UNIVERSITY OF PISA

COMPUTER ENGINEERING MASTER DEGREE

Electronics and Communications Systems

Square Root Raised Cosine FIR Filter

Professors:

Luca Fanucci

Pietro Nannipieri

Luca Zulberti

Candidate:

Alice Orlandini

ACADEMIC YEAR 2023/2024

Contents

1 Introduction **2**
- 1.1 Specifications . . . 2
- 1.2 FIR Filters . . . 2
- 1.3 Square Root Cosine Filter . . . 3
- 1.4 Possible Applications . . . 4

2 Architecture Description **4**
- 2.1 Simplifications . . . 4
- 2.2 Coefficients Representation . . . 5
- 2.3 Design . . . 5

3 Implementation **8**
- 3.1 Testbench and Simulation . . . 10

4 Vivado Logic Synthesis and Implementation **10**
- 4.1 Synthesis and Implementation Summary . . . 10
- 4.2 Timing . . . 11
- 4.3 Utilization . . . 12
- 4.4 Power Consumption . . . 13
- 4.5 Area Occupation . . . 14
- 4.6 Warnings and Errors . . . 14

5 Conclusions **15**

1 Introduction

1.1 Specifications

The purpose of this project is to design a digital circuit that realizes a **FIR filter** (*Finite Impulse Response*) with a **SRRC** (*Square Root Raised Cosine*) impulse response and with the following characteristics:

- the filter is $N = 22$;
- the samples per symbol equal to 4;
- the roll-off factor is equal to 0.5;

The equation to be implemented is:

$$y[n] = \sum_{i=0}^{N} c_i \cdot x[n-i]$$

and the predefined coefficients are:

$c_0 = c_{22} = -0.0165$	$c_1 = c_{21} = -0.0150$	$c_2 = c_{20} = 0.0155$
$c_3 = c_{19} = 0.0424$	$c_4 = c_{18} = 0.0155$	$c_5 = c_{17} = -0.0750$
$c_6 = c_{16} = -0.1568$	$c_7 = c_{15} = -0.1061$	$c_8 = c_{14} = 0.1568$
$c_9 = c_{13} = 0.5786$	$c_{10} = c_{12} = 0.9745$	$c_{11} = 1.1366$

In the project, a **16-bit representation** will be used for inputs, outputs, and coefficients.

1.2 FIR Filters

Some of the most common digital filters available in the market are **Finite Impulse Response (FIR) filters**, which are causal discrete-time Linear Time-Invariant (LTI) systems with a finite impulse response.

A discrete-time causal LTI system is called an FIR filter if the response $h(n)$ to the unit impulse is finite, meaning that $h(n) = 0$ for $n < 0$ and $n \geq M$ with $M > 0$.

The two main characteristics of this type of filters are:

- An FIR filter is always **causal** and **stable**.
- An FIR filter can have a **linear phase**: if the function $h(n)$ is symmetric or antisymmetric with respect to $\frac{M-1}{2}$.

In an FIR filter, the output value $y(n)$ at time n is given by:

$$y(n) = \sum_{k=0}^{M-1} c_k \cdot x(n-k)$$

Therefore, the calculation of $y(n)$ requires:

1. The availability of **input values** at times $n, ..., n - M + 1$.
2. The permanent availability of the **multiplicative coefficients** $c_1, ..., c_{M-1}$;
3. The execution of **multiplications** and **summations** according to the general form.

These operations can be implemented on specialized hardware, which will be described later.

1.3 Square Root Cosine Filter

The **Raised Cosine Filters** are commonly used in digital data communication systems to limit **inter-symbol interference** (ISI).

Intersymbol Interference (ISI) is a phenomenon that occurs in limited bandwidth communication systems where transmitted symbols overlap in time. This can lead to challenges in the demodulation and decoding process as adjacent symbols can influence each other. Consequently, this may result in a degradation of communication system performance.

Band-limited **Nyquist Filters** are a category of filters **designed to significantly reduce inter-symbol interference**, and the raised cosine filter is one such example. Specifically, in the Square Root Raised Cosine Filter, the function $H(f)$ is implemented using two half-cosine periods joined by a straight line. Figure 1 illustrates the behavior of $H(f)$ for various choices of the parameter $0 < \beta < 1$, referred to as the **roll-off coefficient**. This coefficient represents the index of dispersion of the cosine branch around the Nyquist frequency $\frac{f_s}{2}$.

