POLITECNICO DI TORINO

Lab1: Design and implementation of digital IIR Filter

INTEGRATED SYSTEMS ARCHITECTURE

GROUP 08

Rao Muhammad Wajdan s289627
Muhammad Haris Khaan s290345
Muhammad Talha Farooq s28953

Introduction

The construction of a digital IIR filter is the topic of this laboratory report. Firstly, started with the creation of the reference model, which is a 2° IIR Filter in Direct Form II (DFII). The laboratory files were used to create the pseudo-fixed point reference model in Matlab and the fixed-point reference model in C.The difference equations for the project's causative, LTI IIR Filter, and its Block Diagrams in DFI and DFII are then shown in this report, followed by the reference model architecture. These parts are followed by a description of VHDL design and ModelSim simulations. The reference model's last phases were implementation (logic syntehsis), placement, and validation.

After all of these components for the reference filter were accomplished, an enhanced architecture for the filter was created. This involved applying one step of the Look-ahead Transform and obtaining a 3rd Order IIR Filter. All of the previously specified processes were repeated for this improved model, and they are presented in this report. Also, there is a short section that compares the non-optimized and optimized versions. Appendices are also supplied for further information on the VHDL code, Matlab scripts, and the more thorough data acquired. The text results represent a subset of the output filter results.

1 Reference filter model Development

1.1 Main characteristics summary

Just for summarizing, the main characteristics of the system to develop are:

- Filter type: IIR as, $p = 0$ (Group 08)
- Cut-off frequency: 2 kHz; sampling at 10 kHz

1.2 Filter design and coefficient quantization process

As, by using the Surnames of groupmates in descending order as,

Wajdan, Khaan, Farooq so, y=6, x=5

After applying equations 1 and 2 from given description file, it was obtained that $N = 2$ and $n_b = 14$.

1.3 Filter testing

1.3.1 Matlab model: pseudo-fixed-point representation

The values for N and nb were obtained and then entered into my iir filter.m file. This code was run immediately after that. Fig 1, Fig 2, and Fig 3 are examples of the results received after running the previously given file.

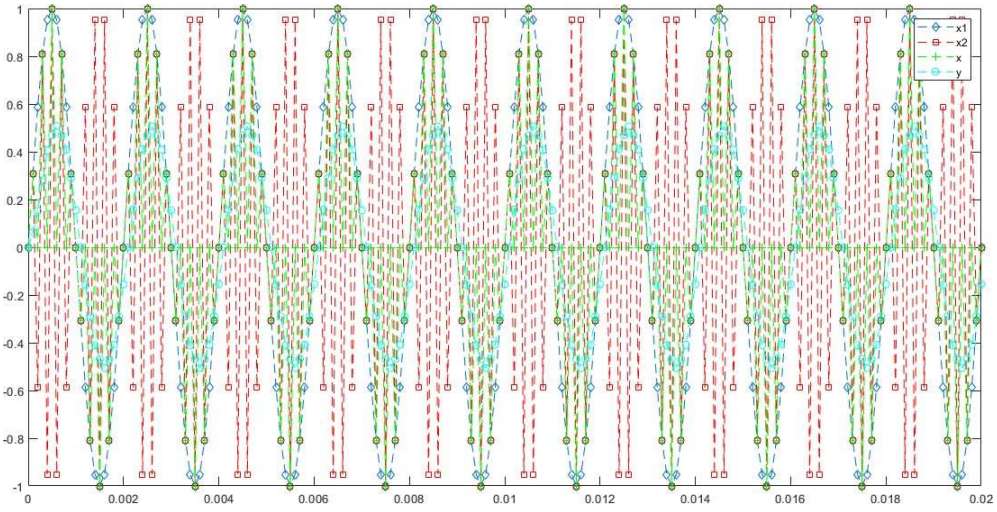

Fig 1: Reference filter Matlab waveforms

There are four waveforms in Fig 1. Waveform 'x1' (blue) is 500 Hz sine wave (in band), 'x2' (red) are 4500 Hz sine wave (out of band), 'x' (green) is the average of the two, and 'y' (light blue) is the output of the quantized coefficients filter. The horizontal axis corresponds to time in seconds, while the vertical axis is normalized to a typical sinusoid. T = 10 1/f = 10 1/500 = 0.02 s is the time span between 0 and 10. Samples were taken in 1/fs = 0.0001 s increments.

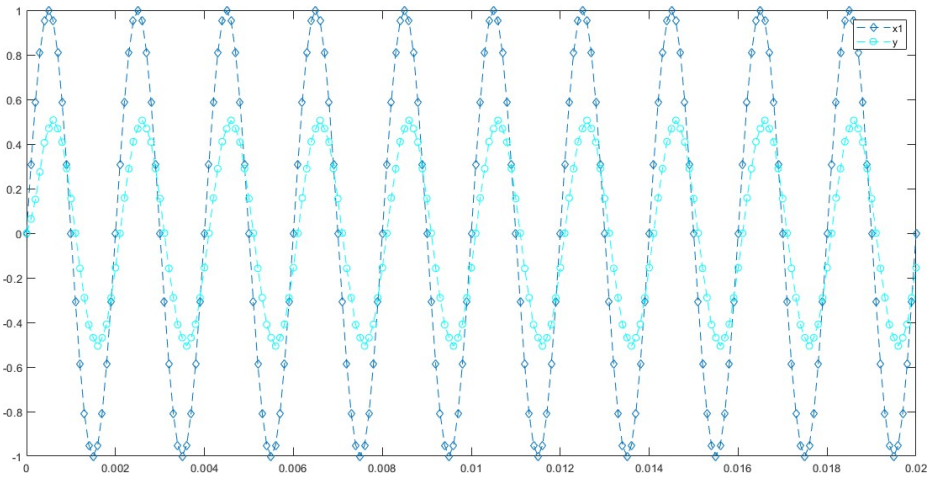

Fig 2: Reference filter Matlab ‘x1’ & ‘y’ waveforms

Fig 2 highlights 'x1' with blue and again utilizes light blue for 'y.' Everything that was said about Fig 1 applies to this as well..

