Universidad Carlos III de Madrid

Laboratory Report

Integrated Circuits and Microelectronics

Andrés Navarro Pedregal (100451730) & Daniel Toribio Bruna (100454242)

Dual Bachelor in Data Science and Engineering and Telecommunication Technologies Engineering

March 15, 2024

Contents

1 Introduction

In this lab report, our goal is to create a waveform generation and FIR (Finite Impulse Response) filter implementation. Our objective is to understand and construct circuits that can generate sinusoidal signals and filter them effectively. These circuits will be designed to operate on FPGA (Field-Programmable Gate Array) boards, specifically the Basys 3 model, utilizing components such as LEDs and digital-to-analog converters (DACs).

Session 1: Waveform Generator In our first session, we create a circuit capable of generating sinusoidal signals represented with 8 bits of precision. This circuit will be responsible for producing these signals at different frequencies, which will be selected through input switches. The generated waveform will be visualized through both LEDs on the FPGA board and an 8-bit DAC (Pmod R2R), ensuring versatility in signal output. Our design will consist of various components including timers, memory units, and counters to facilitate accurate signal generation and display.

Session 2 and 3: FIR Filter Implementation Moving forward, we implement a digital FIR filter alongside the previously developed waveform generator. The FIR filter serves the purpose of refining the generated signals by attenuating frequencies beyond a specified cutoff point. Utilizing filter coefficients obtained from MATLAB, we construct a filter with a predetermined number of stages to achieve the desired filtering effect. Integrating this filter into our existing circuitry, we aim to enhance the quality of the generated signals for various applications.

Throughout these sessions, we'll engage in simulation, synthesis, and practical implementation of our designs on FPGA boards. Additionally, we'll document our progress through test benches, oscilloscope measurements, and final reports, ensuring a comprehensive understanding of the design process and its outcomes.

By the end of these sessions, we anticipate gaining valuable insights into circuit design, signal processing, and FPGA-based system implementation, laying a solid foundation for further exploration in the field of integrated circuits and microelectronics.

2 Design Characteristics

2.1 Full Implemented Functionality

For this circuit, the main functionality is the representation of a signal generation and a FIR filter.

The signal generation will generate a sine signal by means of using timers to modulate the frequency, and a ROM to store the values of the signal. By using a high enough clock frequency, the signal will assimilate almost the same as a continuous sine signal. In the pictures below of the implemented circuit we can see a perfect signal.

For the FIR filter, we will be using different architectures to apply the filter at the output of the signal. For this laboratory we have used 3 different architectures: parallel and pipiline with 2 different set ups. The details of each architecture can be seen below.

2.2 I/O Interface, Type And Functionality

For the interface we will have the different inputs and outputs:

- Clk: it is the clock that the system works in. In order to ensure a proper behaviour, a clock of 10MHz must be used. It is a logic signal with 0 and 1 as values.
- Reset: it is the signal to reset the whole circuit. It is a logic signal with 0 and 1 as values.
- Per: it will be the selector of the frequency in used. It will be a two pin logic signal with 4 different values. Each value for each of the different frequency in used.
- Led: it will be the output of the signal after the filter. It will be a 8 bits with two-complement representation.
- Dac: this will be the same output as the led of 8 bits but using positive numbers.

3 Design Structure

3.1 Block Diagram

3.2 Component Description

3.3 Calculation

3.3.1 Frequencies

1. Values:

   We calculated 16 values of a sine signal with the following code:

```
import math

 # Define parameters
amplitude = 127
frequency = 1  # Adjust frequency as needed
num_points = 16

 # Generate values
values = []
for t in range(num_points):
        values.append((int) (amplitude * math.sin((2 * math.pi * t) / num_point

 # Print the values
for i, val in enumerate(values):
        print(f"f(t_{i}) =", val)
```

```
f(t_0) = 0
f(t_1) = 48
f(t_2) = 89
f(t_3) = 117
f(t_4) = 127
f(t_5) = 117
f(t_6) = 89
f(t_7) = 48
f(t_8) = 0
f(t_9) = -48
f(t_10) = -89
f(t_11) = -117
f(t_12) = -127
```

```
f(t_13) = -117
f(t_14) = -89
f(t_15) = -48
```

2. Periods:

   The frequencies we needed to use were 600, 1000, 2200, and 3900 Hz. For the period, we calculated that with the following code

```
frequencies = [600, 1000, 2200, 3900]
num_points = 16
clock_frequency = 10e7

 # Generate values
values = []
for i in range(len(frequencies)):
        values.append((int) (clock_frequency / (frequencies[i] * num_points)))

 # Print the values
for i, val in enumerate(values):
        print(f"time({i}) =", val)

time(0) = 10416
time(1) = 6250
time(2) = 2840
time(3) = 1602
```

3.3.2 Filter Coefficients

For the coefficients of the filter, we used a 12 order filter, therefore we needed 13 different values. The calculations can be seen below.

