UNIVERSITÀ DI PISA

MSc in Computer Engineering

Electronics and Communication Systems

Calculation of fo Coefficient

STUDENT:

Fabrizio Lanzillo

https://github.com/FabrizioLanzillo/
Image-Elaboration-fo-Coefficient-Calculator

Academic Year 2023-2024

Contents

Chapter 1

Introduction

1.1 Specifications

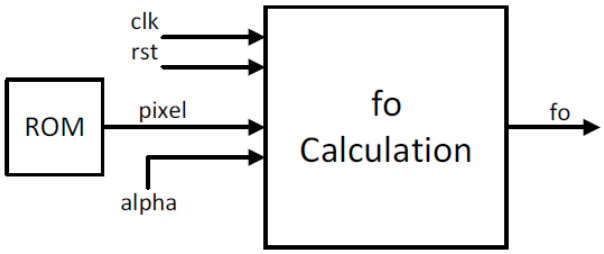
In an image elaboration system, it is necessary to calculate the fo coefficient:

$$f_o = \alpha \cdot y(i-1, j) + (1-\alpha) \cdot y(i, j)$$

where:

- $y(i-1, j)$ and $y(i, j)$ are pixels of the matrix y, which represents an image;
- α is a parameter chosen by the user. $\alpha \in (0, 1)$

It is required to design a digital circuit for implementing such operation. 10-bits are sufficient to represent α in 7.3 fixed-point arithmetic. The output f_0 is represented in N.3 fixed-point arithmetic, where N must be chosen accordingly to the dynamic of the output (to be studied). The interface of the circuit to be designed is as follows:



Pixels' values are read from a ROM containing all matrix elements. Only one pixel per cycle can be read from the ROM. The fo calculation circuit must autonomously read the pixels from memory to provide the output.

You are requested to deal with the various possible error situations, documenting the choices made. In particular, it is necessary to take into consideration:

- Pixel order in ROM
- Values of pixels and alpha out of specifications

The final project report must contain:

- Introduction (circuit description, possible applications, possible architectures, etc.)
- Description of the architecture designed (block diagram, inputs/outputs, etc.)
- VHDL code (with detailed comments) to be attached to the report.
- Test strategy (Test-plan) and related Testbench for verification; a detailed, though not exhaustive, verification is required, including error situations and borderline cases of functioning
- Interpretation of the results obtained in the automatic synthesis/implementation on a Xilinx FPGA platform in terms of maximum clock frequency (critical path), elements used (slice, LUT, etc.) and estimated power consumption. Comment on any warning messages.
- Conclusions

1.2 Possible Architectures

In the circuit design, the specification indicates to represent the α **parameter** with a decimal precision of 10 bits using **Fixed-Point arithmetic**.
Also, the **output** fo should be represented using the **fixed-point arithmetic** on $N.3$ **bits**, where the 3 represents the number of bits of the decimal part and N must be chosen accordingly to the dynamic of the output.

However, this specification has some criticisms. If α is represented with a decimal precision of 10 bits, the resulting output fo would require a representation on a number of bits at least equal to 10 to maintain the same precision.

To overcome these criticisms, it was decided to **represent** α **on 3 bits**, with **all 3 bits** used for the **decimal part**. This allowed to maximize the sensitivity of the decimal part given the 3-bit limitation of the fo output.

In the context of the digital circuit design, the representation of α required special attention. α is a decimal number that is represented using fixed point arithmetic over 3 bits.
From the physical point of view of the circuit, however, the **input is always a sequence of 3 bits**.
In order to make the calculation of fo easier α was converted from a decimal number represented by fixed-point arithmetic to a non-negative integer (unsigned).

This conversion process is critical because, in practice, **we are the ones giving meaning to the bit sequence by deciding where to place the decimal point**. In other words, what we are doing is mapping the decimal representation of α into a bit sequence that the circuit can handle.

Once α has been converted to a non-negative integer, its binary representation is used to compute fo so that the circuit produces a sequence of bits that, when interpreted with fixed-point arithmetic, matches the desired value of fo.

This step required **changing the formula of the fo coefficient in order to calculate fo as unsigned**.
In the **original formula**, α is multiplied by previous pixel ($y(i-1,j)$) and $(1-\alpha)$is multiplied by current pixel ($y(i,j)$).
In the **modified formula**, the value of α that multiplies previous pixel must be multiplied by 2^3, and so consequently the same thing must be done for the term $(1-\alpha)$ that multiplies pixel. So the new formula becomes:

$$f_{ous} = \alpha_{us} \cdot y(i-1,j) + (1 * 2^3 - \alpha_{us}) \cdot y(i,j)$$

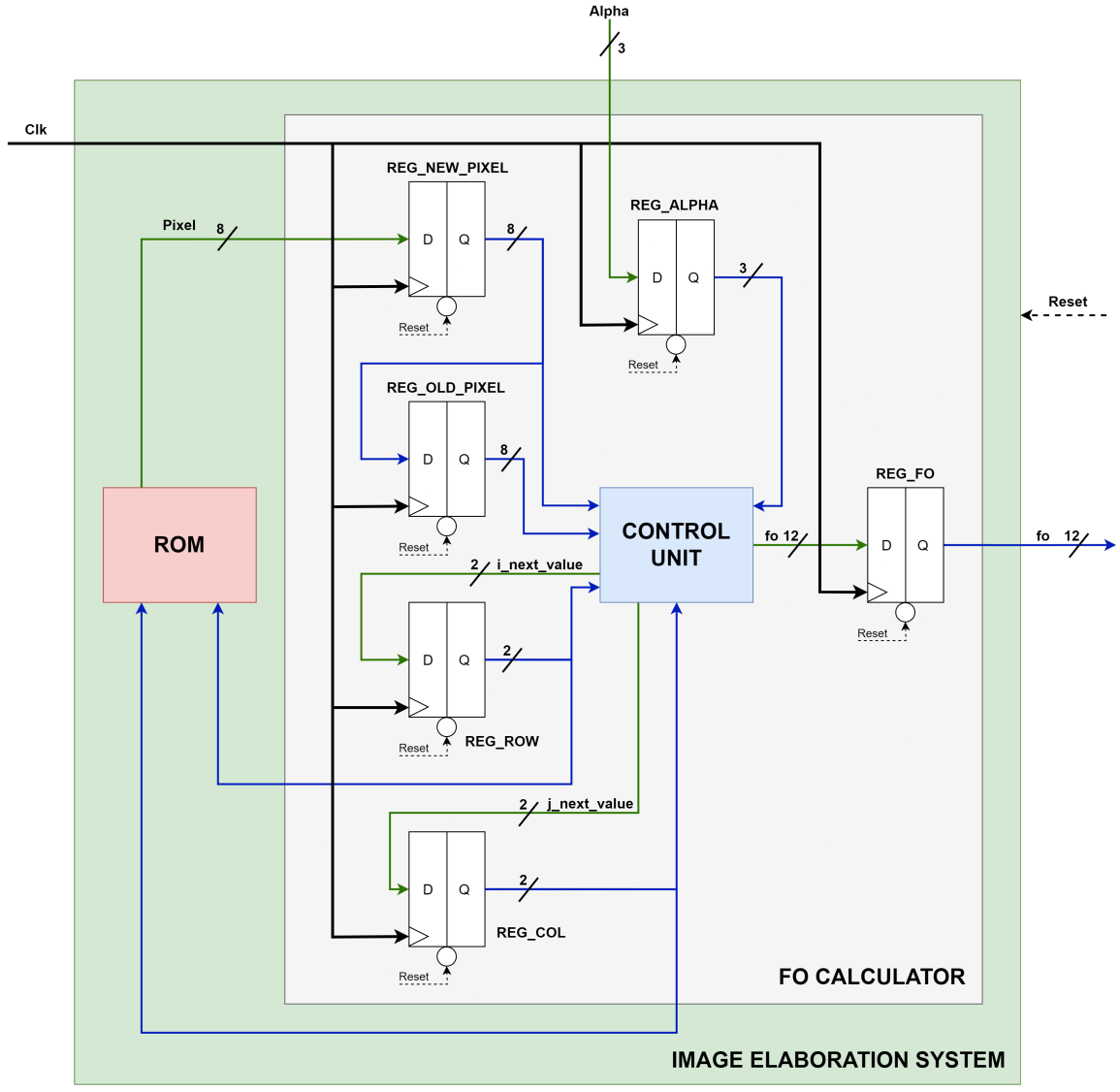
The number of bits of the f_{ous} output is evaluated according to the given formula, that involves a sum of multiplications of the same type of elements.
Having chosen to represent α on 3 bits, we have the multiplication of these 3 bits by the pixel value, which is on 8 bits, which brings the result of this multiplication on 11 bits.
Then we have the sum of two 11 bit values, which brings **the total number of bits of the f_{ous} output to 12**.