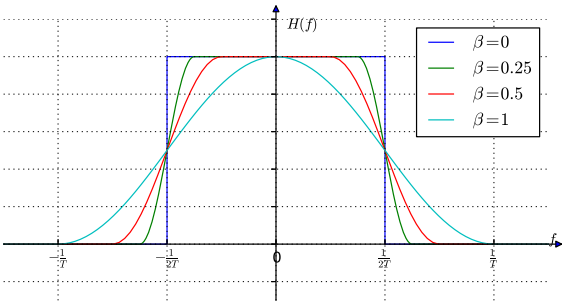

Figure 1: Frequency response of raised-cosine filter with various roll-off factors.

1.4 Possible Applications

A primary application of the Square Root Raised Cosine impulse response FIR filter is in the field of telecommunications, particularly in **digital communication systems**. It is indeed used to enhance spectral efficiency by reducing inter-symbol interference.

In the field of digital audio, this filter is employed in **audio signal processing**. It is used to improve sound quality, reduce noise, and adjust the frequency response of audio systems. This contributes to providing a clearer and more faithful listening experience for users of digital audio devices.

Moreover, in **digital medical devices**, such as those used for detecting biological signals, the FIR SRRC circuit can be employed to filter and process vital signals. For instance, in electrocardiography (ECG) or electroencephalography (EEG), the filter helps eliminate unwanted interference, providing more accurate data for medical diagnosis.

Finally, advanced applications like **computer vision** benefit from the FIR circuit's capabilities to enhance image quality, reduce noise, and preserve crucial details. It can thus be utilized for the analysis of complex visual signals.

In conclusion, the SRRC impulse response FIR circuit is a versatile component with a wide range of applications.

2 Architecture Description

2.1 Simplifications

In this section, an operation will be performed aimed at simplifying the formula used to calculate the output $y[n]$. It is indeed observed that half of the coefficients c_k have the same value, so it is possible to group the identical coefficients.

$$\begin{aligned} y[n] &= \sum_{k=0}^{M-1} c_k \cdot x[n-k] = \\ &= \left(\sum_{k=0}^{\frac{M}{2}-1} c_k \cdot (x[n-k] + x[n-N-k]) \right) + c_{\frac{M}{2}} \cdot x\left[n - \frac{M}{2}\right] \end{aligned}$$

Before this optimization, the number of adders and multipliers required to implement the circuit was 22 and 23, respectively. After the operation, the number of adders remained at 22 units, while the number of multipliers decreased to 12, achieving significant resource savings.

2.2 Coefficients Representation

As for coefficients, the **Fixed-Point representation** for real numbers was chosen. In this context, integers are scaled based on the weight of the Least Significant Bit (**LSB**).

The considered coefficients fall within the range [-0.1568; 1.1366] for their respective c_6 and c_{11}, and a 16-bit representation was used. Consequently, 4 decimal digits are required to achieve the desired accuracy, leading to:

$$\epsilon_A = 10^{-4}$$

Consequently, the calculation yields:

$$F = \left\lceil \log_2 \left(\frac{1}{\epsilon_A} \right) \right\rceil = 14$$

in such a way that:

$$LSB = 2^{-F} \leq \epsilon_A$$

Therefore:

$$LSB = 2^{-14}$$

Finally the quantization applied is:

$$Q[c_k] = floor \left(\frac{c_k}{LSB} \right) \times LSB$$

2.3 Design

The high-level architecture of the circuit is visible in Figure 2.

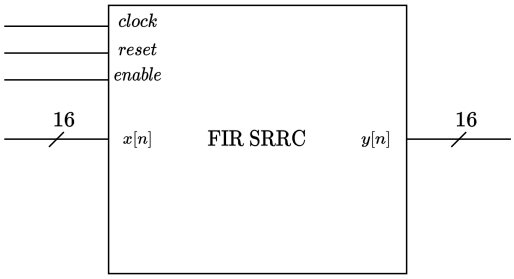

Figure 2: Circuit High-Level Representation

This architecture incorporates four inputs. Firstly, the **input** signal, represented by a width of 16 bits, is referred to as $x[n]$ and constitutes the main data flow in the circuit. Additionally, there are the ***clock***, ***reset***, and ***enable*** signals, each playing a key role in regulating the circuit's behavior. The circuit's **output** is also a 16-bit signal, denoted as $y[n]$.

Descending to a lower level, the circuit has been implemented as shown in Figure 3. In detail, to perform the operation, the procedure involves several phases.

Initially, the inputs are acquired and stored using ***Positive Edge-Triggered D-Flip-Flop*** connected in series, creating a structured sequence of data.