3.4 Simulations

For the simulations, 2 periods of each signal where performed. Note that the filter has a cutoff frequency of 1000 Hz.

For the first frequency, the signal passes without a problem. For the second frequency, it reduces by half as it is the same as the cutoff frequency where the filter will attenuate the signal by one half. And for the other signals, the filter filters out completelly the sine wave.

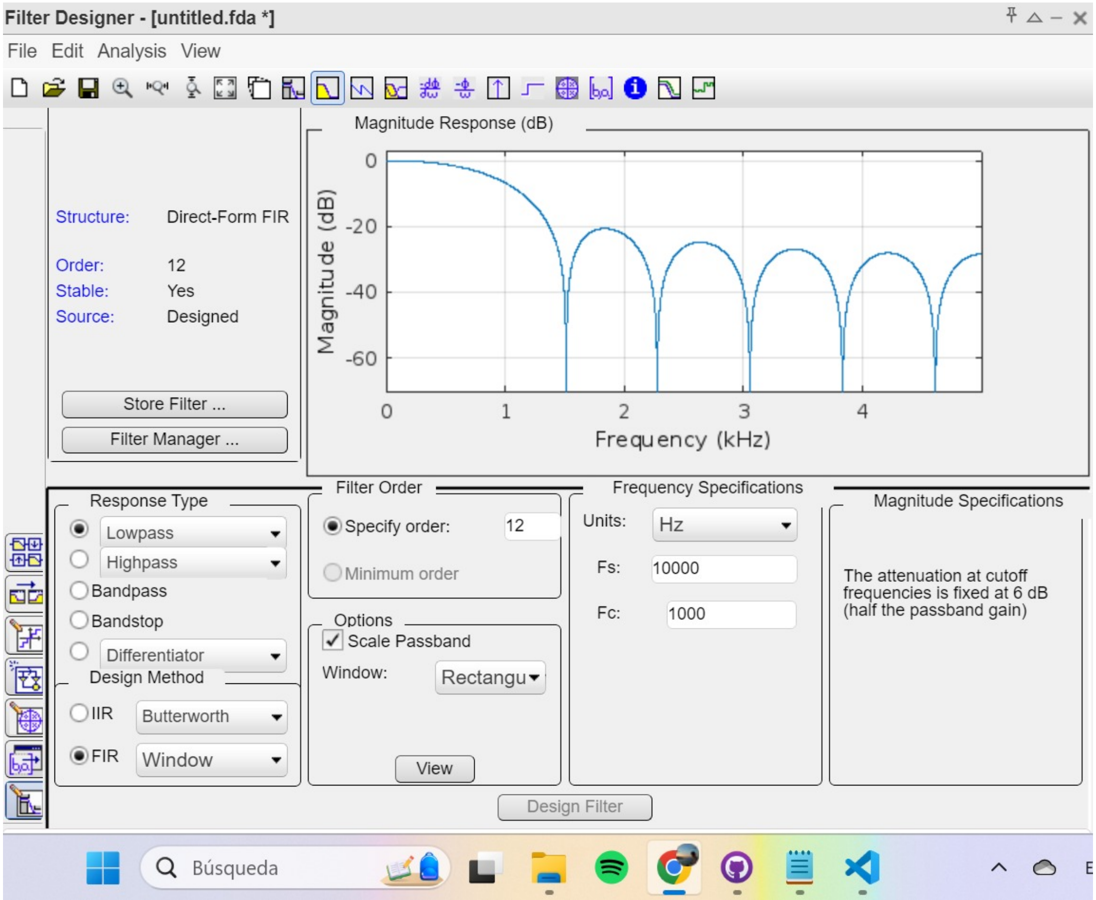
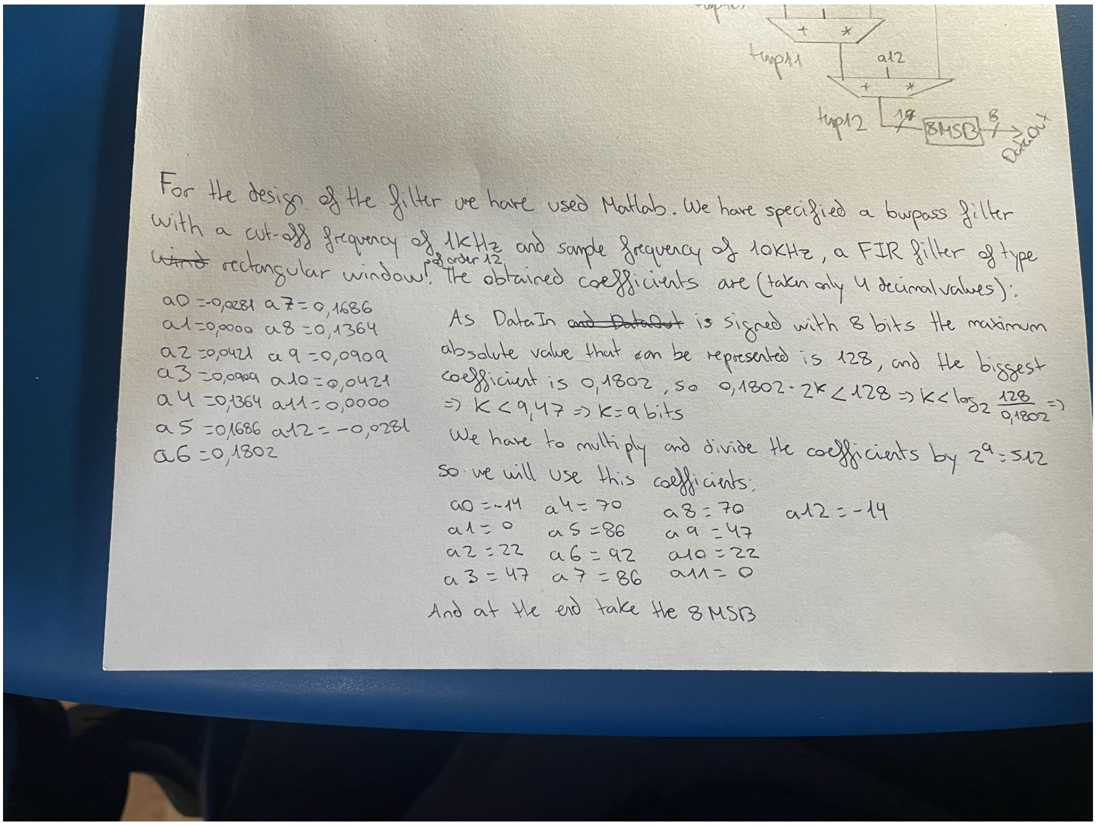