As mentioned previously, the specifications state that the f_{ous} output should be represented using fixed-point arithmetic on N.3 bits, where 3 represents the number of bits of the decimal part and N should be chosen according to the dynamics of the output.
So based on the previous calculation **N is then equal to 9**.

Chapter 2

Architecture Description

The implemented architecture is the following:



The **Image Elaboration System** is composed by 2 sub-components:

- the **ROM** component: That is used as a LUT in order to retrieve the pixels' values of the images;
- the **Fo Calculator** component: which is responsible for calculating the value of fo, taking as input the pixels obtained from the ROM.

The **Fo Calculator** component is further composed of:

- the **Control Unit** component: which implements the combinatory logic of calculating the coefficient fo;
- 6 **Registers**: the are input and output register in order to have a finite and optimized critical path.

In the detail of **Registers**, we have:

- 2 registers that hold the **value of indexes i and j**, which are used to read the pixels in the ROM;
- 2 registers that hold the **values of the pixel and the previous pixel**. To make sure that **only one pixel per cycle can be read from the ROM**, we **connected these two registers in cascade**;
- the register that holds the fo **output value** for one clock cycle;
- the register that holds the **alpha value** chosen by the user.

Chapter 3

VHDL Component implementation

3.1 ControlUnit.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  ------------------------------------------------------
6  -- Entity
7  ------------------------------------------------------
8
9  entity ControlUnit is
10     generic (
11         NBitAlpha :       natural := 3;
12         NBitPixelValue :  natural := 8;
13         NRow :            natural := 4;
14         NBitRow :         natural := 2;
15         NBitCol :         natural := 2;
16         NCol :            natural := 4;
17         -- The # of bits of the output is evaluated from the given formula
18         -- where we have the multiplication of the 3 bits of alpha
19         -- by the pixel value over 8 bits resulting in 11 bits
20         -- next we have the sum of the same quantity so the total bits become
               12
21         NbitFo :          natural := 12
22     );
23     port (
24         ------------------------ input --------------------------
25         alpha : in std_logic_vector(NBitAlpha-1 downto 0);
26         pixel : in std_logic_vector(NBitPixelValue-1 downto 0);
27         previous_pixel : in std_logic_vector(NBitPixelValue-1 downto 0);
28         i_current_value : in std_logic_vector(NBitRow-1 downto 0);
29         j_current_value : in std_logic_vector(NBitCol-1 downto 0);
30
31         ------------------------ output -------------------------
32         i_next_value : out std_logic_vector(NBitRow-1 downto 0);
33         j_next_value : out std_logic_vector(NBitCol-1 downto 0);
```

```
34          fo : out std_logic_vector(NbitFo-1 downto 0)
35      );
36  end entity ControlUnit;
37
38  ----------------------------------------------------------
39  -- Architecture
40  ----------------------------------------------------------
41
42  architecture behavioral of ControlUnit is
43      signal i_s : std_logic_vector(NBitRow-1 downto 0);
44      signal j_s : std_logic_vector(NBitCol-1 downto 0);
45      signal fo_s : std_logic_vector(NbitFo-1 downto 0);
46
47  begin
48      CU_PROC: process(i_current_value, j_current_value, alpha, pixel,
            previous_pixel)
49          begin
50
51              -- check if the index of the row is not greater than the number of
                rows
52              if(to_integer(unsigned(i_current_value)) <= (NRow-1)) then
53                  -- check if we are reading the pixel of the first row
54                  -- in this case we return as fo the zero value
55                  -- because there is not a previous pixel
56                  if(to_integer(unsigned(i_current_value)) = 0) then
57                      -- we assign fo the default value of zero
58                      fo_s <= (others => '0');
59                      -- increment of the row value
60                      i_s <= std_logic_vector(resize(unsigned(i_current_value) +1,
                            NBitRow));
61                      j_s <= j_current_value;
62                  else
63                      -- if we are in a row greater than zero we have the previous
                            pixel
64                      -- we can compute fo
65                      fo_s <= std_logic_vector(resize(unsigned(alpha) *
                            unsigned(previous_pixel) + (8 - unsigned(alpha)) *
                            unsigned(pixel), NbitFo));
66                      -- if we reached the final row
67                      if(to_integer(unsigned(i_current_value)) = (NRow-1)) then
68                          -- we reset the row index value
69                          i_s <= (others => '0');
70                          -- we check that the index of the column is not greater
                                than the number of columns
71                          if(to_integer(unsigned(j_current_value)) < (NCol-1)) then
72                              -- we pass to the next column, by incrementing the
                                    column index value
73                              j_s <=
                                    std_logic_vector(resize(unsigned(j_current_value)
                                    +1, NBitCol));
74                          else
```