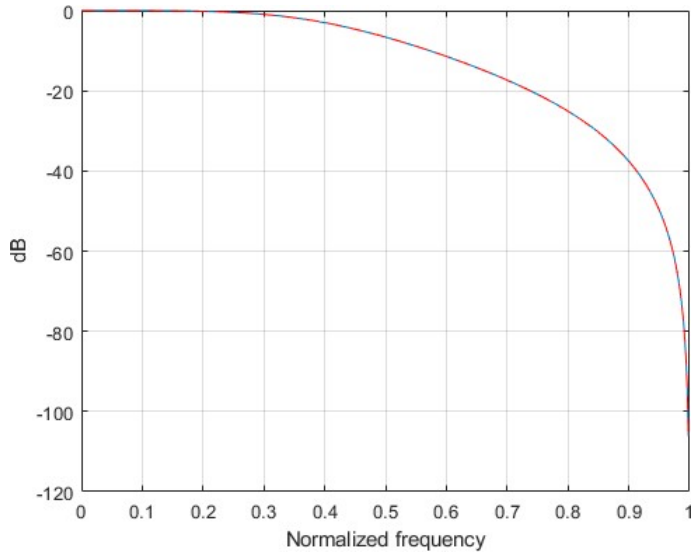

Fig 3: Designed and quantized reference filter transfer functions (Matlab)

Fig 3 is the last illustration, and it depicts two overlapping transfer functions. The first transfer function, shown in red, is a quantized filter design. The light blue one, on the other hand, represents the transfer function of the constructed filter using the a and b coefficients acquired using Matlab's butter function. It's also clear that the behavior is low pass, as one would anticipate from a Butterworth Filter. Coefficients were acquired by running my iir filter.m script for the reference filter to develop in addition to the prior numbers. Starting with the butter function's a and b coefficients (type: double), two coefficient variants were created. Table 1 shows the various variants previously discussed.

Table 1: 2° IIR Digital Filter – Reference model coefficients (Matlab)

Value of i or q	**Feedback a_i**	**Feedback a_q**	**Feed-forward b_i**	**Feed-forward b_q**
0	8192	1.000	1692	0.2065
1	-3028	-0.3696	3384	0.4131
2	1604	0.1958	1692	0.2065

The coefficients in Table 1 with a I subscript indicate integer coefficients in nb = 14 bits that were generated following the procedure in myiir design. a (this method also quantizes the coefficients with the floor function). The ones with a 'q' subscript are quantized and normalized versions of the integer coefficients with respect to 2^{13} (also known as nb = 14 bit real values). Despite having 201 samples, Table 2 only displays the first forty values of the filter since the output values recur frequently after the eleventh value I = 10) as predicted. This may be seen in the table beginning with I = 30. The sole exception is the range of I = 0 to I = 9. As a result, values between I = 40 and I = 49 will be the same as values between I = 20 and I = 29.

Table 2: 2° IIR Digital Filter – first forty output values (Matlab) of the reference model

i	Value	*i*	Value
0	0	20	-1269
1	522	21	-4
2	1238	22	1292
3	2247	23	2370
4	3325	24	3360
5	3850	25	3838
6	4155	26	4144
7	3843	27	3841
8	3344	28	3345
9	2375	29	2376
10	1268	30	1268
11	3	31	3
12	-1293	32	-1293
13	-2371	33	-2371
14	-3361	34	-3361
15	-3839	35	-3839
16	-4145	36	-4145
17	-3842	37	-3842
18	-3346	38	-3346
19	-2377	39	-2377

1.3.2 C model: fixed-point representation

The myfilterII.c code, which employs the DFII for the IIR Filter, was downloaded for the fixed-point representation in C. The filter order (for the original, non-optimized version N = 2 as mentioned), the number of bits nb = 14, and the ai and bi coefficients from the Matlab pseudo-fixed model were the only changes applied to this file. Only the first forty output values acquired using the C model will be shown today, in order to be consistent with the Matlab output samples already presented. However, it should be emphasized that the output values in this example (C) were sinusoidal but not periodical; there would be some minor changes between periods. For further information, see the C results of the non-optimized filter in the Appendices, which has all 201 output values.

