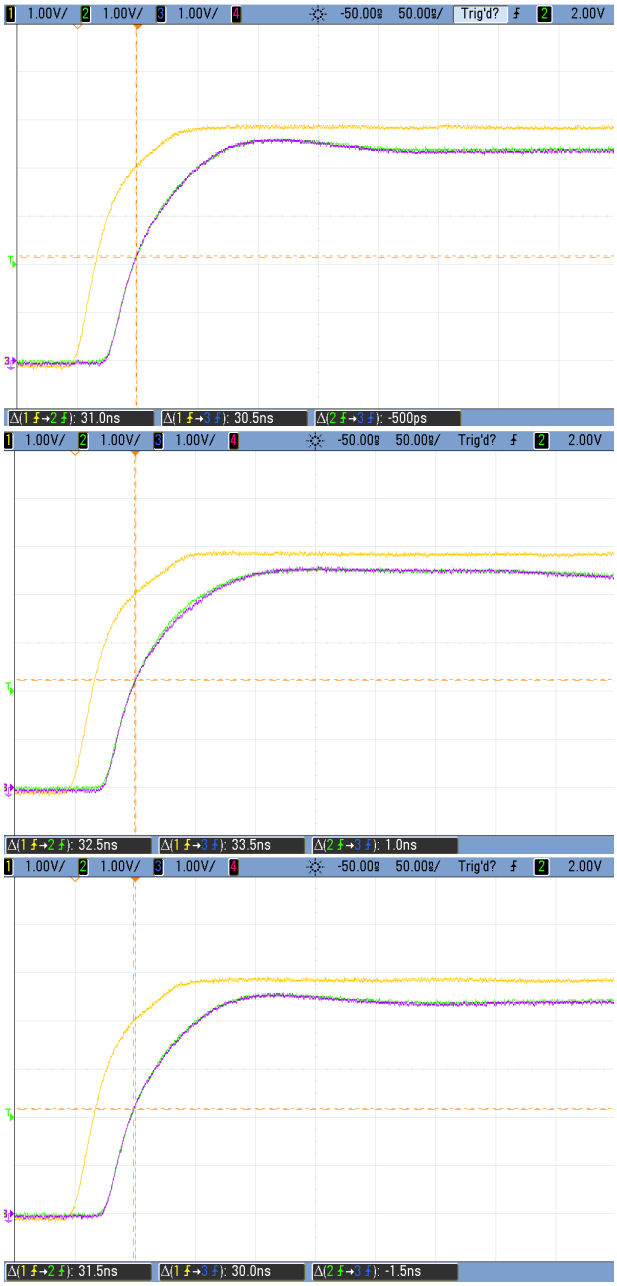

Figure 1.8: I/O Delay measurements at 1Hz, 1 kHz and 100 kHz respectively

Lastly, some general testing for the correct behaviour of the machine was tested, which was the one expected and designed, as well as consistent to the one obtained in the Verilog simulation. From Figures 1.9 and 1.10 it can be seen that the maximum output voltage is around 4.3V when fed with a 5V source and a minimum of near 0V, taking into account the noise.

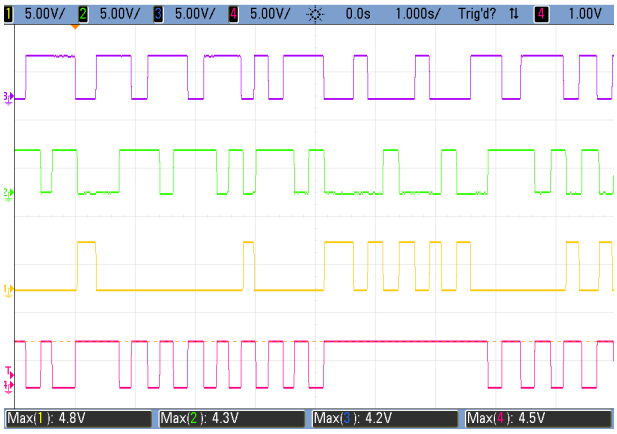

Figure 1.9: High I/O voltage measurements

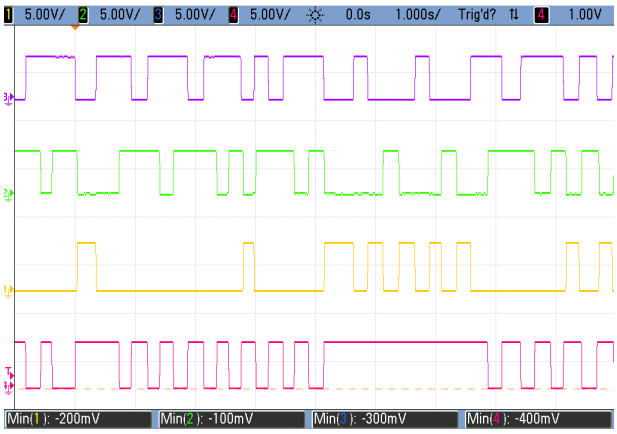

Figure 1.10: Low I/O voltage measurements

1.1.4 Conclusions

From all the measurements taken from the physical device and the simulations ran in Verilog, it can be concluded that, given the case use this device would have, it is working as expected, with low delays between input and output signals and accounting for all possible input cases, with a safety precausion of disabling all pumps from activating if there is an error in the system or it is wished to be shut down.