```
75                          -- we reset the column index value
76                          j_s <= (others => '0');
77                      end if;
78                  else
79                      -- if we have not reached the final row we just
                            increment the
80                      -- row index value without change the column one
81                      i_s <= std_logic_vector(resize(unsigned(i_current_value)
                            +1, NBitRow));
82                      j_s <= j_current_value;
83                  end if;
84              end if;
85
86          else
87              fo_s <= (others => '0');
88              i_s <= (others => '0');
89              j_s <= (others => '0');
90          end if;
91
92      end process;
93
94      -- assignment to the outport value the signals
95      i_next_value <= i_s;
96      j_next_value <= j_s;
97      fo <= fo_s;
98  end architecture behavioral;
```

This component is the **core of the calculation of the fo coefficient**; it is also **responsible for the increment logic of the i and j indices** that are necessary **to retrieve the pixel from the ROM**.

3.2 ROM.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

------------------------------------------------------------
-- Entity
------------------------------------------------------------

entity ROM is
    generic (
        Npixel : natural := 4;
        NBitPixelValue: natural := 8;
        NBitRow : natural := 2;
        NBitCol : natural := 2
    );
    port (
        ---------------- input ----------------------
        -- i is the counter of the row of the matrix
        i : in std_logic_vector(NBitRow-1 downto 0);
        -- j is the counter of the row of the matrix
        j : in std_logic_vector(NBitCol-1 downto 0);

        ---------------- output ---------------------
        pixel : out std_logic_vector(NBitPixelValue-1 downto 0)
    );
end entity ROM;

------------------------------------------------------------
-- Architecture
------------------------------------------------------------

architecture dataflow of ROM is
    signal pixel_addr_int : integer range 0 to Npixel*Npixel-1;
    -- Type definition for the ROM array
    type rom_array_t is array (0 to Npixel*Npixel-1) of
        unsigned(NBitPixelValue-1 downto 0);
    -- Constant ROM array with predefined values, these are the pixel value
    -- who have been generated randomly
    constant rom : rom_array_t := (
        x"FF", x"AA", x"55", x"33", -- first column 4 values
        x"11", x"22", x"33", x"44", -- next column 4 values
        x"66", x"77", x"88", x"99", -- next column 4 values
        x"BB", x"CC", x"DD", x"EE" -- last column 4 values
    );
begin

    -- we use the row index and the column index to access the relative
        address of the pixel value
```

```
47      pixel_addr_int <= to_integer(unsigned(j)*to_unsigned(Npixel,3) +
            unsigned(i));
48      -- we use the address to index the pixel value in the ROM array
49      -- we assign it to the output port
50      pixel <= std_logic_vector(rom(pixel_addr_int));
51
52  end architecture;
```

As anticipated earlier the ROM is implemented as a LUT and the main elements are explained in the comments of the vhdl code.
The pixels' matrix of the image is generated randomly.

3.3 DFF_N.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.all;

------------------------------------------------------------
-- Entity
------------------------------------------------------------

entity DFF_N is
    generic (NBit : natural := 8);
    port (
        ---------------- input ---------------------
        clk     : in  std_logic;
    a_rst_n : in std_logic;
    en      : in  std_logic;
      D       : in  std_logic_vector(NBit - 1 downto 0);

        ---------------- output ---------------------
    Q       : out std_logic_vector(NBit - 1 downto 0)
    );
end DFF_N;

------------------------------------------------------------
-- Architecture
------------------------------------------------------------

architecture rtl of DFF_N is

begin
    -- flip-flop sequential logic, asyncronous reset
    DFF_N_PROC: process(clk, a_rst_n)
        begin
            if(a_rst_n = '0') then
                Q <= (others => '0');
            elsif(rising_edge(clk)) then
                if(en = '1') then
                    Q <= D;
                end if;
            end if;
        end process;
end rtl;
```

3.4 FoCalculator.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4  use IEEE.NUMERIC_STD.all;
5
6  ----------------------------------------------------------
7  -- Entity
8  ----------------------------------------------------------
9
10 entity FoCalculator is
11     generic (
12         NBitAlpha :    natural := 3;
13         NBitPixelValue: natural := 8;
14         NRow :         natural := 4;
15         NBitRow :      natural := 2;
16         NBitCol :      natural := 2;
17         NCol :         natural := 4;
18         NBitFo :       natural := 12
19     );
20     port (
21         ---------------- input ----------------------
22         clk            : in  std_logic;
23         a_rst_n        : in  std_logic;
24       pixel          : in std_logic_vector(7 downto 0);
25         alpha          : in std_logic_vector(NBitAlpha-1 downto 0);
26
27         ---------------- output ---------------------
28         i_next_value   : out std_logic_vector(NBitRow-1 downto 0);
29         j_next_value   : out std_logic_vector(NBitCol-1 downto 0);
30         fo : out std_logic_vector(NBitFo-1 downto 0)
31     );
32 end FoCalculator;
33
34 ----------------------------------------------------------
35 -- Architecture
36 ----------------------------------------------------------
37
38 architecture rtl of FoCalculator is
39     ----------------------------------------------------------
40     -- Signals
41     ----------------------------------------------------------
42     ---------------- input ----------------------
43     signal alpha_in_ext : std_logic_vector(NBitAlpha-1 downto 0);
44     signal pixel_in_ext : std_logic_vector(NBitPixelValue-1 downto 0);
45     signal previous_pixel_in_ext : std_logic_vector(NBitPixelValue-1 downto 0);
46     signal i_current_value_in_ext : std_logic_vector(NBitRow-1 downto 0);
47     signal j_current_value_in_ext : std_logic_vector(NBitCol-1 downto 0);
48
49     ---------------- output ---------------------
```

```
50      signal i_next_value_out_ext : std_logic_vector(NBitRow-1 downto 0);
51      signal j_next_value_out_ext : std_logic_vector(NBitCol-1 downto 0);
52      signal fo_out_ext : std_logic_vector(11 downto 0);
53
54      ------------------------------------------------------------
55      -- Costants
56      ------------------------------------------------------------
57      constant one   : std_logic := '1';
58
59      ------------------------------------------------------------
60      -- DFF_N Component
61      ------------------------------------------------------------
62
63      component DFF_N
64          generic (NBit : positive := 8);
65          port (
66              clk    : in  std_logic;
67              a_rst_n : in std_logic;
68              en     : in  std_logic;
69              D      : in  std_logic_vector(NBit - 1 downto 0);
70              Q      : out std_logic_vector(NBit - 1 downto 0)
71          );
72      end component;
73
74      ------------------------------------------------------------
75      -- ControlUnit Component
76      ------------------------------------------------------------
77      component ControlUnit
78          generic (
79              NBitAlpha : natural := 3;
80              NBitPixelValue : natural := 8;
81              NRow : natural := 4;
82              NBitRow : natural := 2;
83              NBitCol : natural := 2;
84              NCol : natural := 4;
85              NbitFo : natural := 12
86          );
87          port (
88              ---------------- input ---------------------
89              alpha : in std_logic_vector(NBitAlpha-1 downto 0);
90              pixel : in std_logic_vector(NBitPixelValue-1 downto 0);
91              previous_pixel : in std_logic_vector(NBitPixelValue-1 downto 0);
92              i_current_value : in std_logic_vector(NBitRow-1 downto 0);
93              j_current_value : in std_logic_vector(NBitCol-1 downto 0);
94
95              ---------------- output --------------------
96              i_next_value : out std_logic_vector(NBitRow-1 downto 0);
97              j_next_value : out std_logic_vector(NBitCol-1 downto 0);
98              fo : out std_logic_vector(NbitFo-1 downto 0)
99          );
100     end component;
```

```
101
102 begin
103
104     -- Register for the new pixel
105     REG_NEW_PIXEL: DFF_N
106         generic map (NBit => NBitPixelValue)
107         port map(
108             clk    => clk,
109             a_rst_n => a_rst_n,
110             en     => one,
111             D      => pixel,
112             Q      => pixel_in_ext
113         );
114
115     -- Register for the old pixel
116     REG_OLD_PIXEL: DFF_N
117         generic map (NBit => NBitPixelValue)
118         port map(
119             clk    => clk,
120             a_rst_n => a_rst_n,
121             en     => one,
122             D      => pixel_in_ext,
123             Q      => previous_pixel_in_ext
124         );
125
126     -- Register for the row index value
127     REG_ROW: DFF_N
128         generic map (NBit => NBitRow)
129         port map(
130             clk    => clk,
131             a_rst_n => a_rst_n,
132             en     => one,
133             D      => i_next_value_out_ext,
134             Q      => i_current_value_in_ext
135         );
136
137     -- Register for the column index value
138     REG_COL: DFF_N
139         generic map (NBit => NbitCol)
140         port map(
141             clk    => clk,
142             a_rst_n => a_rst_n,
143             en     => one,
144             D      => j_next_value_out_ext,
145             Q      => j_current_value_in_ext
146         );
147
148     -- Register for the alpha value
149     REG_ALPHA: DFF_N
150         generic map (NBit => NBitAlpha)
151         port map(
```

```
            clk    => clk,
            a_rst_n => a_rst_n,
            en     => one,
            D      => alpha,
            Q      => alpha_in_ext
        );

    -- Register for the fo output value
    REG_FO: DFF_N
    generic map (NBit => NBitFo)
    port map(
        clk    => clk,
        a_rst_n => a_rst_n,
        en     => one,
        D      => fo_out_ext,
        Q      => fo
    );

    -- Control Unit component
    CONTROL_UNIT: ControlUnit
        generic map (
            NBitAlpha => NBitAlpha,
            NBitPixelValue => NBitPixelValue,
            NRow => NRow,
            NBitRow => NBitRow,
            NBitCol => NBitCol,
            NCol => NCol,
            NbitFo => NbitFo
        )
        port map(
            alpha => alpha_in_ext,
            pixel => pixel_in_ext,
            previous_pixel => previous_pixel_in_ext,
            i_current_value => i_current_value_in_ext,
            j_current_value => j_current_value_in_ext,

            i_next_value => i_next_value_out_ext,
            j_next_value => j_next_value_out_ext,
            fo => fo_out_ext
        );

    -- assignment of the output port that are connected with the rom
    i_next_value <= i_current_value_in_ext;
    j_next_value <= j_current_value_in_ext;
end rtl;
```