Figure 1: Matlab Coefficient Calculations



For the design of the filter we have used Matlab. We have specified a lowpass filter with a cut-off frequency of 1KHz and sample frequency of 10KHz, a FIR filter of order 12 of type ~~wind~~ rectangular window! The obtained coefficients are (taken only 4 decimal values):

a0 = -0,0281 a7 = 0,1686
a1 = 0,0000 a8 = 0,1364
a2 = 0,0421 a9 = 0,0909
a3 = 0,0909 a10 = 0,0421
a4 = 0,1364 a11 = 0,0000
a5 = 0,1686 a12 = -0,0281
a6 = 0,1802

As DataIn ~~and DataOut~~ is signed with 8 bits the maximum absolute value that can be represented is 128, and the biggest coefficient is 0,1802, so $0{,}1802 \cdot 2^K < 128 \Rightarrow K < \log_2 \frac{128}{0{,}1802} \Rightarrow K < 9{,}47 \Rightarrow K = 9$ bits

We have to multiply and divide the coefficients by $2^9 = 512$ so we will use this coefficients:

a0 = -14 a4 = 70 a8 = 70 a12 = -14
a1 = 0 a5 = 86 a9 = 47
a2 = 22 a6 = 92 a10 = 22
a3 = 47 a7 = 86 a11 = 0

And at the end take the 8 MSB

Figure 2: Filter Coefficients Adaptation For Use.

Figure 3: VHDL Test Bench Simulation

4 Architectures

Different architectures have been proposed to reduce the critical region of the filter so we can use higher frequencies. Note, that the VHDL code is separated in different files for each filter.

4.1 Parallel Architecture

For the parallel architecture, we use the following diagram.

This architecture has a big critical region compared to the pipeline architecture as there are no intermediate registers to save the information.

4.2 One Flip-Flop Pipeline Architecture

For the pipeline architecture with one intermediate step, we use the following diagram.