1.2 Mealy State Machine

In the Mealy state machine, the output value not only depends on the state we are but also depends on the input values. This is to say the output could be represented as a function like $Z = f(X_1.....X_n, Q_1....Q_n)$ where Z: output, Q: State and X:Input event as could be visualize:

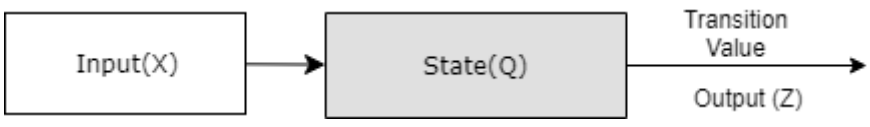

Figure 1.11: Mealy sate machine simple representation

In this exercise, two sensors I and S function as input event to the state machine. Analyzing the possible states and event we obtain the following Mealy machine diagram:

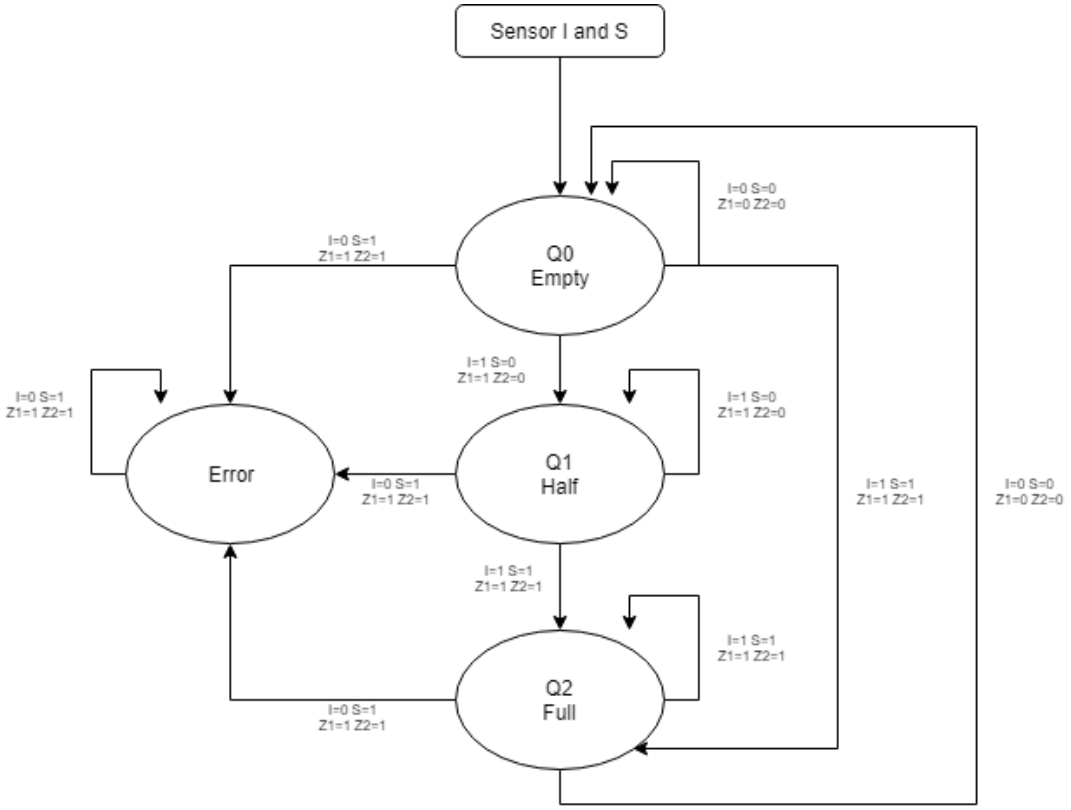

Figure 1.12: Exercise 1: Mealy sate machine flow chart

Which is represented as follow:

State(Q)			**Input(X)**							
			I=0 S=0		**I=0 S=1**		**I=1 S=0**		**I=1 S=1**	
Representation	**Q2**	**Q1**	**Q2**	**Q1**	**Q2**	**Q1**	**Q2**	**Q1**	**Q2**	**Q1**
Empty	0	0	0	0	0	1	1	0	1	1
Error	0	1	0	0	0	1	1	0	1	1
Half	1	0	0	0	0	1	1	0	1	1
Full	1	1	0	0	0	1	1	0	1	1
Output(Z)	**Z1**	**Z2**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

Table 1.2: Exercise 1: Mealy sate machine

So the logic circuit would be:

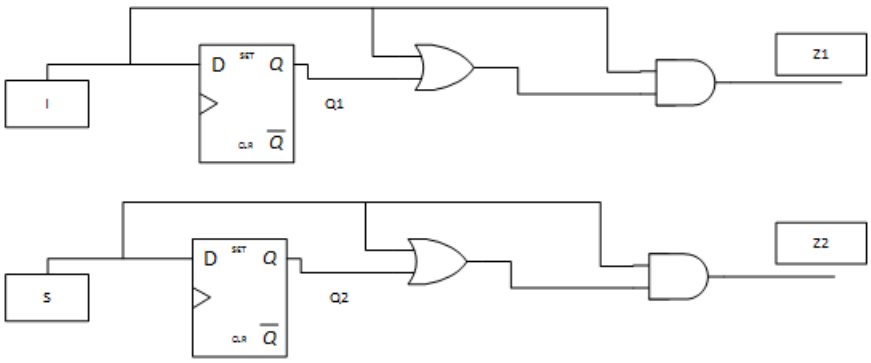

Figure 1.13: Exercise 1: Mealy logic circuit

We can notice that the input event and the output event are the same which could make the $Z_N = X_N$ with N: the output or input number, but as mention be in the Mealy state machine the output $Z = f(X_1.....X_n, Q_1....Q_n)$, so we considered essential the use of sate in the circuit is dependent with the state and the input to have a clear view of being a Mealy state machine.

1.2.1 Simulation

For the simulation of this stage machine, as the pump B1 and B2 alternate their function when $I = 1 y S = 0$ and for the activation of the pump that depends on output voltage is needed the following circuit is added:

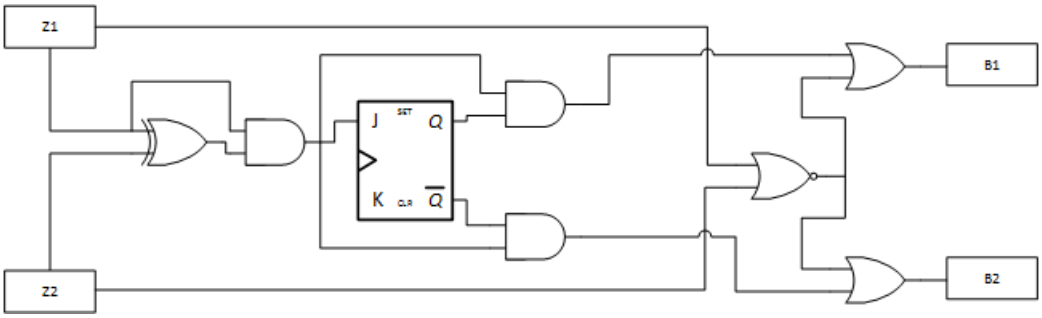

Figure 1.14: Exercise 1: Mealy additional logic circuit for the simulation

Simulation the complete circuit in Verilog and testing the possibles values of input in Gtkwave the result was:

Figure 1.15: Exercise 1: Simulation results

2 Excercise 2: State Machines to Detect the sequence 1-1-0-1

In this exercise, the detection of the sequence 1-1-0-1 inside a longer sequence of bits, is done with a Moore's state machine and with a Mealy's state machine. The main difference between these two state machines is that in Moore's one, the output only depends on the present state of the machine, while in Mealy's one, the output depends on the present state as well as on the input. This causes a time displacement between the output of both state machines. Moore's output "answers" to the input one clock later after having reached the new state, while Mealy's output "answers" immediatly during the same clock that the input arrives. This is because Mealy's reacts not only according to the present state, but also depending on the input's value.
For the implementation of both state machines, five states are needed and they are represented with the letters from A to E:
- A (000): "IDLE", the state in which the first digit of the sequence has not yet been detected.
- B (001): The first digit of the sequence, "1", has arrived.
- C (010): The second digit of the sequence, "1", has arrived.
- D (011): The third digit of the sequence, "0", has arrived.
- E (100): The last digit of the sequence, "1", has arrived.

The states are digitally represented with three bits, as they allow to get five different combinations, so that the name of each state is represented with a binary number. Such representation is the one given between parenthese, after the name of each state. It is important to mention that no matter which is the present state, whenever a "0" arrives from reset, the state changes to state A (IDLE).

2.1 Moore's Type State Machine

In figure 2.1 it is represented the sequence detection with the Moore's state machine.

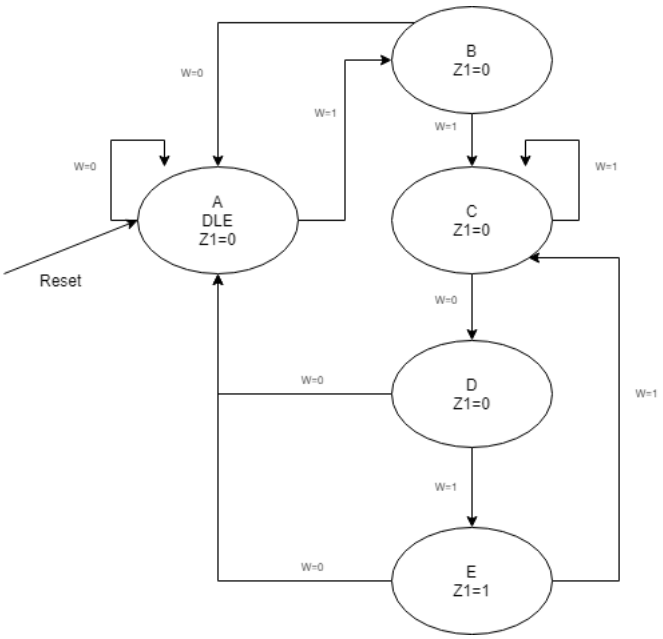

Figure 2.1: Moore's State Machine, for detecting the sequence 1-1-0-1.