Table 3: 2° IIR Digital Filter – first forty output values (C) of the reference model

i	Value	*i*	Value
0	0	20	-1270
1	522	21	-5
2	1238	22	1293
3	2245	23	2370
4	3324	24	3361
5	3848	25	3839
6	4154	26	4144
7	3841	27	3840
8	3343	28	3344
9	2375	29	2375

10	1268	30	1267
11	3	31	2
12	-1293	32	-1295
13	-2371	33	-2372
14	-3362	34	-3362
15	-3841	35	-3839
16	-4147	36	-4145
17	-3842	37	-3841
18	-3347	38	-3347
19	-2377	39	-2377

1.3.3 Matlab and C model comparison

The numbers in Table 2, which refers to the Matlab output values, and Table 3, which contains the C values, are, as predicted, different. The discrepancy, however, is thought to be within the predicted range. The largest difference is two digits (least significant digits in base 10) in absolute terms, i.e. a maximum absolute error of 2, which was discovered among all 201 values and their related Matlab output values, as shown in the tables. This is to be expected because of the way of representing integers in Matlab and C in this labo- ratory. In Matlab, the script that was used builds pseudo-fixed point integer coefficients starting from double type values (IEEE double precision format for floating point numbers), which are theoutput data types of the *butter* function. These values are shifted by 13 positions to the left and,then, the floor function is applied to them. On the contrary, C uses int data types directly and shifts in the code for this laboratory.

2 VLSI implementation:

2.1 Initial architecture development

2.1.1 Filter interface

Fig 4 shows the filter interface required for this laboratory. It's been taken from the laboratory instructions.

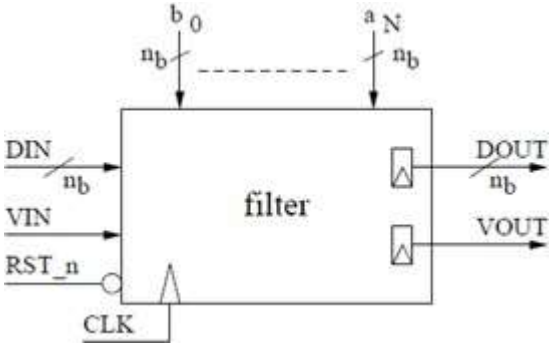

Fig 4: Lab filter interface

2.1.2 2° Order IIR Filter: general equation

The general form for a finitely-computable, causal, LTI system that depends on current and past inputs and past outputs is the following:

$$y[n] = \sum_{k=1}^{N} a_k \cdot y[n-k] + \sum_{k=0}^{M} b_k \cdot x[n-k] \quad (1)$$

Where x[] and y[] refer to the input and output signals, respectively. Additionally, the feedback coefficients are represented by ak with k [1, N], whereas the feed-forward coefficients are represented by bk with k [1, M]. This laboratory's filter is a 2nd Order Infinite Impulse Response (IIR) Filter, with two poles in the Z-plane. As a result, if N = 2 and M = 2, the general form of a generic 2nd Order Discrete Time IIR Filter may be obtained using Equation 1. Of course, in a generic system like the one just described, feed-forward coefficients b0, b1, b2 and feedback coefficients a1, a2 will be present. Section 2.3.1 contains the exact values for this laboratory. The following is the generic difference equation for such a system:

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + a_1 \cdot y[n-1] + a_2 \cdot y[n-2] \quad (2)$$

Where $x[n]$ and its shifted versions are the input samples to the filter coming from the discrete time input signal $x[\cdot]$ (or continuous time input signal $x(\cdot)$ which is previously sampled). Similarly, $y[n]$ and its shifted versions correspond to the output samples of the discrete time output signal $y[\cdot]$ coming out of the filter. It's worth noting that, due to its impulse response, a filter like the one described in Equation 2 would have a recursive architecture or structural block diagram. Additionally, beginning circumstances must be specified in order to identify the unique, specific system, as the same equations might have various solutions.As a result, initial rest conditions will be employed, which means that for the IIR Filter to optimize, x[n] = y[n] = 0 n 0 will be used. Finally, in Equation 2, the LTI IIR Filter n [0,] theoretically for a causal, but in this experiment, n [0, 200] to produce 201 samples.

2.1.3 Initial Block Diagrams: non-optimized filter

In Fig 5 there is Direct Form I or DFI Block Diagram of the filter described by Equation 2.

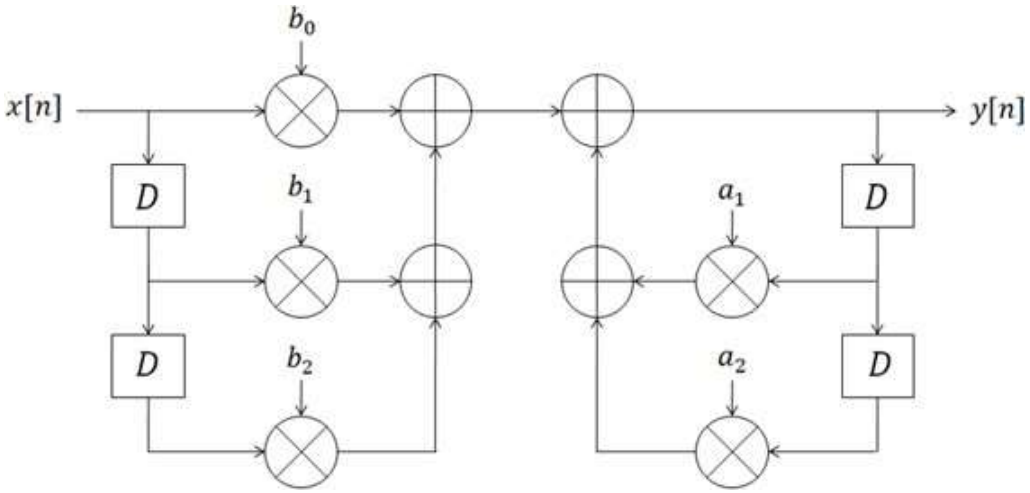

Fig 5: DFI Block Diagram of a generic, causal, 2nd Order IIR Filter

The recursive structure present in Fig 5 was used to develop the Direct Form II or DFII Block Diagram of the filter. This can be seen in Fig 6.