We added a single temporary flip-flop to reduce the critical region of the filter by half. Therefore, we expect that even though we will increase the area of the implementation we increase the maximum frequency that the filter can work at.

4.3 Two Flip-Flop Pipeline Architecture

For the final part, we added a second stage in the pipeline architecture to see if it further increases the maxiumum frequency.

5 Synthesis Results

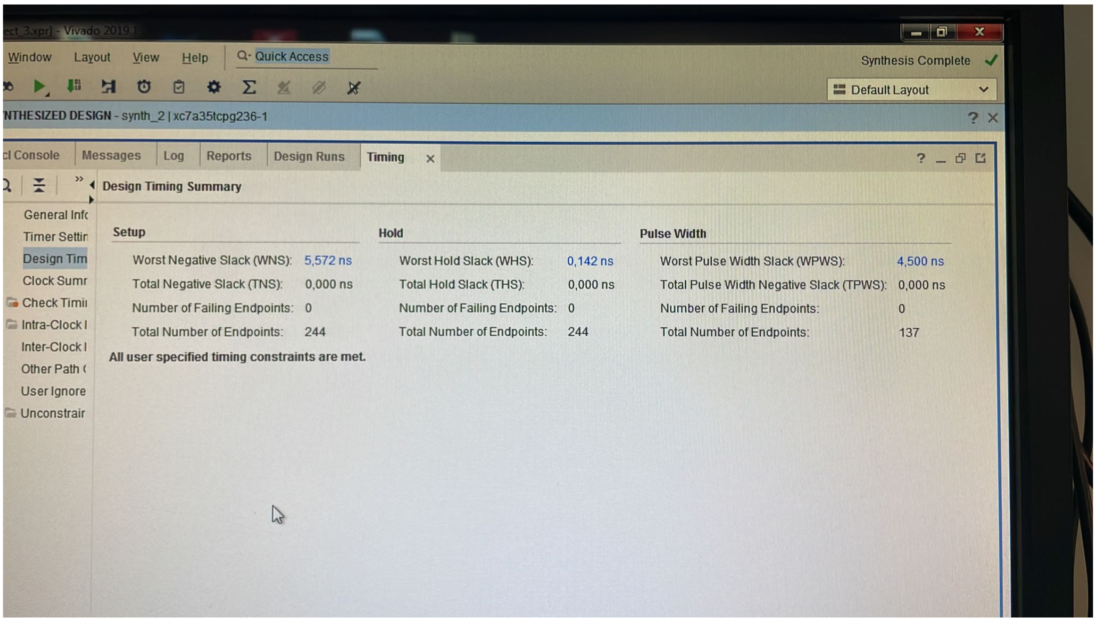
	Parallel	One Flip-Flop Pipeline	Two Flip-Flop Pipeline
Logic LUTS	462	465	462
Flip Flop Registers	136	161	186
WPWS			
Max Frequency			

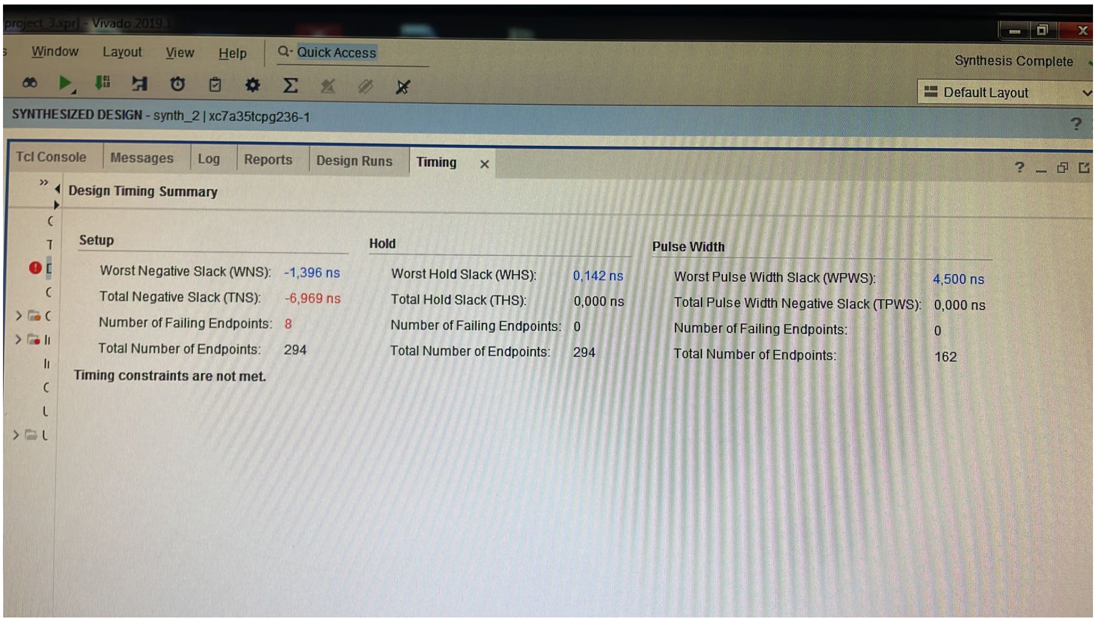
As we can see in the results from the table above our assumptions are correct. As the pipeline architecture is an implementation of the parallel one but with more registers to save temporary data, the number of flip-flops, therefore the area increases the more intermediate steps we add.