From above's diagram (the one in figure 2.1), the following table 2.1 is obtained.

Present State	Next State depending on w		Output z
	w=0	w=1	
A (000)	A (000)	B (001)	0
B (001)	A (000)	C(010)	0
C (010)	D (011)	C (010)	0
D (011)	A (000)	E (100	0
E (100)	A (000)	C (010)	1

Table 2.1: Moore's Next States and Outputs.

The way the reset influences is not shown in the tables because it works the same way for it ocurring at any of the states. If reset=1, nothing happens. But if it equals 0, the reset occurs. It works as a negative reset. Apart from that, the present state is represented as a 3 bit vector y: (y3,y2,y1) while the next state is represented as Y: (Y3,Y2,Y1). From the previous table 2.1 the following Karnaugh maps are used to get to simple logic expressions. The first Karnaugh map is to get Y1, the second one for Y2, the third one for Y3 and the last one for the output z. As three bits are being used to represent the states, there are combinations of these bits that act as don't cares (x) in the Karnaugh maps. This is because we have only five states, and three bits give more than five combinations. This helps simplifying even more the final expressions.

w y3 \ y2 y1	00	01	11	10
00	0	0	0	1
01	0	X	X	X
11	0	X	X	X
10	1	0	0	0

w y3 \ y2 y1	00	01	11	10
00	0	0	0	1
01	0	X	X	X
11	1	X	X	X
10	0	1	0	1

w y3 \ y2 y1	00	01	11	10
00	0	0	0	0
01	0	X	X	X
11	0	X	X	X
10	0	0	1	0

y3 \ y2 y1	00	01	11	10
0	0	0	0	0
1	1	X	X	X

From the Karnaugh maps, we get to the following expressions:

$$Y1 = \overline{y1}(\overline{y2y3}w + y2\overline{w})$$

$$Y2 = y2\overline{y1} + w(y3 + \overline{y2}y1)$$

$$y3 = y1y2w$$

$$z = y3$$

These expressions lead to the following circuit, shown in figure 2.2:

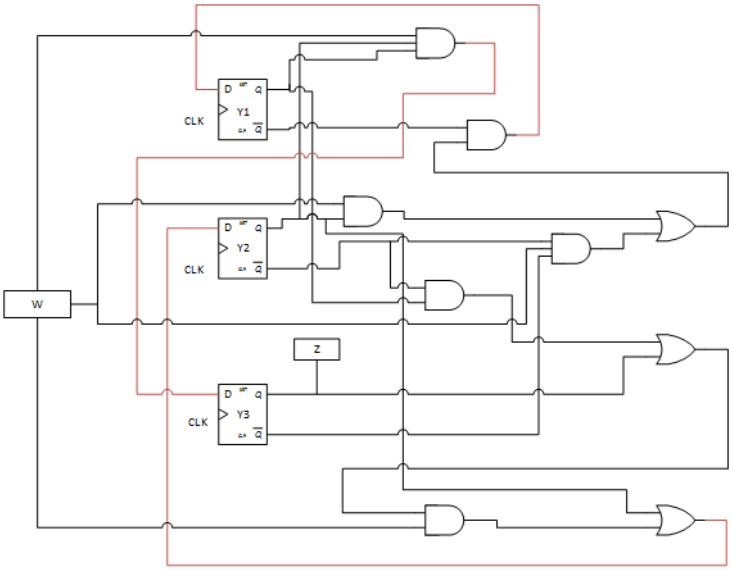

Figure 2.2: Logical circuit for the Moore's state machine.

The previous figure 2.2 shows how the present state is saved using the three D flip-flops, each saving one of the bits that form the represented state's value.

2.1.1 Coding and testing of the Moore's State Machine

This state machine was written in verilog, considering the previous state and deciding what state whould come next, according to the input. In this case, each state has an output associated. The code was tested using a 64-bit sequence arriving to the state machine through the input. This sequence considers the different cases for the detection of the sequence and the cases in which it should go back to the IDLE state. Two different tests were carried out. One of them without resetting the state machine, in order to check the correct functioning of the transitions among states. The other test was made using the same input sequence, and adding another reset sequence, in which the reset response to a reset is tested in all of the states. The tests are clearly seen using gtkwave, and the results are shown bellow in figures 2.3 and 2.4.