This component contains the control unit and the registers for the various inputs and outputs of the control unit.

3.5 ImageElaborationSystem.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4  use IEEE.numeric_std.all;
5
6  ----------------------------------------------------------
7  -- Entity
8  ----------------------------------------------------------
9  entity ImageElaborationSystem is
10     generic (
11         NBitAlpha :    natural := 3;
12         NBitFo :       natural := 12;
13         NBitPixelValue: natural := 8;
14         NRow :         natural := 4;
15         NBitRow :      natural := 2;
16         NBitCol :      natural := 2;
17         NCol :         natural := 4;
18         Npixel :       natural := 4
19     );
20     port (
21         ---------------- input ----------------------
22         clk        : in  std_logic;
23         a_rst_n    : in  std_logic;
24         alpha      : in std_logic_vector(NBitAlpha-1 downto 0);
25
26         ---------------- output ---------------------
27         fo : out std_logic_vector(NBitFo-1 downto 0)
28     );
29 end ImageElaborationSystem;
30
31 ----------------------------------------------------------
32 -- Architecture
33 ----------------------------------------------------------
34
35 architecture rtl of ImageElaborationSystem is
36
37     ----------------------------------------------------------
38     -- FoCalculator Component
39     ----------------------------------------------------------
40     component FoCalculator
41         generic (
42             NBitAlpha :    natural := 3;
43             NBitPixelValue: natural := 8;
44             NRow :         natural := 4;
45             NBitRow :      natural := 2;
46             NBitCol :      natural := 2;
47             NCol :         natural := 4;
48             NBitFo :       natural := 12
49         );
```

```
50          port (
51              ---------------- input ----------------------
52              clk        : in  std_logic;
53              a_rst_n    : in  std_logic;
54              pixel      : in std_logic_vector(7 downto 0);
55              alpha      : in std_logic_vector(NBitAlpha-1 downto 0);
56
57              ---------------- output ---------------------
58              i_next_value : out std_logic_vector(NBitRow-1 downto 0);
59              j_next_value : out std_logic_vector(NBitCol-1 downto 0);
60              fo : out std_logic_vector(NBitFo-1 downto 0)
61          );
62      end component;
63
64      ------------------------------------------------------
65      -- ROM Component
66      ------------------------------------------------------
67      component ROM
68          generic (
69              Npixel :       natural := 4;
70              NBitPixelValue: natural := 8;
71              NBitRow :      natural := 2;
72              NBitCol :      natural := 2
73          );
74          port (
75              ---------------- input ----------------------
76              -- i is the counter of the row of the matrix
77              i : in std_logic_vector(NBitRow-1 downto 0);
78              -- j is the counter of the row of the matrix
79              j : in std_logic_vector(NBitCol-1 downto 0);
80
81              ---------------- output ---------------------
82              pixel : out std_logic_vector(NBitPixelValue-1 downto 0)
83          );
84      end component;
85
86      ------------------------------------------------------
87      -- Signals
88      ------------------------------------------------------
89      signal pixel_ext : std_logic_vector(7 downto 0);
90      signal i_next_value_ext : std_logic_vector(NBitRow-1 downto 0);
91      signal j_next_value_ext : std_logic_vector(NBitCol-1 downto 0);
92
93      begin
94
95          -- Fo Calculator component
96          FO_CALCULATOR: FoCalculator
97          generic map (
98              NBitAlpha => NBitAlpha,
99              NBitPixelValue => NBitPixelValue,
100             NRow => NRow,
```

```
            NBitRow => NBitRow,
            NBitCol => NBitCol,
            NCol => NCol,
            NBitFo => NBitFo
        )
        port map(
            clk => clk,
            a_rst_n => a_rst_n,
            pixel => pixel_ext,
            alpha => alpha,

            i_next_value => i_next_value_ext,
            j_next_value => j_next_value_ext,
            fo => fo
        );

        -- ROM component
        C_ROM: ROM
        generic map (
            Npixel => Npixel,
            NBitPixelValue => NBitPixelValue,
            NBitRow => NBitRow,
            NBitCol => NBitCol
        )
        port map(
            i => i_next_value_ext,
            j => j_next_value_ext,

            pixel => pixel_ext
        );

end rtl;
```

Finally, the ImageElaborationSystem entity contains the ROM and Fo Calculator and represents the system in its totality, it only receives as input the alpha value, the clk, and the reset providing as output the value of fo.

Chapter 4

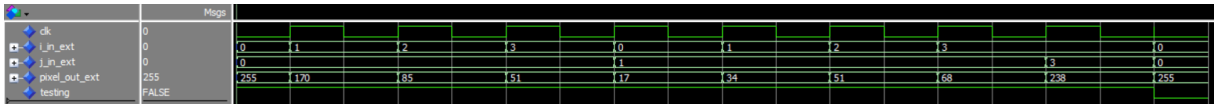
Test-plan and Testbenches

In order to **verify the correct behavior** of the final device, several **testbenches were conducted**.
In particular, we performed **incremental testbenches**, starting with the base entities such as the **Control Unit** and **ROM**, then testing the **Fo Calculator** and finally the **Image Elaboration System** to verify the system in its entirety.
Modelsim was used to perform the simulation of the various components.

In the following paragraphs we will analyze each of these testbenches.

4.1 ROM Testbench

The ROM testbench consists of creating a stimuli process, where different row and column indices are given as input to the component and through the Modelsim simulation wave it is verified that for each of them, the pixel returned from the ROM is exactly what is indicated by the specified position in the matrix.
Here in the image we can see the Modelsim simulation wave:



If we analyze the wave we see that the various output pixels, given the two input indices, are correct, in fact if we compare them with those in the matrix in the ROM vhdl file we see that we have a match for each of them.

4.2 Control Unit Testbench

In order to verify the correct behavior of this component, several tools and scripts were used.
In particular, a **python script was used to simulate the high-level behavior of the control unit component**.
The **data obtained from this script** were used in order **to validate the results obtained from the Modelsim simulation** and verify that **they were consistent and correct**.

This is the python script:
validate_control_uniti_tb_results.py