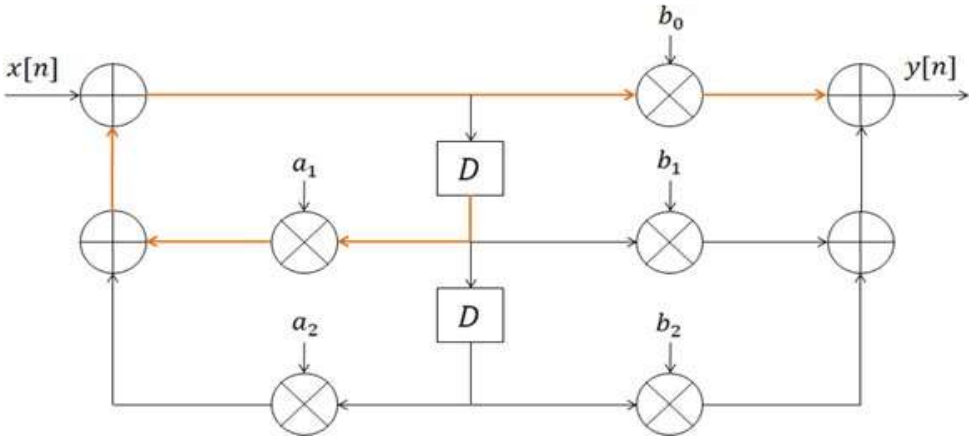

Fig 6: DFII Block Diagram of a generic, causal, 2nd Order IIR Filter

The crucial route is marked in orange in Fig 6, since there will be registers at the input and output for implementation purposes. From now on, the time period associated with critical pathways will be referred to as tcp, and the iteration bound will be referred to as T. By denoting multiplier execution times as Tm and adder execution times as Ta, it is possible to observe that:

$$t_{cp} = 2T_m + 3T_a > T_\infty = T_m + 2T_a \qquad (3)$$

Considering that Tm > Ta in general. This demonstrates that there are universal strategies for lowering the critical route delay in order to reach the iteration bound T. As a result, the 2nd Order IIR Filter was optimized utilizing pipelining to generate one optimized version and parallel processing with a degree of parallelism P = 2 to build a second optimized version at this step. These versions, however, were eliminated since the Look-ahead Transform was required first.Despite this, the 'non-optimized version' of the structure in Figure 6 was programmed in VHDL. This was done with the goal of subsequently confirming that the findings acquired by the "optimized version" (Section 5) were correct.

2.1.4 VHDL: reference IIR Filter

This section contains the VHDL implementation of the above-mentioned filter. The primary elements of the VHDL code may be separated into two categories. One will be the arithmetic section, which will include all of the blocks that do addition and multiplication. A fresh sample is provided to these arithmetic units in the other section, and the output is shifted to the output from all of the registers.
Let's get right to the point with the code discussion:

```
entity IIR_CA is
  port (
    CLK       : in    std_logic;
    RST_n     : in    std_logic;
        VIN       : in  std_logic;
        DIN                     : in    std_logic_vector(13 downto 0); --14 bit data input
```

```
        B0                  : in    std_logic_vector(13 downto 0);
        B1                  : in    std_logic_vector(13 downto 0);
        B2                  : in    std_logic_vector(13 downto 0);
        A1                  : in    std_logic_vector(13 downto 0);
        A2                  : in    std_logic_vector(13 downto 0);VOUT
            : out std_logic;
    DOUT    : out std_logic_vector(13 downto 0));

end IIR_CA;
```

All of the above-mentioned components were included. It was necessary to de-sign a 14-bit filter. An additional bit was considered to avoid overflow in the DOUT port, but this would conflict with the fact that the filter was 14 bits, thus it was decided to handle the overflow later in the code. The signal declaration part of the code is shown next.

```
signal fb       : signed(13 downto 0);
signal ff           : signed(13 downto 0);
signal w            : signed(13 downto 0); signal
reg1                  : signed(13 downto 0);
signal reg2           : signed(13 downto 0);
signal temp1: signed(27 downto 0); --If we have nbit the
multiplication can leads to 2*n bits. signal temp2:
signed(27 downto 0);
signal temp3: signed(27 downto 0); signal
temp4: signed(27 downto 0); signal temp5:
signed(27 downto 0); signal mul1 :
signed(13 downto 0); signal mul2 :
signed(13 downto 0); signal mul3 :
signed(13 downto 0); signal mul4 :
signed(13 downto 0); signal mul5 :
signed(13 downto 0);
signal xin: std_logic_vector(13 downto 0); signal DOUT_temp:
std_logic_vector(13 downto 0);
```

The following step was to specify all of the required signals, as shown above. This is simply accomplished by first identifying all of the signals on the Block Diagram, then specifying them here. Because the multiplication process might result in a result on 2n bits if n is the number of bits needed to complete the operation, special attention was taken with the signals used to transmit the result of multiplication. The 2n multiplication bit is stored in the temp signals before they are shifted to accept just the MSB bits. The arithmetic part is now revealed.

```
temp1  <=  reg1*signed(A1);--multi
        mul1    <= temp1(26 downto 13);--shifttemp2
        <= reg2*signed(A2);
        mul2    <= temp2(26 downto 13); --taking only 14 bit of MSB
        temp3 <= reg1*signed(B1); mul3
                <= temp3(26 downto 13);
        temp4 <= reg2*signed(B2);
```

```
mul4    <= temp4(26 downto 13);
temp5 <= w*signed(B0);
mul5    <= temp5(26 downto 13);fb
<= -mul1-mul2;--feedback ff <=
mul3+mul4;--feedforwardw
        <= signed(xin)+fb;
DOUT_temp <= conv_std_logic_vector(ff+mul5,14); --Converting it to vector 14 bits
DOUT <= DOUT_temp(13 downto 0);
```