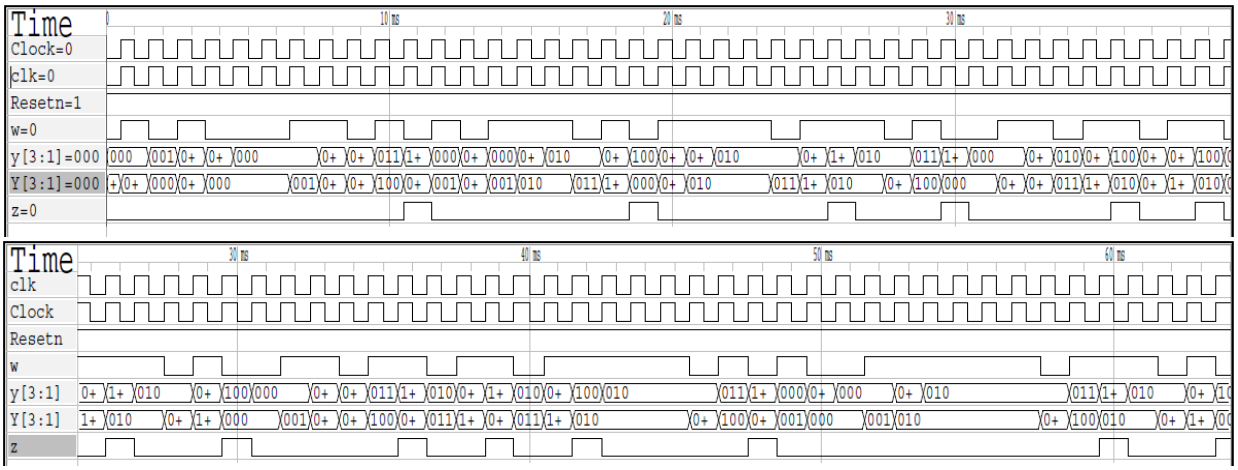

Figure 2.3: Test of Moore's State Machine without resetting.

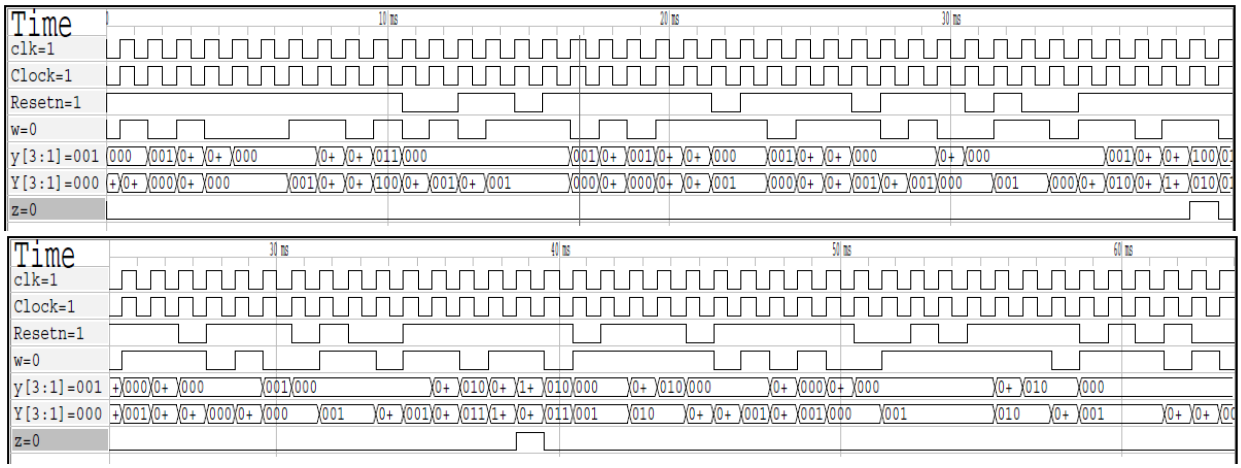

Figure 2.4: Test of reset in Moore's State Machine.

In the previous figures 2.3 and 2.4, as previously mentioned, w is the input, y is the present states' vector, Y the next state's vector and z the output. It can be seen that the states' transition works correctly according to the input and to the reset signals. The output is 1 when the sequence 1-1-0-1 is detected.

2.2 Mealy Type State Machine

In figure 2.5 it is represented the sequence detection with the Mealy's state machine.

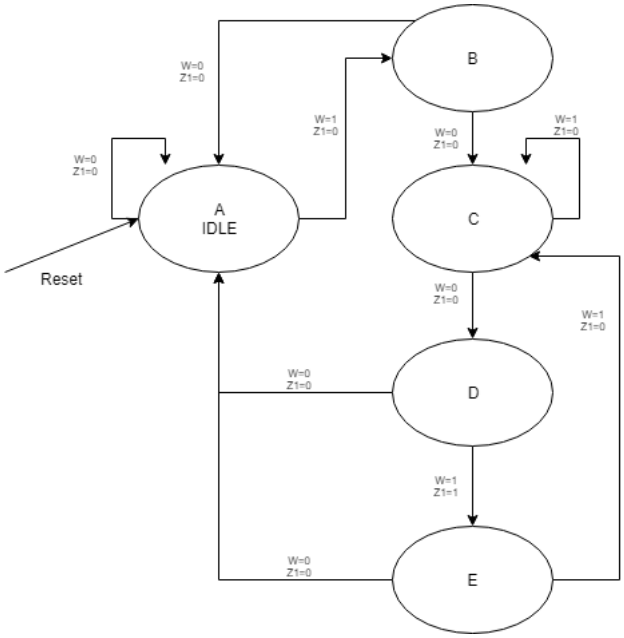

Figure 2.5: Mealy's State Machine, for detecting the sequence 1-1-0-1.