```
import os
import numpy as np

def compute_fo_decimal(alpha, previous_pixel, pixel):
    if alpha != 0:
        return alpha * previous_pixel + (1 - alpha) * pixel
    else:
        return 0

def compute_fo_fixed_point_arithmetic(alpha, previous_pixel, pixel):
    if alpha != 0:
        return alpha * previous_pixel + (8 - alpha) * pixel
    else:
        return 0

def convert_decimal_to_fixed_point(decimal_number, bit_length):
    return int(decimal_number * (2**bit_length))

def main():
    file_path = os.path.join('..', 'conf_files', 'control_unit_input.txt')
    with open(file_path, 'r') as f:
        lines = f.readlines()
        alpha_decimal = float(lines[0].strip())
        alpha_fixed_point = convert_decimal_to_fixed_point(alpha_decimal, 3)
        NCol = int(lines[1].strip())
        NRow = int(lines[2].strip())
        matrix = np.array([int(x) for x in lines[3].split(',')]).reshape(NCol,
            NRow).T
        print("-----------------------------------------------------")
        print("[Conf]")
        print(f"\talpha_decimal = {alpha_decimal};")
        print(f"\talpha_fixed_point = {alpha_fixed_point};")
        print("-----------------------------------------------------")
        for j in range(NCol):
            for i in range(NRow):
                pixel = matrix[i][j]
                previous_pixel = matrix[i-1][j] if i > 0 else 0
                if i == 0:
                    fo_fixed_point = 0
                    fo_decimal = 0
                else:
                    fo_fixed_point =
                        compute_fo_fixed_point_arithmetic(alpha_fixed_point,
                        previous_pixel, pixel)
                    fo_decimal = compute_fo_decimal(alpha_decimal,
                        previous_pixel, pixel)
                print("[Output Control Unit]")
                print("IN:")
                print(f"\ti_current_value = {i}")
```

```
                print(f"\tj_current_value = {j}")
                print(f"\tprevious_pixel = {previous_pixel}")
                print(f"\tpixel = {pixel}")
                print("OUT:")
                print(f"\tfo_decimal = {fo_decimal}")
                print(f"\tfo_fixed_point = {fo_fixed_point}")
                print(f"\tfo_decimal_converted_to_fixed_point =
                    {convert_decimal_to_fixed_point(fo_decimal, 3)}")
                print(f"\ti_next_value_out_ext = {(i + 1) % NRow}")
                print(f"\tj_next_value_out_ext = {(j + 1) % NCol if i + 1 ==
                    NRow else j}")
                print("-------------------------------------------------")

if __name__ == "__main__":
    main()
```

Through a configuration file, the matrix, the alpha value, and the number of rows and columns in the matrix are given to this script as input.
For example, if we provide as input to this script the matrix contained in the ROM, and an alpha value that corresponds to 0.25, we get this output:

```
-------------------------------------------------
[Conf]
  alpha_decimal = 0.25;
  alpha_fixed_point = 2;
-------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 0
  j_current_value = 0
  previous_pixel = 0
  pixel = 255
OUT:
  fo_decimal = 0
  fo_fixed_point = 0
  fo_decimal_converted_to_fixed_point = 0
  i_next_value_out_ext = 1
  j_next_value_out_ext = 0
-------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 1
  j_current_value = 0
  previous_pixel = 255
  pixel = 170
OUT:
  fo_decimal = 191.25
  fo_fixed_point = 1530
  fo_decimal_converted_to_fixed_point = 1530
  i_next_value_out_ext = 2
```

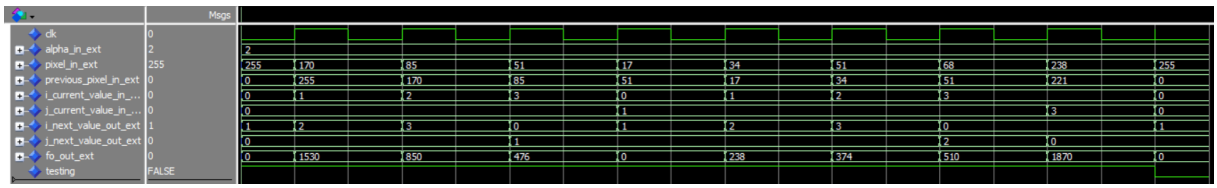
```
  j_next_value_out_ext = 0
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 2
  j_current_value = 0
  previous_pixel = 170
  pixel = 85
OUT:
  fo_decimal = 106.25
  fo_fixed_point = 850
  fo_decimal_converted_to_fixed_point = 850
  i_next_value_out_ext = 3
  j_next_value_out_ext = 0
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 3
  j_current_value = 0
  previous_pixel = 85
  pixel = 51
OUT:
  fo_decimal = 59.5
  fo_fixed_point = 476
  fo_decimal_converted_to_fixed_point = 476
  i_next_value_out_ext = 0
  j_next_value_out_ext = 1
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 0
  j_current_value = 1
  previous_pixel = 0
  pixel = 17
OUT:
  fo_decimal = 0
  fo_fixed_point = 0
  fo_decimal_converted_to_fixed_point = 0
  i_next_value_out_ext = 1
  j_next_value_out_ext = 1
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 1
  j_current_value = 1
  previous_pixel = 17
  pixel = 34
OUT:
  fo_decimal = 29.75
  fo_fixed_point = 238
  fo_decimal_converted_to_fixed_point = 238
```

```
  i_next_value_out_ext = 2
  j_next_value_out_ext = 1
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 2
  j_current_value = 1
  previous_pixel = 34
  pixel = 51
OUT:
  fo_decimal = 46.75
  fo_fixed_point = 374
  fo_decimal_converted_to_fixed_point = 374
  i_next_value_out_ext = 3
  j_next_value_out_ext = 1
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 3
  j_current_value = 1
  previous_pixel = 51
  pixel = 68
OUT:
  fo_decimal = 63.75
  fo_fixed_point = 510
  fo_decimal_converted_to_fixed_point = 510
  i_next_value_out_ext = 0
  j_next_value_out_ext = 2
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 0
  j_current_value = 2
  previous_pixel = 0
  pixel = 102
OUT:
  fo_decimal = 0
  fo_fixed_point = 0
  fo_decimal_converted_to_fixed_point = 0
  i_next_value_out_ext = 1
  j_next_value_out_ext = 2
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 1
  j_current_value = 2
  previous_pixel = 102
  pixel = 119
OUT:
  fo_decimal = 114.75
  fo_fixed_point = 918
```