The arithmetic part is shown above. All of the surgeries take place at the same time. The operations are carried out in accordance with the Block Diagram, which involves multiplying with the appropriate coefficients. To conserve the sign bits, signed data types are utilized. After that, the final result is transformed to 14 bits, and the remaining LSBs are discarded. Another alternative is to make the DOUT port 2N bit, although this is not as economical in terms of design due to the large number of pins (the final filter). Following that, a section of the code is given in order.

```
process(CLK,RST_n,VIN) begin
if RST_n = 'O' then -- Everything 0 if reset
          reg1 <= (others => 'O'); reg2
          <= (others => 'O'); VOUT  <=
          'O';
          xin    <= (others => 'O');
elsif CLK'event and CLK = '1' and VIN = '1' thenVOUT <= 'O';
          xin(13 downto 0) <= DIN;
          reg2 <= reg1; -- In proces this will model after a FF so no need to add an extra FF.
          reg1 <= w;
          VOUT <= '1';
          end   if;
end process;
```

This is the sequential part of the code thus this is only executed at each clock pulse. It's possible to have many delay elements in the original design so it's possible to define a component register and add it in between signals. Alternatively, it's possible to assign them here in process. As the assignment is done at every clock pulse the compiler automatically puts a register here to save the previous value and then pass it on when the clock pulse arrives. For being a very small system the assignment here in process statement seems feasible but in a large system it's better to put the registers.

2.2 Simulation

A simulation of the above-mentioned filter was run. The findings were identical to those obtained using the C program. The Appendices include the whole set of output results from the C code for the reference IIR Filter.

2.2.1 ModelSim Timing Diagrams

In this section, the timing diagram that was obtained while simulating the structure will be discussed.

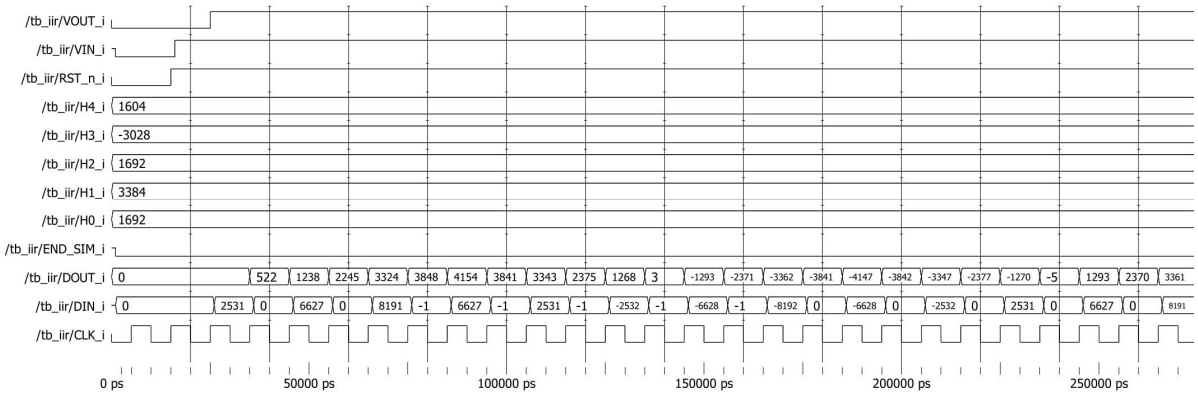

Fig 7: Unoptimized Filter Timing Diagram

In Fig 7, it's possible to see that some time after VIN changes, from low to high, it's followed by VOUT. Once the DIN is available after some delay, it's also possible to see that data on the DOUT port starts to appear.

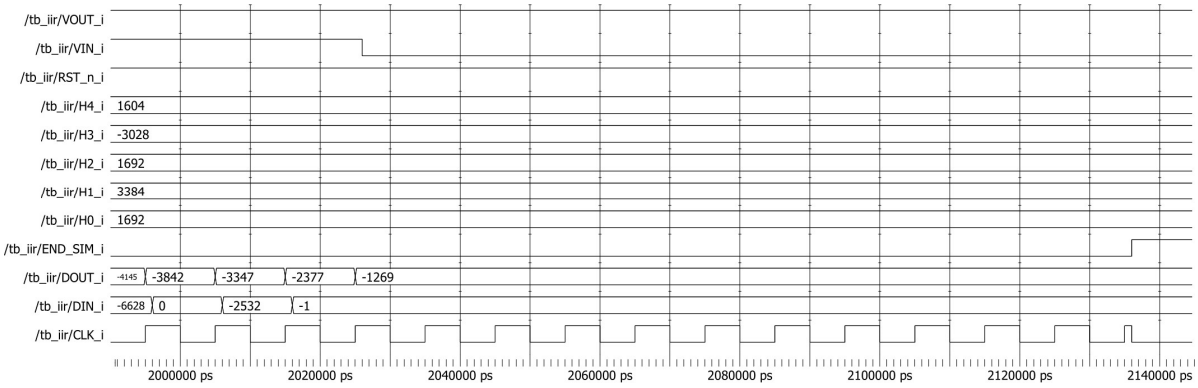

Fig 8: Unoptimized Filter Timing Diagram Showing Extra Signals

Fig 8 depicts the operation of signal END SIM, which is critical because on a physical design, this signal must be examined in order to verify the final results. This pin can be linked to an interrupt pin or added to an ADC pin that can be watched continuously. And, fig 9 depicts the reference filter's proper behavior with regard to the VIN signal.