It can be seen how in this case, the transition between one state to another now also causes a change in the output. It doesn't wait to get to the new state in order to change it. The following table is obtained from above's diagram in figure 2.2 .

Present State	Next State depending on w		Output z depending on w	
	w=0	w=1	w=0	w=1
A (000)	A (000)	B (001)	0	0
B (001)	A (000)	C (010)	0	0
C (010)	D (011)	C (010)	0	0
D (011)	A (000)	E (100)	0	1
E (100)	A (000)	C (010)	0	0

Table 2.2: Mealy's Next States and Outputs.

2.2.1 Coding and testing of the Mealy's State Machine

The same tests with the same input sequences used for Moore's state machine, were used to test the Mealy's state machine. The results are seen in figures 2.6 and 2.7.

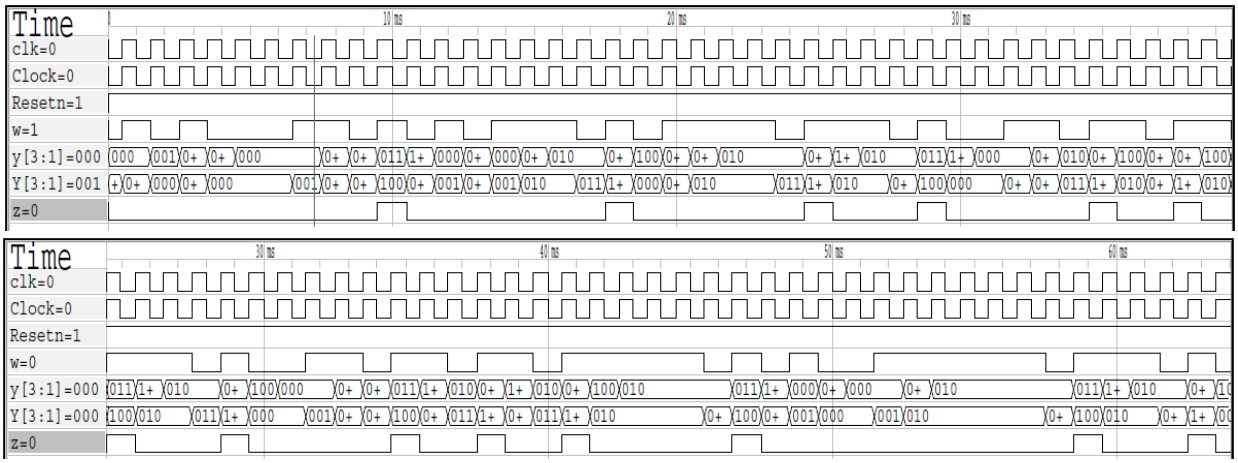

Figure 2.6: Test of Moore's State Machine without resetting.

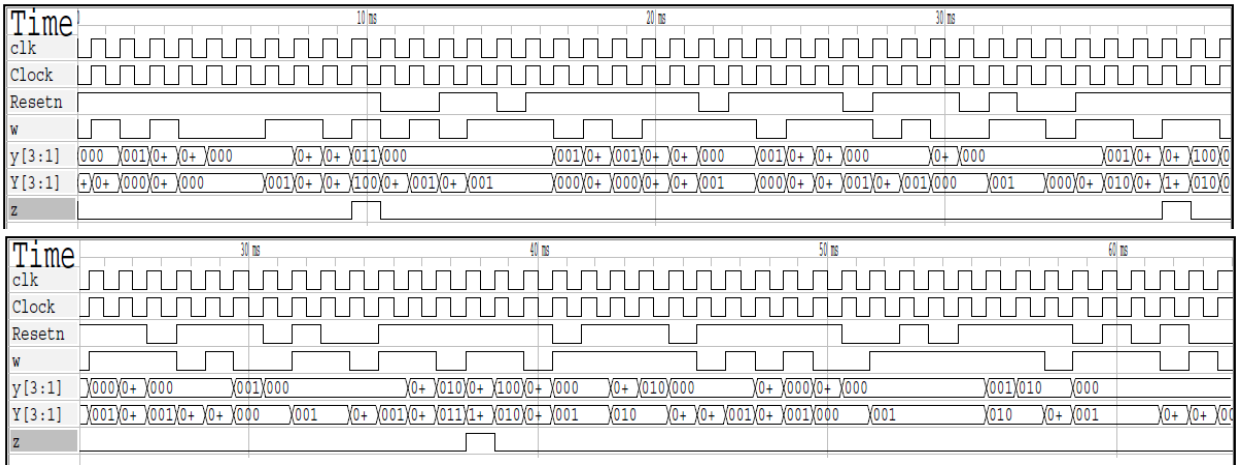

Figure 2.7: Test of reset in Moore's State Machine.