Subsequently, these data undergo a summation process, and each result is multiplied by the corresponding coefficient. The multiplications generate intermediate results, which are finally summed together to obtain a final value representative of the desired output of the circuit. The final result is then connected to an additional D-Flip-Flop, in order to ensure stability issues, and will yield the circuit's ultimate output.

In terms of sizing, it was decided to extend the outputs of the Flip-Flops from 16 to 17 bits, allowing to disregard the carry out of the adders. After multiplication, a total of 34 bits will be obtained. These outputs were not further extended for the final addition, as no carry out issues were detected during the testing phase. Finally, the output was taken into account by considering the most significant bits.

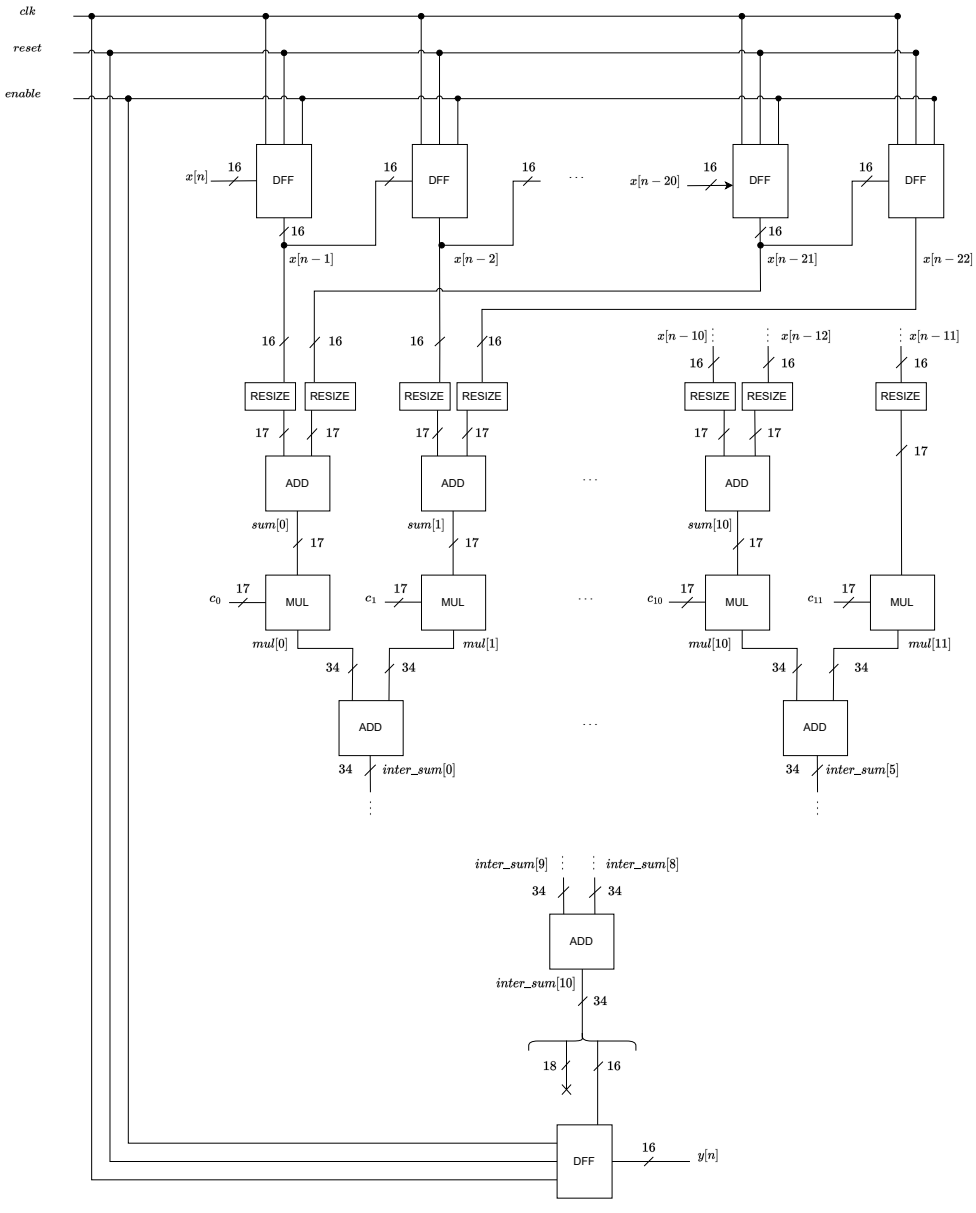

Figure 3: Circuit Low-Level Representation

3 Implementation

In this section, the focus will be on the analysis of the **VHDL** code that has been developed to implement the described functionality.