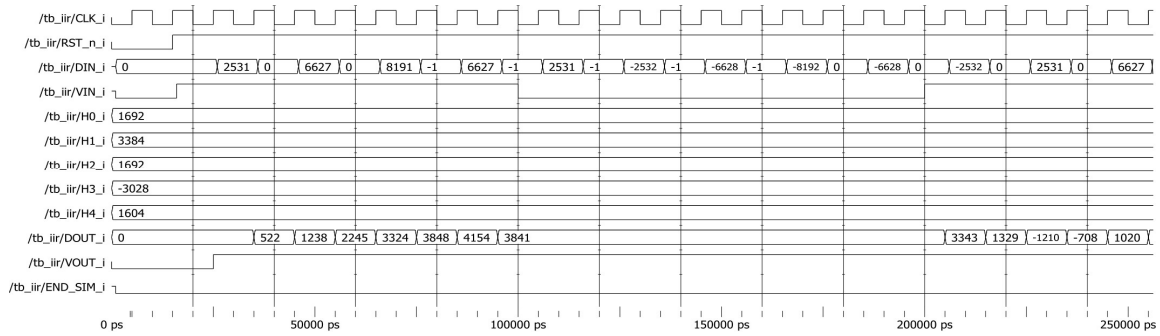

Fig 9: Behavior of VIN in the Reference IIR Filter

2.2.2 ModelSim and C result comparison

In the same way that int data types are used in C++, VHDL offers a package that implements fixed point numbers and combines them easily with other read formats and operations like addition and subtraction.The std logic vector data type with 14 bits was utilized to represent numbers in the design for this lab, but the int data type in C was used to allocate a lot of space for expressing a fixed point number. The amount of bits required is moved in C code, resulting in the same number of bits being taken from the MSBs. In the VHDL, on the other hand, all operations are performed with the same amount of bits, ensuring that no data is lost. This is supported by the fact that the results from C and VHDL matched flawlessly after simulation and output data retrieval - they were identical. As a result, only the C findings are included in Appendix I.

2.3 Implementation

2.3.1 Logic synthesis

Synopsys is the program that will be used to synthesis the components in this project; it is a very popular tool that has a lot of powerful capabilities to help Electronic Engineers/students synthesize their work and get a few reports that may modify the design due to design restrictions. Scripts and reports generated by Design Compiler Graphical will be displayed in this chapter, particularly the Time, Power, and Area reports, which are the most critical parts of any Electronic Engineer/student design.

The first thing done was to open the *Design Compiler Graphical* by launching the terminal in a folder where the work folder will be desired to be saved on. For launching the tool two files were used: *init synopsys* and *.synopsys dc.setup.* The commands used:

source init_synopsys
design_vision

If the two files stated above are in the same location and Synopsys is properly installed, Design Compiler Graphical should launch normally. It is now able to analyze the files that must be synthesized. Only one component, termed IIR CA, was examined in the unoptimized version, hence the command used was:

analyze -f vhdl -lib WORK ./IIR_CA

Another command was used to help in estimation the power consumption which is:

set power preserve rtl hier names true

Then elaboration was done using this command:

elaborate IIR_CA -arch bhv -lib WORK > elaborate.txt

```
Loading db file '/software/synopsys/syn_current/libraries/syn/gtech.db'
Loading db file '/software/synopsys/syn_current/libraries/syn/standard.sldb'
  Loading link library 'NangateOpenCellLibrary'
  Loading link library 'gtech'
Running PRESTO HDLC

Inferred memory devices in process
        in routine IIR_CA line 61 in file
                './src/IIR.vhd'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       xin_reg       | Flip-flop |   14  |  Y  | N  | Y  | N  | N  | N  | N  |
|      reg1_reg       | Flip-flop |   14  |  Y  | N  | Y  | N  | N  | N  | N  |
|      reg2_reg       | Flip-flop |   14  |  Y  | N  | Y  | N  | N  | N  | N  |
|      VOUT_reg       | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
===============================================================================
Presto compilation completed successfully.
Elaborated 1 design.
Current design is now 'IIR_CA'.
1
```

Here, it was asked to find the maximum frequency, by finding the clock value that generates the slack equals to 0, in order to create this clock, this command was used:

create clock -name MY CLK -period 10 CLK

This period was changed couple of times until the slack came out 0 in the timing report. However, the clock found for the maximum frequency is 6 ns, so the command later on used was:

create clock -name MY CLK -period 6 CLK

It was essential to force the compile not to change the clock by using this command:

set dont touch network MY CLK

Due to any unanticipated delays, such as jitter, some uncertainties will now be added to the clock and the inputs. Furthermore, it was anticipated that all of the inputs had the same delays that would be influenced by the clock. The commands that were utilized:

set clock uncertainty 0.07 [get clocks MY CLK]

set input delay 0.5 -max -clock MY CLK [remove from collection [all inputs] CLK]

set output delay 0.5 -max -clock MY CLK [all outputs]