As well as in the previous case, it can be seen that the Mealy's state machine works correctly according to the

input w and to the reset signals.

2.2.2 Comparison of Moore's State Machine with Mealy's State Machine

Even though both state machines work as expected and although they both answer correctly to the changes of states, in the gtkwave simulations it can be clearly seen the difference between the effect that each of the state machines have on the output z. In Moore's case, the output is high one clock after the sequence finishes appearing through the input. However, this one-clock delay doesn't happen in Mealy's State Machine. In the second one, the output doesn't only depend on the present state, but also on the input signal when going from one state to another. So the output is seen immidiatly when the last bit of the input signal arrives. This means that the Moore's output signal is the same as Mealy's one, but occuring one clock later.

3 Exercise 3

For this exercise we were asked to implement the Moore machine shown on Figure 3.1, but with one condition, everything inside our machine should work on 3.3v and inputs and outputs should work on 5v logic.

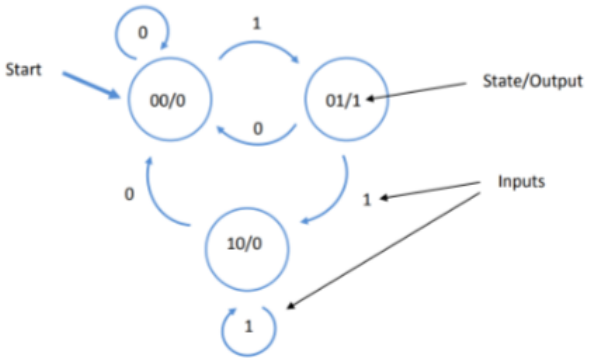

Figure 3.1: Moore Machine

3.1 State Machine Implementation

We proceeded to convert the diagram to a table that represents it, and we got Table 3.1, and by transforming each bit (Output, and next states bits) into a truth table, we could create our logic implementation for each bit. The Truth Tables are shown on Tables 3.2.

State	W=0	W=1	Output
00	00	01	0
01	00	10	1
10	00	10	0

Table 3.1: Moore Machine Table

S1 S0	Output
00	0
01	1
10	0
11	x

S1	S0	W	S1'	S0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	x	x
1	1	1	x	x

Table 3.2: Truth Tables

So by re-writing those truth tables we have got the following equations:

$$Output = S_1 . \bar{S_0} \tag{3.1}$$

$$S_1^{'} = \bar{S_1} S_0 w + S_1 \bar{S_0} w$$

$$S_0^{'} = \bar{S_1} \bar{S_0} w$$

So, by implementing those formulas into a synchronized logic circuit, we have got the Moore machine.

3.2 Level Converter Implementation

To convert the voltage levels on the PCB for compatibility, we decided to use 2N7000 Mosfet Transistor. We used it to implement an inverter with Open-Collector. By connecting the output to a pull-up network with the voltage we want, we can convert the level with no problems, form 5(v) to 3.3(v), and vice versa.
For the Pull-Up network, we decided to use a resistor $R = 1\,(k\Omega)$ so that when it's conected to ground, te current flowing through the inverter is less than $10\,(mA)$, and when the inverter produces a High Z, the resistor its not enough big to produce a High Z to the output, and set the output to a logic 1.

3.3 PCB Implementation

We proceeded to implement the PCB using Altium Designer, the Schematic for this Finite State Machine is shown on Figure 3.2. The Top and Bottom Layers are shown on Figure 3.3.