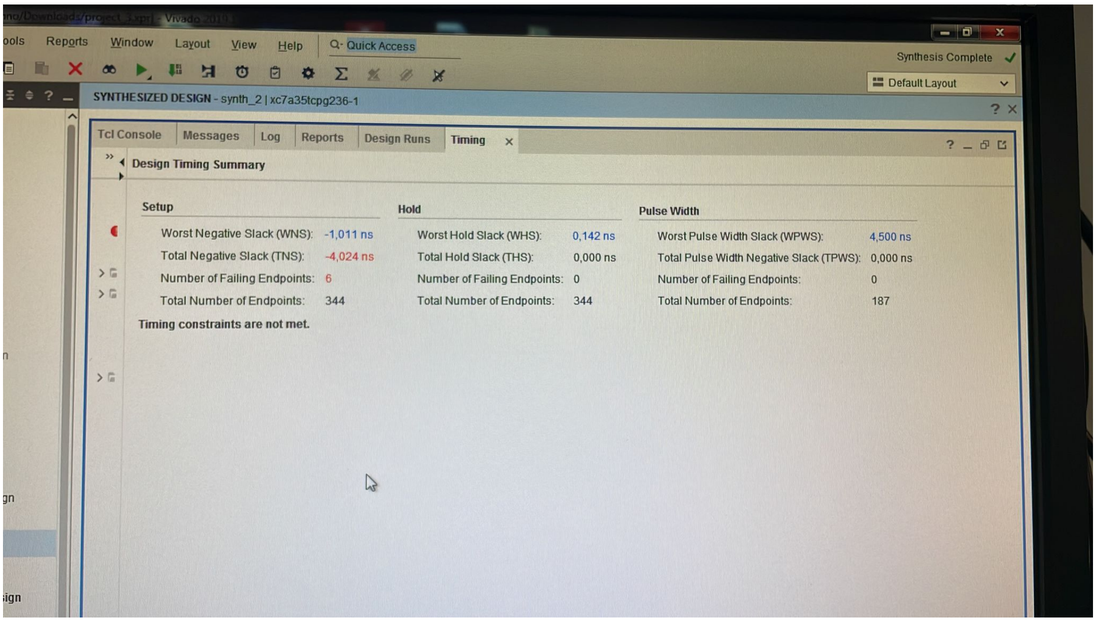
This temporary registers are used to reduce the critical region of the filter so the filter can be used at higher frequencies as seen.

Finally, the LUTS (look up tables) stay almost constant among the 3 architectures analyzed. This is as the computations are the same for the 3 cases.

5.1 Time Results




Window Layout View Help
Quick Access
Synthesis Complete
Default Layout
SYNTHESIZED DESIGN - synth_2 | xc7a35tcpg236-1
Tcl Console Messages Log Reports Design Runs Timing
Design Timing Summary
Setup
Worst Negative Slack (WNS): -1,396 ns
Total Negative Slack (TNS): -6,969 ns
Number of Failing Endpoints: 8
Total Number of Endpoints: 294
Timing constraints are not met.
Hold
Worst Hold Slack (WHS): 0,142 ns
Total Hold Slack (THS): 0,000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 294
Pulse Width
Worst Pulse Width Slack (WPWS): 4,500 ns
Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 162


Reports Window Layout View Help
Quick Access
Synthesis Complete
Default Layout
SYNTHESIZED DESIGN - synth_2 | xc7a35tcpg236-1
Tcl Console Messages Log Reports Design Runs Timing
Design Timing Summary
Setup
Worst Negative Slack (WNS): -1,011 ns
Total Negative Slack (TNS): -4,024 ns
Number of Failing Endpoints: 6
Total Number of Endpoints: 344
Timing constraints are not met.
Hold
Worst Hold Slack (WHS): 0,142 ns
Total Hold Slack (THS): 0,000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 344
Pulse Width
Worst Pulse Width Slack (WPWS): 4,500 ns
Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 187

6 Hardware Results

For the hardware results we have tested with the oscilloscope the 4 signals to see the filter in action. In the pictures below, there are two signals. The one above is the sine filtered, and the one below is the sine not filtered.