Below is the code for the **main entity** of the circuit: the `fir_srrc`.

```
entity fir_srrc is
    generic (
        FilterOrder : natural := 22;
        NBit : natural := 16
    );
    port (
        clk : in std_logic;
        resetn : in std_logic;
        enable : in std_logic;
        x : in std_logic_vector(NBit-1 downto 0);
        y : out std_logic_vector(NBit-1 downto 0)
    );
end entity fir_srrc;
```

The ***Filter Order*** and the ***Number of Bits*** for `x` and `y` have been set to 22 and 16, respectively, as per the specifications.

In order to store the previous 21 inputs, a ***Positive Edge-Triggered D-Flip-Flop*** has been implemented:

```
entity d_flip_flop is
    generic (
        Nbit : natural := 16
    );
    port (
        clk : in std_logic;
        resetn : in std_logic;
        enable : in std_logic;
        d   : in std_logic_vector(Nbit-1 downto 0);
        q   : out std_logic_vector(Nbit-1 downto 0)
    );
end entity d_flip_flop;
```

In order to perform additions, a ***Ripple Carry Adder*** has been realized, starting with a ***Full Adder***. This approach enables sequential bit-by-bit addition:

```
entity ripple_carry_adder is
    generic(
        NBit : natural := 16
    );
    port(
        a : in std_logic_vector(NBit-1 downto 0);
        b : in std_logic_vector(NBit-1 downto 0);
        c_in : in std_logic;
        sum : out std_logic_vector(NBit-1 downto 0);
        c_out : out std_logic
    );
end ripple_carry_adder;
```

Finally, to perform the multiplication between the sum result and the coefficient, a dedicated ***Multiplier*** has been employed.

```
entity multiplier is
    generic (
        NBit : natural := 16
    );
    port (
        a : in std_logic_vector(NBit-1 downto 0);
        b : in std_logic_vector(NBit-1 downto 0);
        result : out std_logic_vector((2*NBit)-1 downto 0)
    );
end entity multiplier;
```

3.1 Testbench and Simulation

Regarding the testbench, the reset and enable signals were initially tested. Subsequently, input values for `x`, both positive and negative, were tested to ensure the proper operation of the filter.

As seen in the waveform in Figure 4, the output `y` (in analog form) behaves like a raised cosine filter. However, precision is not optimal during rapid variations due to the approximations inherent in this type of filter.

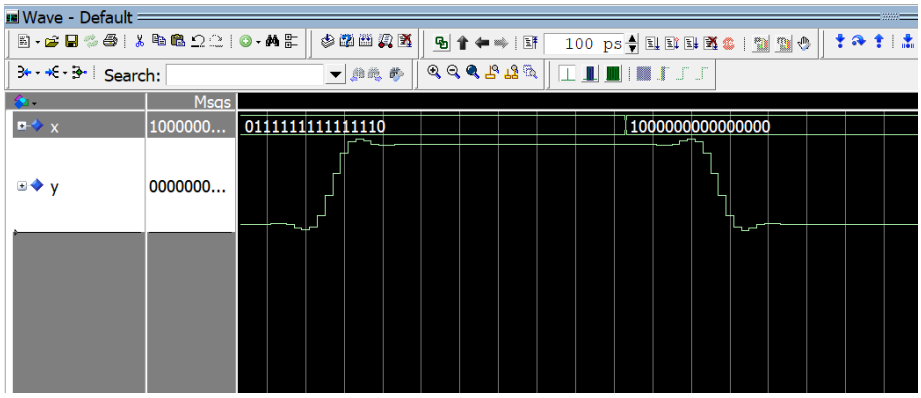

Figure 4: Wave

4 Vivado Logic Synthesis and Implementation

4.1 Synthesis and Implementation Summary

The first thing that was done was to perform **synthesis** through Vivado, introducing a timing constraint of 32 nanoseconds. This timing constraint corresponds to a frequency of 32.000 Hz, commonly used in digital transmissions.

The synthesis stage was completed without producing any errors or warnings as shown in Figure 5.