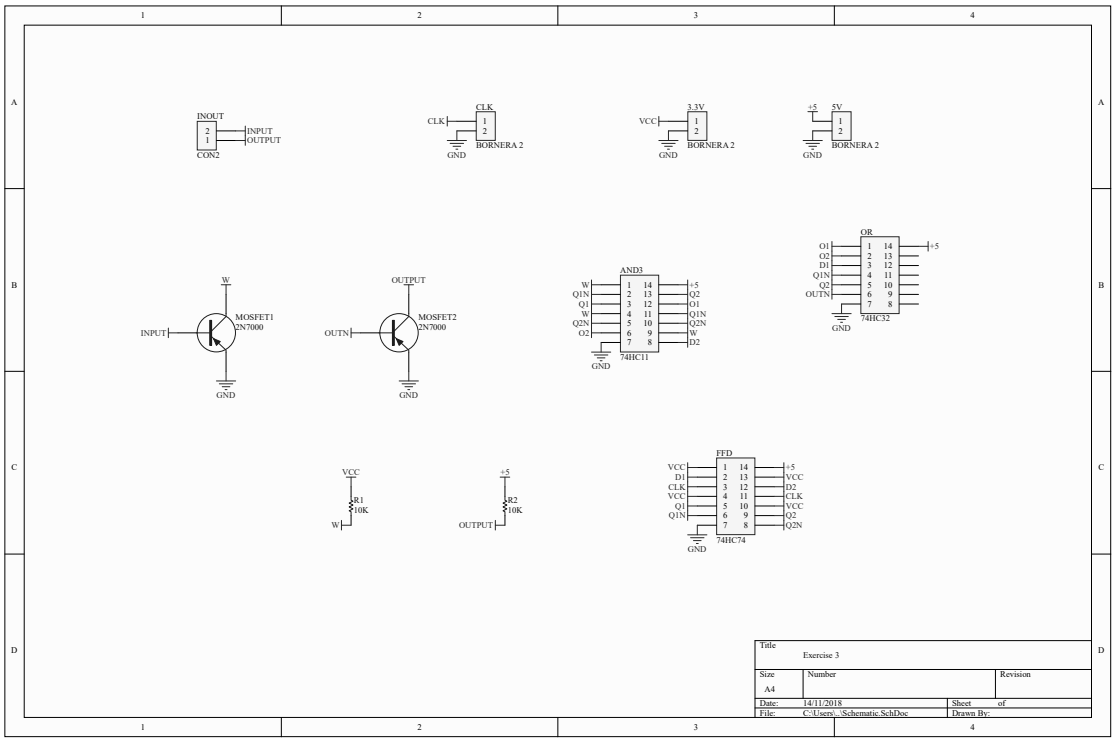

Figure 3.2: Schematic

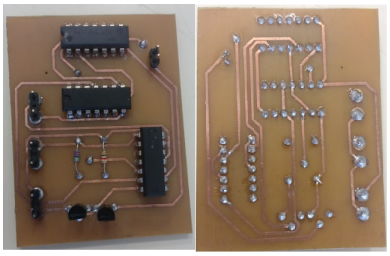

Figure 3.3: Printed Board

3.4 Mealy State Machine Re-Implementation

Finally, we were asked to re-implement the Moore State machine described in the previous Subsections, into a Mealy State Machine. As we know, Mealy Machines Outputs depend on states and inputs, different from Moore that only depends on states. So the truth tables for the next state, mantains its form and formula as the moore state machine, however, the output truth table, and by consequence, its formula , change ash shown on Table 3.5.

State	W=0	W=1
00	00/0	01/1
01	00/0	10/0
10	00/0	10/0

Table 3.3: Mealy State Machine

We can now easily see that in Table 3.3, states 01 and 10 are equal, so we can simplify those states into only one state. The final Table is shown on Table 3.4.

State	W=0	W=1
0	0/0	1/1
1	0/0	1/0

Table 3.4: Mealy Machine simplified

State	W	Output
0	0	0
0	1	1
1	0	0
1	1	0

Table 3.5: Output Truth Table

Finally, the formulas for the output and the next states are the following:

$$Output = \bar{St}.W$$

$$State = W$$

Implementing this would become someting like shown on Figure 3.4.

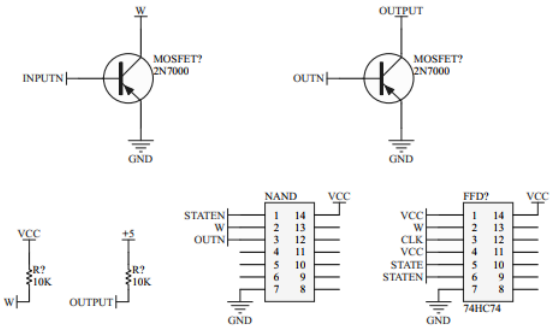

Figure 3.4: Schematic Mealy Machine

We builded this schematic in a breadboard, as shon on Figure 3.5.

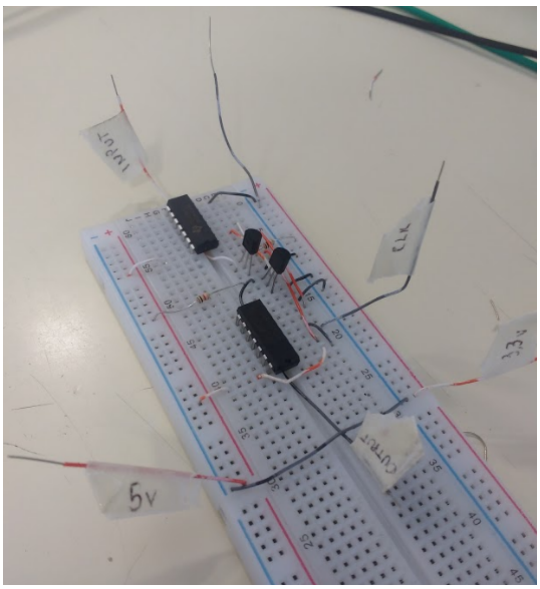

Figure 3.5: Mealy Implementation

3.5 Conclusions

By analizing the behaviour of both circuits, how they behave in the same test conditions, and comparing the amount of components used in both PCBs, we conclude that in this case, Mealy's implementation is more efficient because it utilizes less components and realizes the same operation. However, we need to clarify that not always Mealy's implementation is more efficient than Moore's, this depends on each state machine to implement.