```
  fo_decimal_converted_to_fixed_point = 918
  i_next_value_out_ext = 2
  j_next_value_out_ext = 2
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 2
  j_current_value = 2
  previous_pixel = 119
  pixel = 136
OUT:
  fo_decimal = 131.75
  fo_fixed_point = 1054
  fo_decimal_converted_to_fixed_point = 1054
  i_next_value_out_ext = 3
  j_next_value_out_ext = 2
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 3
  j_current_value = 2
  previous_pixel = 136
  pixel = 153
OUT:
  fo_decimal = 148.75
  fo_fixed_point = 1190
  fo_decimal_converted_to_fixed_point = 1190
  i_next_value_out_ext = 0
  j_next_value_out_ext = 3
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 0
  j_current_value = 3
  previous_pixel = 0
  pixel = 187
OUT:
  fo_decimal = 0
  fo_fixed_point = 0
  fo_decimal_converted_to_fixed_point = 0
  i_next_value_out_ext = 1
  j_next_value_out_ext = 3
--------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 1
  j_current_value = 3
  previous_pixel = 187
  pixel = 204
OUT:
  fo_decimal = 199.75
```

```
  fo_fixed_point = 1598
  fo_decimal_converted_to_fixed_point = 1598
  i_next_value_out_ext = 2
  j_next_value_out_ext = 3
-------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 2
  j_current_value = 3
  previous_pixel = 204
  pixel = 221
OUT:
  fo_decimal = 216.75
  fo_fixed_point = 1734
  fo_decimal_converted_to_fixed_point = 1734
  i_next_value_out_ext = 3
  j_next_value_out_ext = 3
-------------------------------------------------
[Output Control Unit]
IN:
  i_current_value = 3
  j_current_value = 3
  previous_pixel = 221
  pixel = 238
OUT:
  fo_decimal = 233.75
  fo_fixed_point = 1870
  fo_decimal_converted_to_fixed_point = 1870
  i_next_value_out_ext = 0
  j_next_value_out_ext = 0
-------------------------------------------------
```

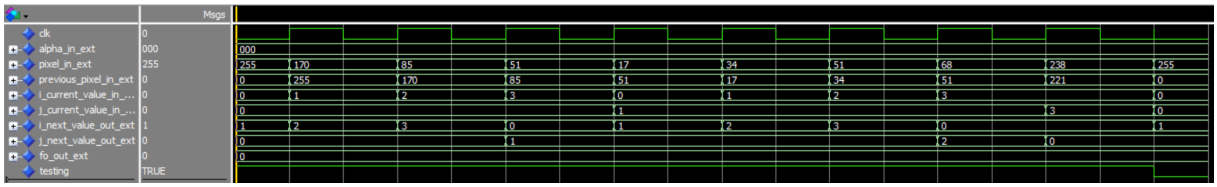
If we compare the results contained in the script output with those obtained from the wave of the testbench simulation on Modelsim (which we can see in the image) we see that the **results are consistent**.

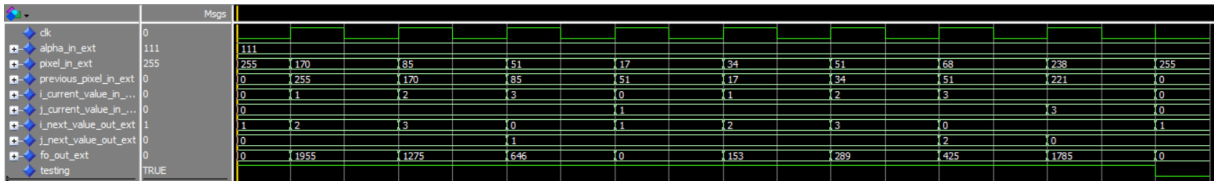


In the figure of the wave we do not see all the possible outputs because the size of the image did not allow to contain all of them, this is also true for the other testbenches, however, all the special cases i.e. the initial and the final case were included.

In order to verify that the system behaves correctly, even when the inputs are varied, **other tests were also performed where we vary the value of α in some limit cases**, i.e. when it is 0 or when it is 0.875, that in binary is 111.