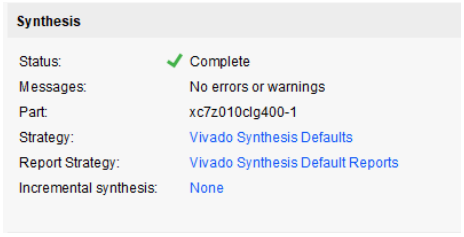

Figure 5: Synthesis Summary

At this point, the **implementation** was carried out, which also did not show any warnings or errors, as can be seen in Figure 6.

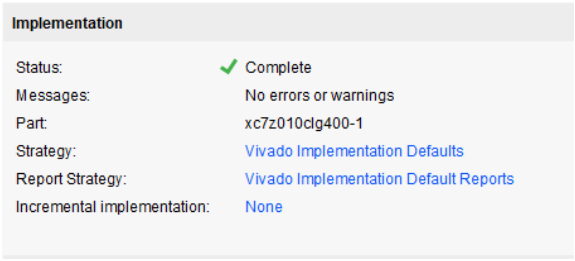

Figure 6: Implementation Summary

4.2 Timing

The timing summary in the implementation stage is as follows (Figure 7): In the summary, the

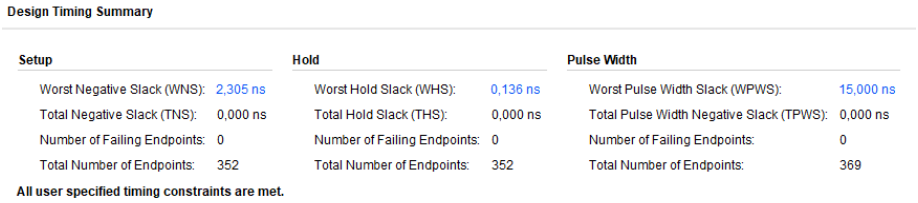

Figure 7: Timing Summary

following parameters can be found:

- **Worst Negative Slack**: indicates the worst negative timing discrepancy among signal paths in the circuit. A negative discrepancy indicates insufficient delay to meet timing requirements.
- **Total Negative Slack**: represents the overall sum of negative timing discrepancies across all signal paths in the circuit.
- **Worst Hold Slack**: measures the worst timing discrepancy concerning hold requirements among critical signal paths. It indicates whether the hold time between signals is sufficient.
- **Total Hold Slack**: is the total sum of hold timing discrepancies across all signal paths.
- **Worst Pulse Width Slack**: indicates the worst timing discrepancy in pulse width, crucial for periodic signals like clocks.

- **Total Worst Pulse Width**: represents the total sum of timing discrepancies in pulse width across all signal paths in the circuit.

In the case of the circuit in question, all these values are positive, so **there are no timing issues**.

We can calculate the **maximum usable frequency** as:

$$f_{max} = \frac{1}{(32 - 2.305) \cdot 10^{-9}} \approx 33.7 \ MHz$$

The **critical path** is the following (Figure 8):

Intra-Clock Paths - FIR_Clock - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	2.305	28	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[14]/D	28.516	8.391	20.125	31.0	FIR_Clock	FIR_Clock		0.035
Path 2	2.600	28	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[15]/D	28.269	8.391	19.878	31.0	FIR_Clock	FIR_Clock		0.035
Path 3	3.136	27	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[13]/D	27.691	8.267	19.424	31.0	FIR_Clock	FIR_Clock		0.035
Path 4	3.318	27	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[12]/D	27.507	9.564	17.943	31.0	FIR_Clock	FIR_Clock		0.035
Path 5	3.605	26	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[11]/D	27.220	8.143	19.077	31.0	FIR_Clock	FIR_Clock		0.035
Path 6	3.882	25	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[9]/D	26.868	8.019	18.849	31.0	FIR_Clock	FIR_Clock		0.035
Path 7	4.040	26	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[10]/D	26.786	9.012	17.774	31.0	FIR_Clock	FIR_Clock		0.035
Path 8	4.760	25	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[8]/D	26.068	8.019	18.049	31.0	FIR_Clock	FIR_Clock		0.035
Path 9	5.181	24	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[7]/D	25.644	7.895	17.749	31.0	FIR_Clock	FIR_Clock		0.035
Path 10	5.590	24	14	input_reg_gen[9...op_i/q_reg[2]/C	output_reg/q_reg[6]/D	25.283	7.895	17.388	31.0	FIR_Clock	FIR_Clock		0.035

Figure 8: Critical Path

4.3 Utilization

In Figure 9, we observe the circuit **utilization percentages**.

As shown, 35% of the utilization is attributed to the **I/O**. This is reasonable since both the input and output consist of 16 bits.