It was assumed that the load of each output is the input capacitance of a buffer. The buffer used is in the *NangateOpenCellLibrary* technology is BUF X4, which has input with name A. The commands used for doing so are:

set OLOAD [load of NangateOpenCellLibrary/BUF X4/A]

set load $OLOAD [all outputs]

Then simply after setting all configuration and analyzing the required documents, it is time to compile. The command for that is simply:

Compile

It is now able to view the reports that are necessary to check the design. Furthermore, as previously stated, the reports that will be reviewed presently are time and area reports, with power reports to follow. The commands for creating timing and area reports may be found below:

report timing > timing 166.67M.txt

report area > area report@6ns.txt

The timing report shows the following:

```
clock MY_CLK (rise edge)                                 6.00       6.00
clock network delay (ideal)                              0.00       6.00
clock uncertainty                                       -0.07       5.93
output external delay                                   -0.50       5.43
data required time                                                  5.43
--------------------------------------------------------------------------
data required time                                                  5.43
data arrival time                                                  -5.43
--------------------------------------------------------------------------
slack (MET)                                                         0.00
```

As explained before, it was required to find slack equals to zero, this means, that that clock used, was the maximum clock possible. It is possible now as requested to use maximum frequency dividedby 4, but first, let's show the area using this clock.

```
Library(s) Used:

    NangateOpenCellLibrary (File: /home/isa12/Desktop/ISA2018/lab1/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:            102
Number of nets:             382
Number of cells:            145
Number of references:        17

Combinational area:      3995.586022
Noncombinational area:    228.760007
Net Interconnect area:     undefined  (Wire load has zero net area)

Total cell area:         4224.346191
Total area:                undefined
1
```

For the maximum frequency divided by 4, the timing report shows the following:

```
clock MY_CLK (rise edge)                                  24.00      24.00
clock network delay (ideal)                                0.00      24.00
clock uncertainty                                         -0.07      23.93
output external delay                                     -0.50      23.43
data required time                                                   23.43
-----------------------------------------------------------------------------
data required time                                                   23.43
data arrival time                                                   -5.85
-----------------------------------------------------------------------------
slack (MET)                                                         17.58
```

While the area shows:

```
Library(s) Used:

    NangateOpenCellLibrary (File:
/home/isa12/Desktop/ISA2018/lab1/Non_optimized/Fm_divide4/Synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:            102
Number of nets:             379
Number of cells:            142
Number of references:        16

Combinational area:      3963.666022
Noncombinational area:    228.760007
Net Interconnect area:     undefined  (Wire load has zero net area)

Total cell area:         4192.426270
Total area:                undefined
1
```

There is a distinction between the area where the maximum frequency is used and the area where some slack is provided. The reason for this is because certain resources were preserved when compiling. In other words, with greater slack, it is feasible to postpone some resources since they will no longer be vital and may be freely employed. However, as demonstrated, the difference is not significant, implying that the design is sequential, implying that each data is dependent on the one before it, enabling the resources to be conserved.

It is now able to combine the hierarchy and export a Verilog file that contains all of the components utilized in the top-level entity. This is useful for modeling the design after synthesizing to ensure that

the results haven't changed in ModelSim. This netlist will also be utilized in ModelSim to verify for the power-switching simulation. SDF and SDC are also significant since they include the netlist's delays as well as inputs and outputs limitations, respectively. To create such files, you must first ungroup the cells in order to flatten the hierarchy, then transform the hierarchy to Verilog type, and last export. The instructions to do the following tasks are as follows:

ungroup -all -flatten
change_names -hierarchy -rules verilog
write_sdf ./netlist/IIR.sdf
write -f verilog -hierarchy -output ./netlist/IIR.v
write_sdc ./netlist/IIR.sdc

As will be shown in Chapter 6, it is now able to test the findings using ModelSim and the Verilog file (IIR.v). It was also necessary to create a saif file for the design, which is significant since it contains information about switching power activities. It was necessary to produce the saif file using the technique utilized before to NangateOpenCellLibrary in order for this to operate. To create this file, first launch the non-graphical Design Compiler with following commands.:

source init_synopsys
dc_shell-xg-t

After that, read the technology, then convert it to saif file, using these commands:

read_file NangateOpenCellLibrary_typical_ecsm_nowlm.db
lib2saif -out ./saif/NangateOpenCellLibrary.saif NangateOpenCellLibrary

It is possible now to use *ModelSim* to generate the IIR back.saif which will be having the informa- tion of the switching power for the unoptimized version using the maximum frequency divided by 4, since this is what is focused on as requested.