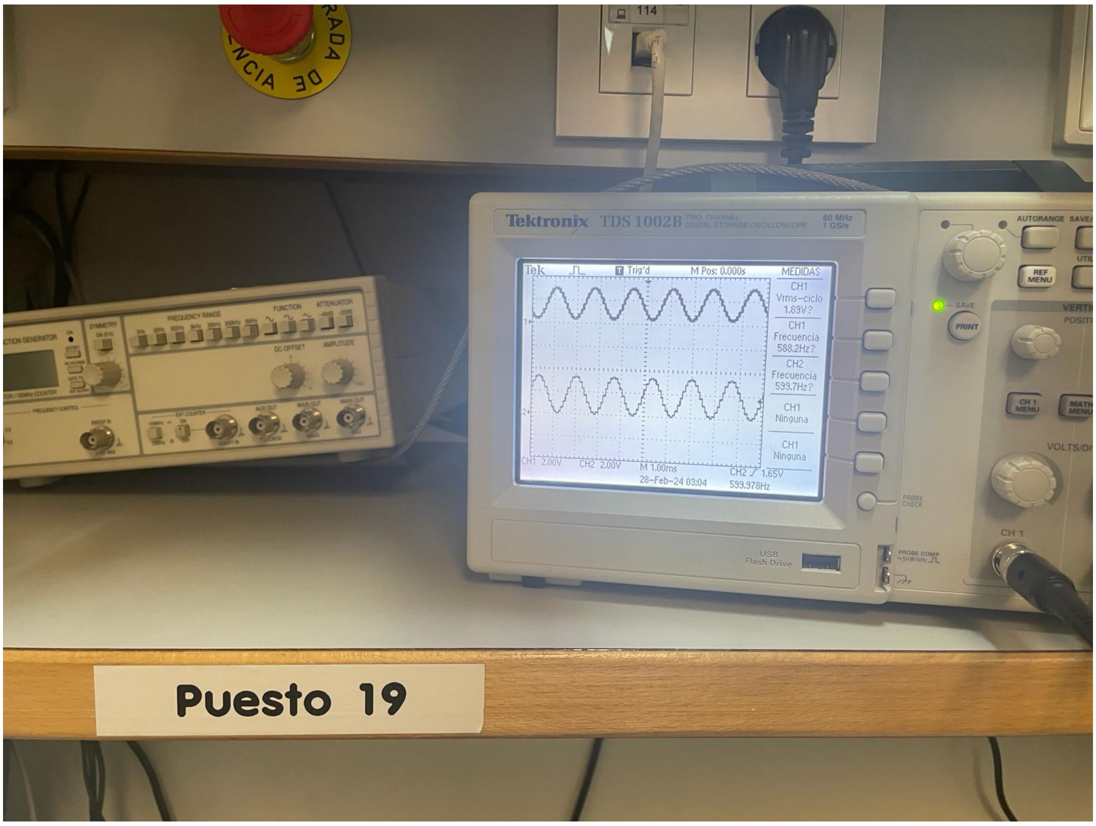


Figure 4: For the frequency 600Hz, we can see that the filter allows the sinal to pass as it is below the cut-off frequency. The signal is almost not attenuated.