In the circuit, there is also a 15% utilization in the **DSP resources**, referring to the use of digital signal processing resources present on the FPGA. These DSP resources are dedicated to implementing mathematical operations and algorithms typical in digital signal processing applications, such as multiplications and additions. Given that the circuit is rich in these features, this percentage is considered reasonable.

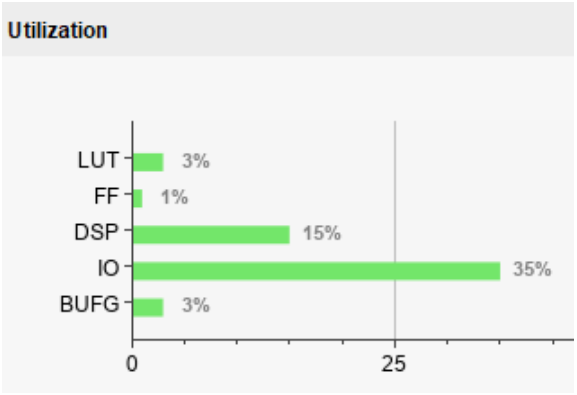

Figure 9: Utilization Summary

4.4 Power Consumption

Regarding power consumption, the result shown in Figure 10 has been obtained. It is noted that the total power consumed by the circuit is 0.125 W, with 26% attributable to dynamic power and the remaining 74% to static power.

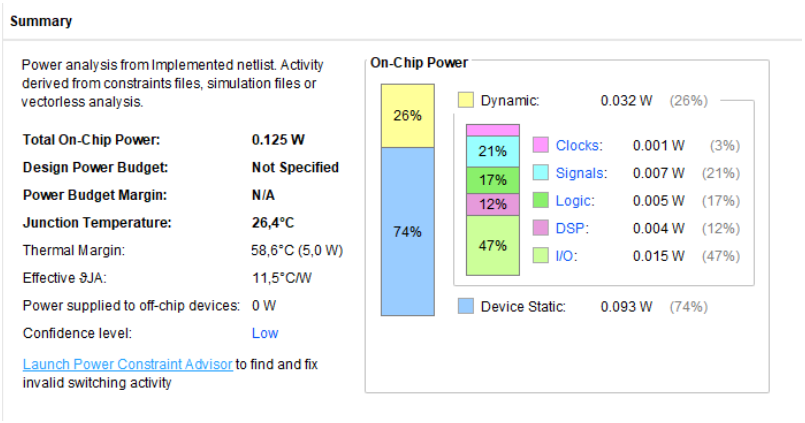

Figure 10: Power Summary

4.5 Area Occupation

In the Figure 11, the area occupied by the circuit is shown.

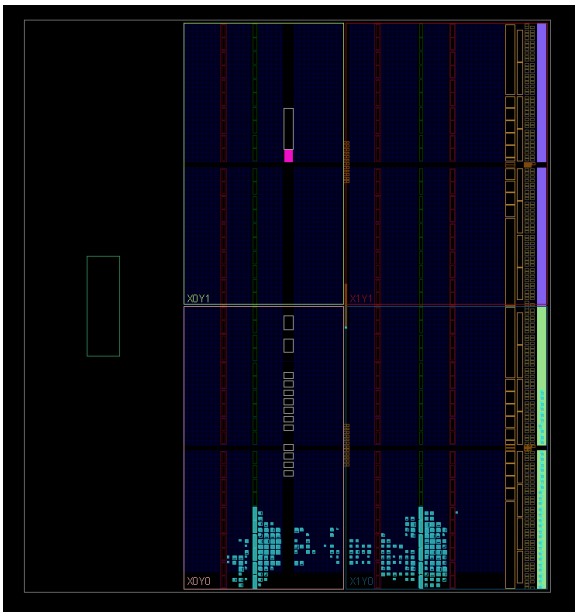

Figure 11: Area Occupation

4.6 Warnings and Errors

Finally, no warnings or error messages were detected throughout the synthesis and implementation phase, as shown in the Figure 12.

Figure 12: Synthesis and Implementations Error or Warning Messages

5 Conclusions

In conclusion, this project involved implementing an **FIR Filter SRRC**. This process began with simplifying the design and selecting a coefficient representation. Subsequently, it was translated into VHDL and concretely realized. Finally, the circuit's characteristics were analyzed through the Vivado software.

A potential **optimization** for the circuit, aimed at increasing the maximum usable frequency, could involve dividing the combinatorial network formed by adders and multipliers using D-Flip-Flops to create a pipeline. However, this would result in an increase in the delay between input modification and output update.