Here we can see in the first image the wave with $\alpha = 0$ and in the second one the one with $\alpha = 0.875$



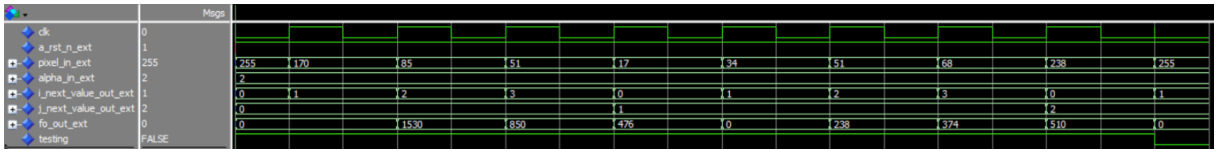


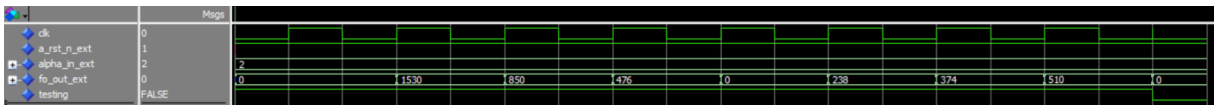
Also here the results were compared with those of the script, the outputs can be found in the attached files.

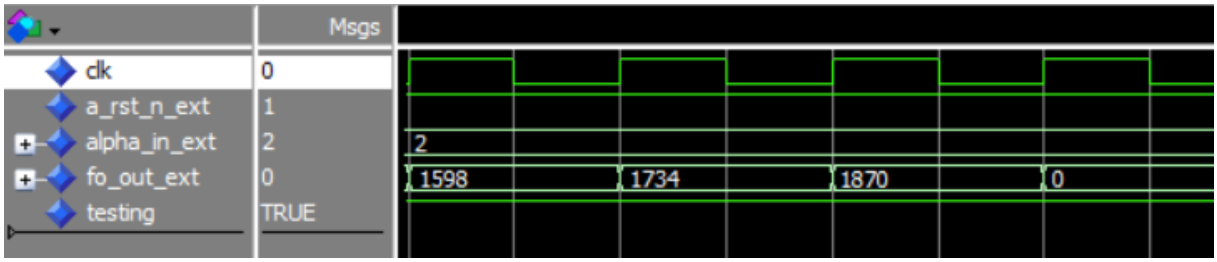
4.3 Fo Testbench and Image Elaboration System Testbench

To test these two entities, the **corresponding testbenches were simulated on Modelsim** and the results were compared with those of the script. Both tests were conducted with the α value at 0.25.

In the following images we can see first the wave of Fo Testbench and then the wave of System Testbench (2 images).





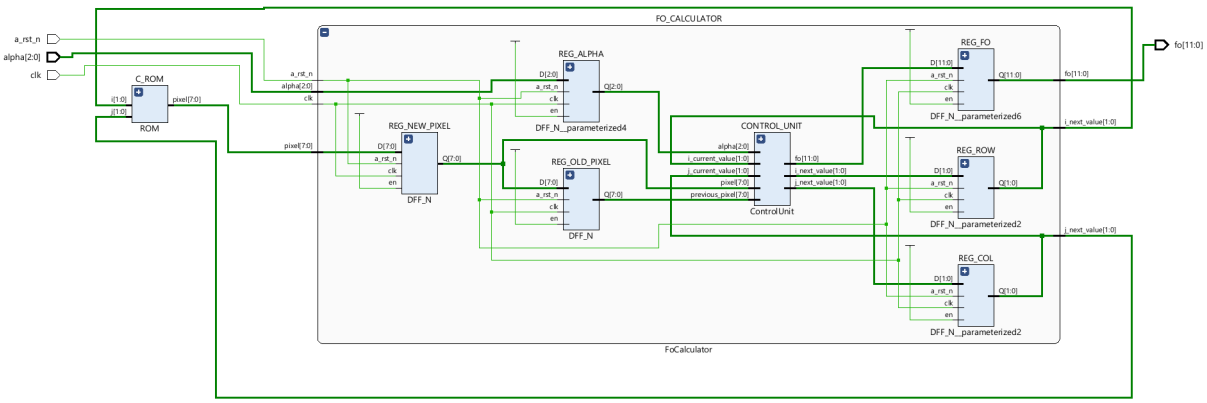


For these two testbenches it is **important to notice**, as we can see from the waves, that **the ouput is shifted by one clock cycle with respect to the inputs**, this is **due to the presence of the registers that maintain the result stable for one clock cycle**.

Chapter 5

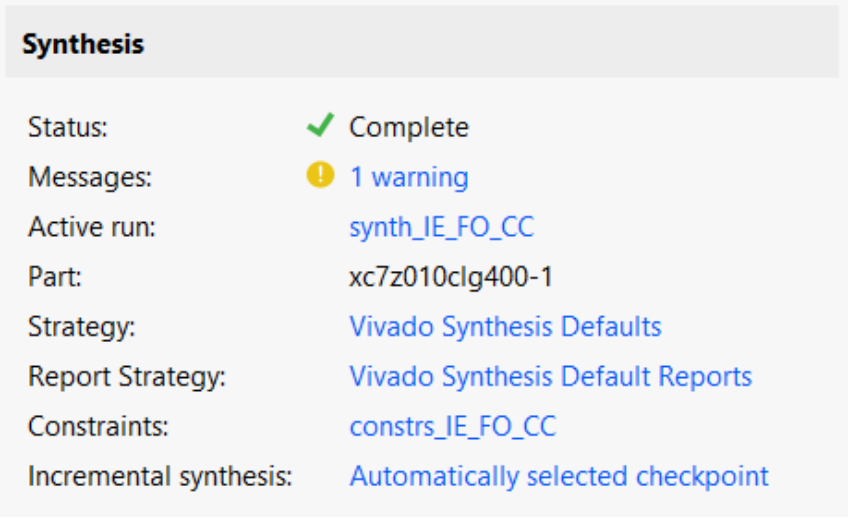
Synthesis and Implementation

The next step is to use the **Vivado Synthesis Tool** to perform code **synthesis** and **implementation**.
Before proceeding with the synthesis, an additional check was made on the correctness of the system.
This is done by comparing the **schema** obtained through **Vivado's RTL analysis**, with the original scheme, the Vivado schema is the following:

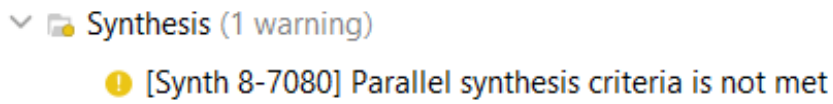


Everything is like expected so we can proceed through the synthesis.
The **results that will be shown** in this chapter are **obtained** with the **clock constraint** and in **Out-of-Context mode**.

5.1 Synthesis Result



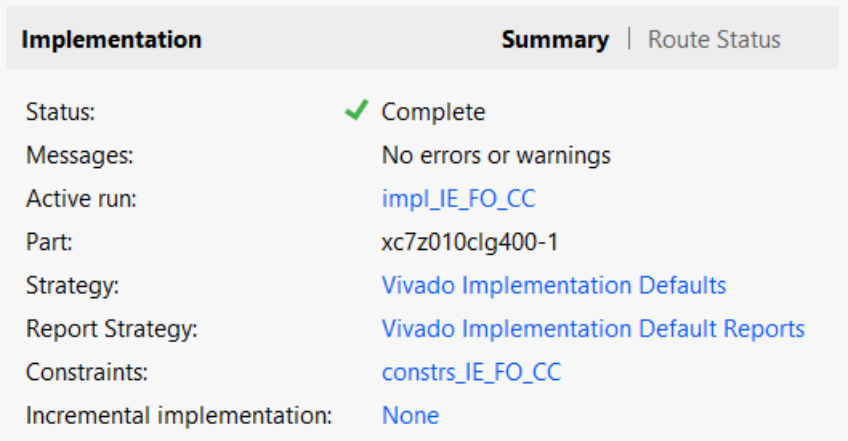
After the synthesis Vivado shows a warning, as we can see from the picture that represent the synthesis results.
In particular this is the warning:



After conducting research, it appears that this warning may occur with certain host machine configuration options and it does not depend on the device under consideration, provided to the synthesis.
After the synthesis I can go through the implementation to see the results.

5.2 Implementation Results and Timings

After the execution of the implementation there were no errors or warnings:



The implementation has been executed with the following **constraint**:

```
create_clock -period 8.000 -name IE_FO_CC_clk -waveform {0.000 4.000} -add
    [get_ports -filter { NAME =~ "*clk*" && DIRECTION == "IN" }]
```

Which sets the clock frequency to $f_{clock} = \frac{1}{8_{ns}} = 125MHz$.

This frequency was chosen because it is the **frequency at which the Zybo board operates**.
The implementation with this clock constraint returns the following timings results:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1,634 ns	Worst Hold Slack (WHS):	0,215 ns	Worst Pulse Width Slack (WPWS):	3,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	23	Total Number of Endpoints:	23	Total Number of Endpoints:	26

All user specified timing constraints are met.

As we can see from the image the **Worst Negative Slack (WNS) is positive**, this means that we can **run our implementation at a maximum frequency**, calculated as:

$f_{max} = \frac{1}{T_{clock} - WSN} = 157,1MHz$, with $T_{clock} = 8_{ns}$, the period of the clock.

The WSN corresponds to the lowest slack among the slacks of all the critical paths.

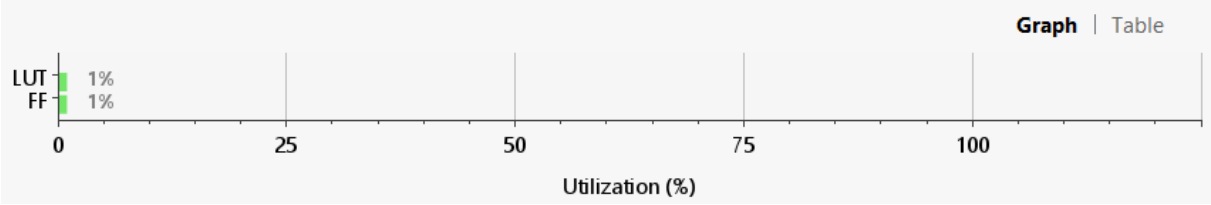
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	1.634	7	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[9]/D	6.357	3.126	3.231
Path 2	2.064	7	4	13	FO_CALCULATOR/REG..._PIXEL/Q_reg[6]/C	FO_CALCULATOR/REG_FO/Q_reg[10]/D	5.881	2.966	2.915
Path 3	2.081	7	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[8]/D	5.910	3.009	2.901
Path 4	2.184	6	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[7]/D	5.764	2.859	2.905
Path 5	2.342	6	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[6]/D	5.649	2.791	2.858
Path 6	2.576	6	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[5]/D	5.371	2.660	2.711
Path 7	2.710	6	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[4]/D	5.281	2.570	2.711
Path 8	2.960	5	3	13	FO_CALCULATOR/REG..._PIXEL/Q_reg[4]/C	FO_CALCULATOR/REG_FO/Q_reg[2]/D	5.074	2.283	2.791
Path 9	3.212	5	4	16	FO_CALCULATOR/REG..._PIXEL/Q_reg[5]/C	FO_CALCULATOR/REG_FO/Q_reg[3]/D	4.735	2.176	2.559
Path 10	3.499	5	3	13	FO_CALCULATOR/REG..._PIXEL/Q_reg[4]/C	FO_CALCULATOR/REG_FO/Q_reg[1]/D	4.496	1.905	2.591

We can see from the image, that the path from REG_NEW_PIXEL to REG_FO in the FO_CALCULATOR component is **the one that has the greatest impact on the delay and on the clock frequency**.

5.3 Resource Utilization

The resource utilization by this architecture is the following:

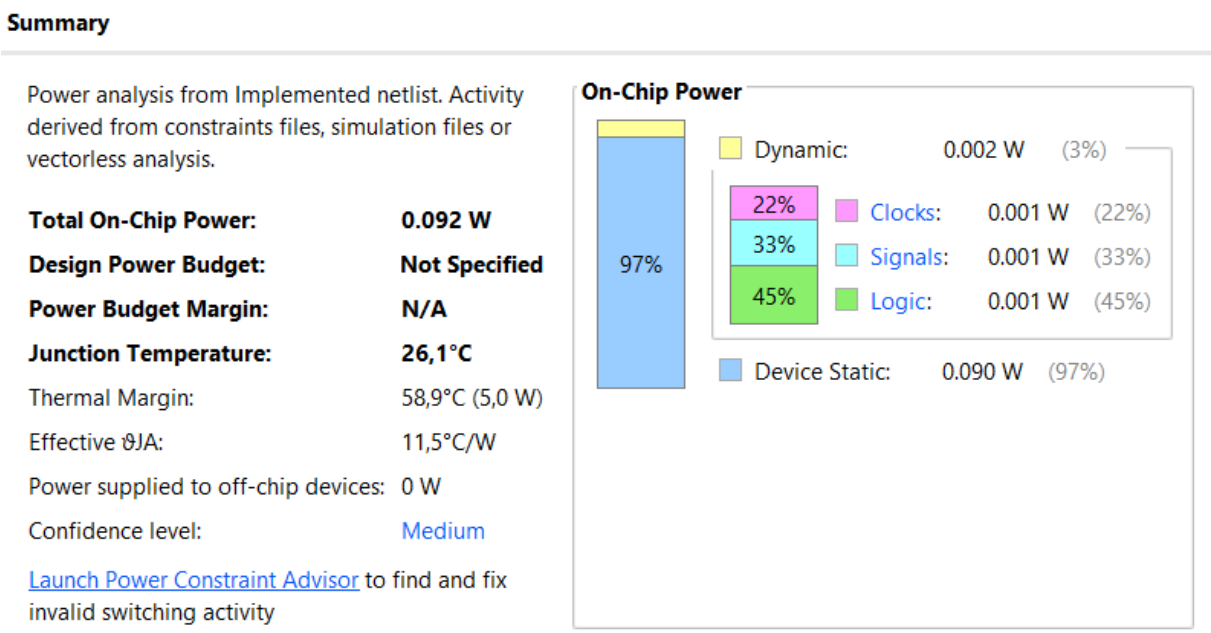
Resource	Utilization	Available	Utilization %
LUT	60	17600	0.34
FF	26	35200	0.07



As might be expected, the **resources used are rather low compared to those available**, due to the simplicity of the operations the device has to perform, especially for LUTs and Flip-Flops, which are 1% or less of the total.
This architecture is then feasible on the **Zybo Zynq 7000 board**.
I/O usage is **not specified** because of the **Out-Of-Context Mode**.

5.4 Power Consumption

The power consumption of this architecture is the following:

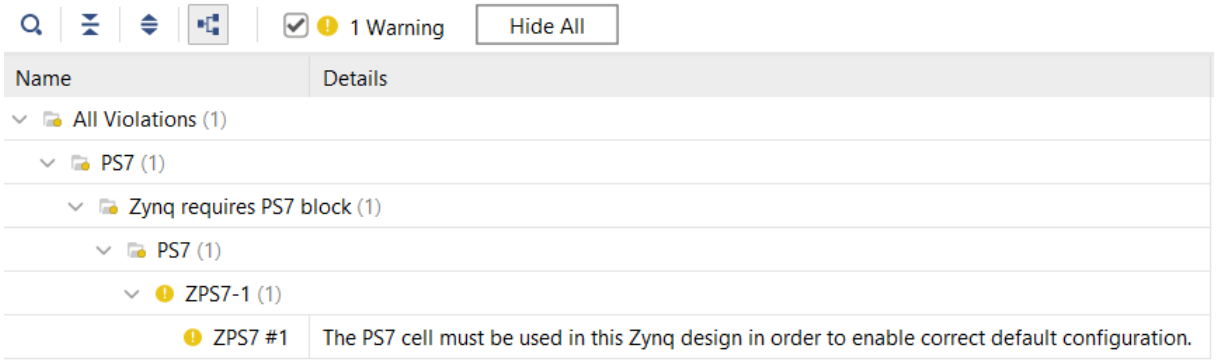


As can be seen in the image, a **total power of 0.092W is required, which is strongly unbalanced between static and dynamic power**.
Dynamic power covers about 3% of the entire consumption (0.002W), and static power

covers the remaining consumption (0.090W).
The greatest contribution for dynamic power comes from logic, which covers 45% of power consumption, then we have 33% for signals, while the remaining 22% is for clock.

5.5 DRC Violation

The only warning in the implementation results is:



We can ignore this warning in our case because it refers to the ARM processor environment.

We can get all these results if the implementation is run with the '-mode out_of_context' option, since **we do not want to actually test the project on the Zybo board**.

Chapter 6

Conclusions

The aim of this project is to design and implement a digital circuit to compute the fo coefficient in an image processing system, running on a Zybo Zynq 7000; **considering the successfully executed testbenches and the results of the implementation we can consider the work concluded**.
We can consider this implementation to be an architecture **that can be extended to meet real and more complicated needs that may need higher accuracy**.
We can hypothesize some future developments that this project may have, for example:

- Evaluate the **use of another FPGA that has fewer ports and is more suitable for our needs**.
- Consider **using another FPGA with a higher clock frequency** to **take advantage of the potential increase in frequency**.