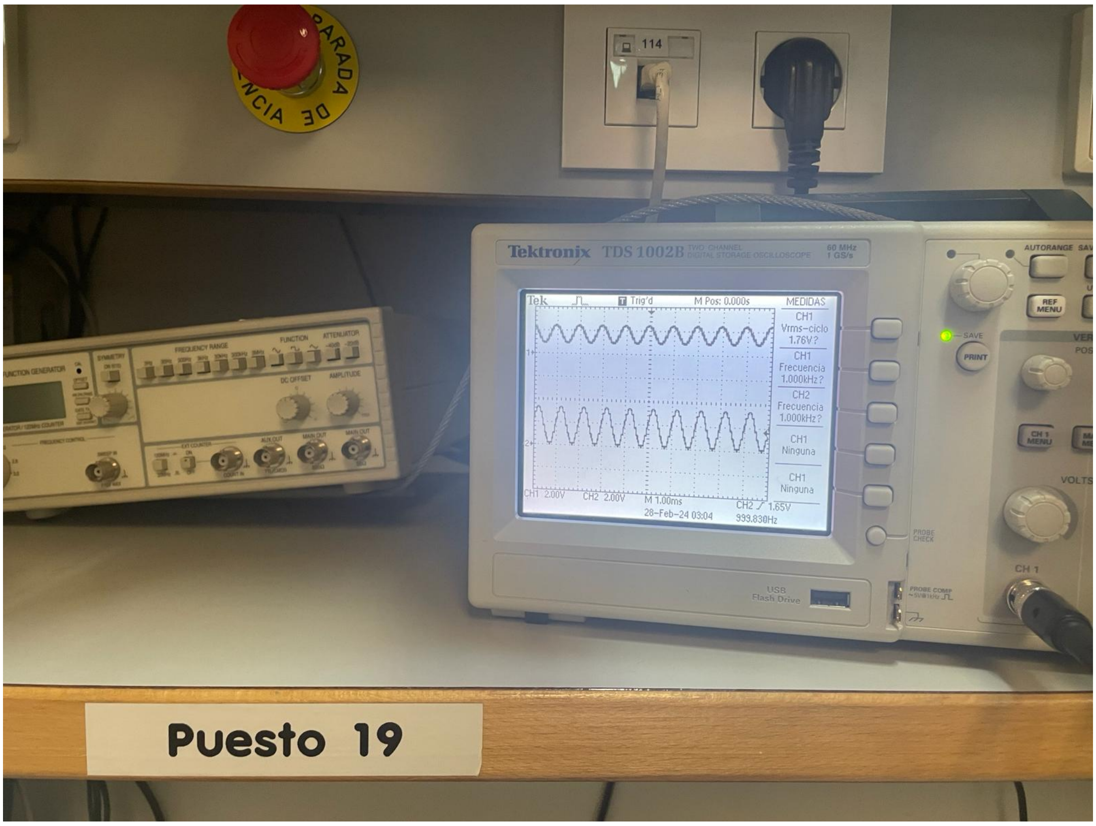


Figure 5: For the frequency 1000Hz, we can see that the filter allows the sinal to pass as it is below the cut-off frequency. The signal is almost not attenuated.

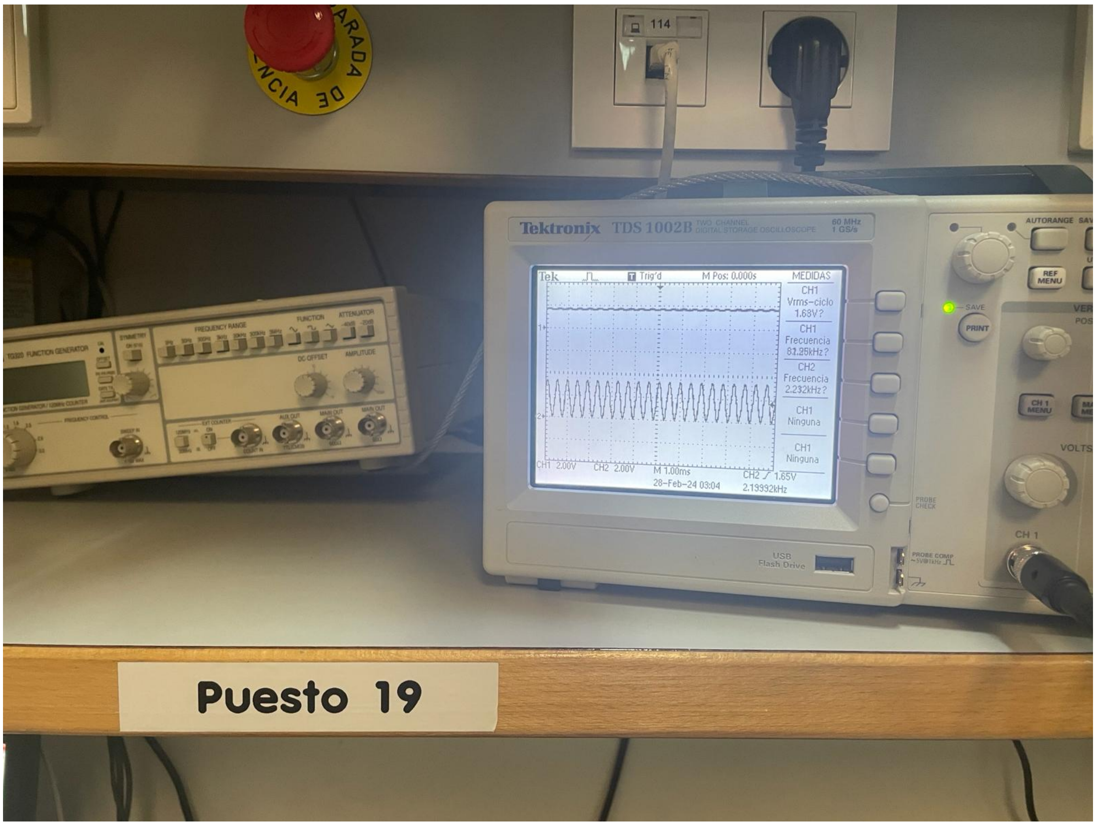


Figure 6: For the frequency 2200Hz, we can see that the filter filters the whole signal as it is expected.

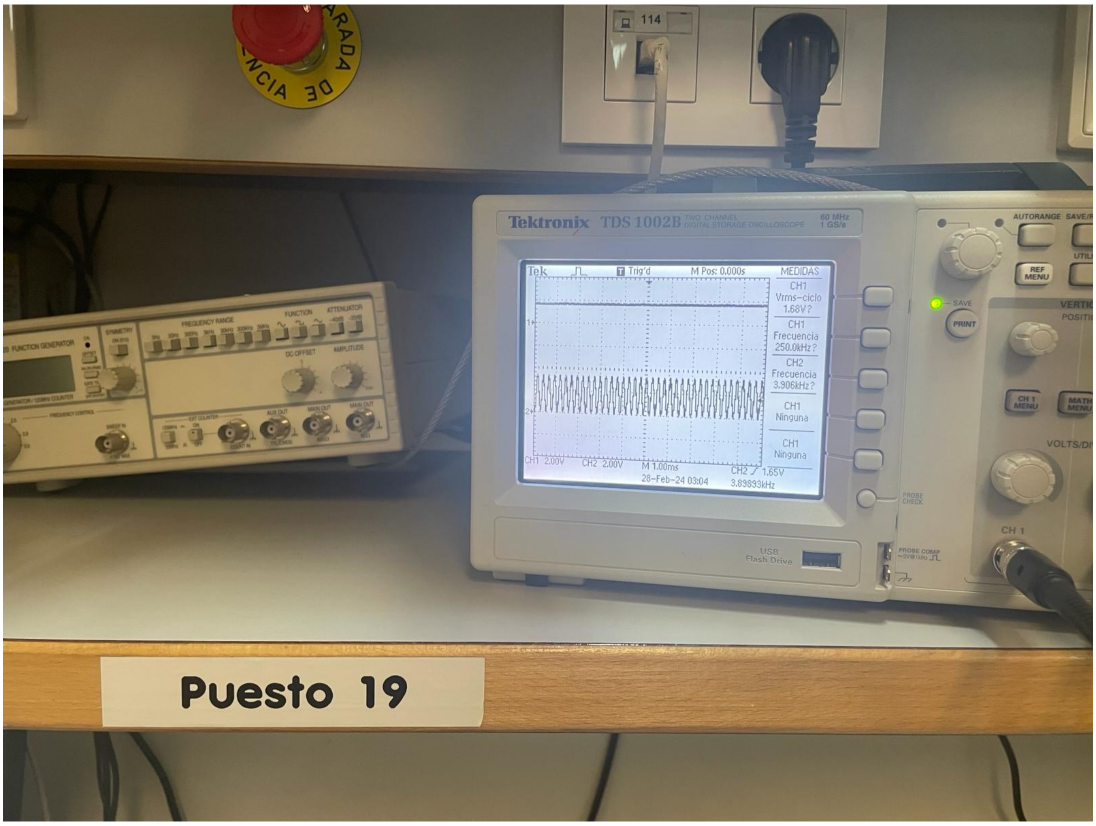


Figure 7: For the frequency 3900Hz, we can see that the filter filters the whole signal as it is expected, even more than the 2200Hz frequency.

7 Conclusion

All in all, in this laboratory we have learned how to develope a signal generator and FIR filter in a FPGA. We have designed the filter with different architectres to see the different implementations. In the end, we were able to identify the advantages and drawback of each implementation, and the ability to increase the clock cycle of a system by decreasing the critical path of our code with the use of Flip-Flops.