After that, using this file, it is possible to return to *Synopsys* Design compiler to generate a power report. To generate the power report in Design_compiler, simply these commands were used:

read_verilog -netlist ./IIR.v
read_saif -input ./saif/IIR back.saif -instance tb iir/UUT -unit ns -scale 1
create clock -name MY CLK CLK
report_power > power_report.txt

The power report shows the following:

```
Library(s) Used:

    NangateOpenCellLibrary (File:
/home/isa12/Desktop/ISA2018/lab1/Non_optimized/Fm_divide4/Synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)


Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
IIR_CA                 5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 312.2427 uW   (51%)
  Net Switching Power  = 301.1029 uW   (49%)
                         ---------
Total Dynamic Power    = 613.3456 uW  (100%)

Cell Leakage Power     =  86.3076 uW
```

As can be observed from the above data, the total Dynamic power contributes the most to the overall power because it deals with the amount of components employed in the design. However, this is seen as a minor consideration. Also, as previously said, the lower the area, the lower the dynamic power, thus since the maximum frequency divided by four is lower, the total dynamic power is projected to be lower. The power report of the maximum frequency, on the other hand, confirms this.

```
Library(s) Used:

    NangateOpenCellLibrary (File: /home/isa12/Desktop/ISA2018/lab1/synopsys/Powerswitching/NangateOpenCellLibrary_typical_ecsm_nowlm.db)


Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
IIR_CA                5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 313.2803 uW   (51%)
  Net Switching Power  = 302.6384 uW   (49%)
                         ---------
Total Dynamic Power    = 615.9188 uW  (100%)

Cell Leakage Power     =  87.2843 uW
```

2.4 Advanced filter model development

2.4.1 Matlab model: pseudo-fixed-point representation

This time, the *myiirfilter.m* file was modified to change the filter order after the application of the Look-ahead Transform. Now, $N = 3$. The *myiir design.m* file was also modified so as to have the new coefficients starting from the original ones, following equation 1. After running *my iir filter.m*, three plots were obtained once again. These are shown in Figure 10, Figure 11 and Figure 12.

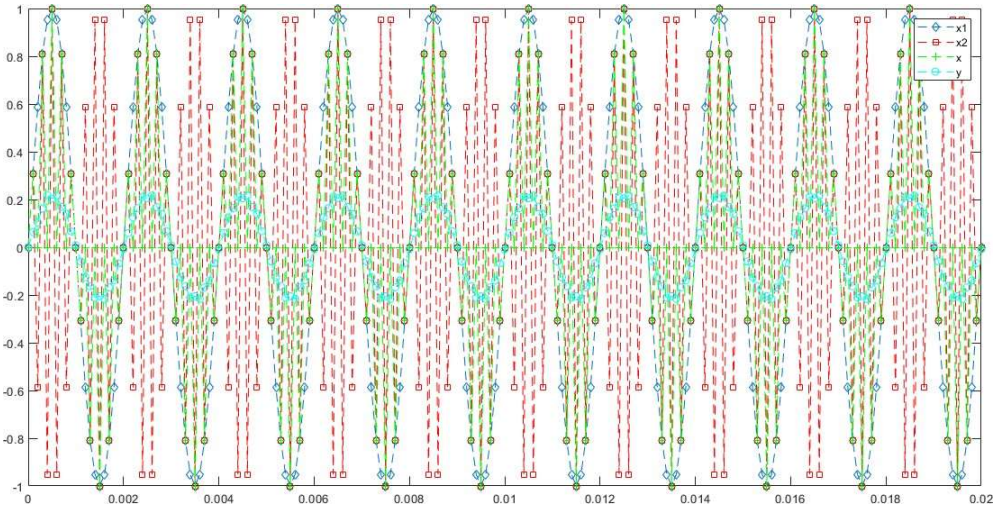

Figure 10: Optimized filter Matlab waveforms

Signals ‘x1’, ‘x2’, ‘x’ and ‘y’, the sampling frequency, the time steps, the time range, the horizontal and vertical axis are all the same as before. Once again, it’s possible to see that ‘y’ is decreased in amplitude with respect to ‘x1’. Signal ‘y’ also has the same frequency as ‘x1’, just like before. Thisis more evident in Figure 11.

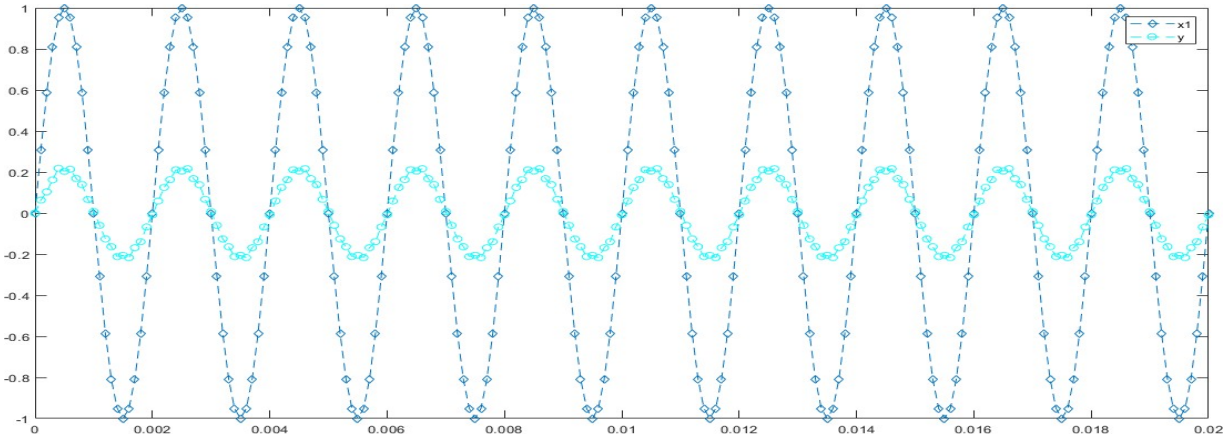

Figure 11: Optimized filter Matlab ‘x1’ and ‘y’ waveform

Figure 11 shows that, apparently, there is either no phase shift or a very small phase shift between 'x1' and 'y'. Next, Figure 12 shows the comparison of two filter transfer functions which will be immediately described.

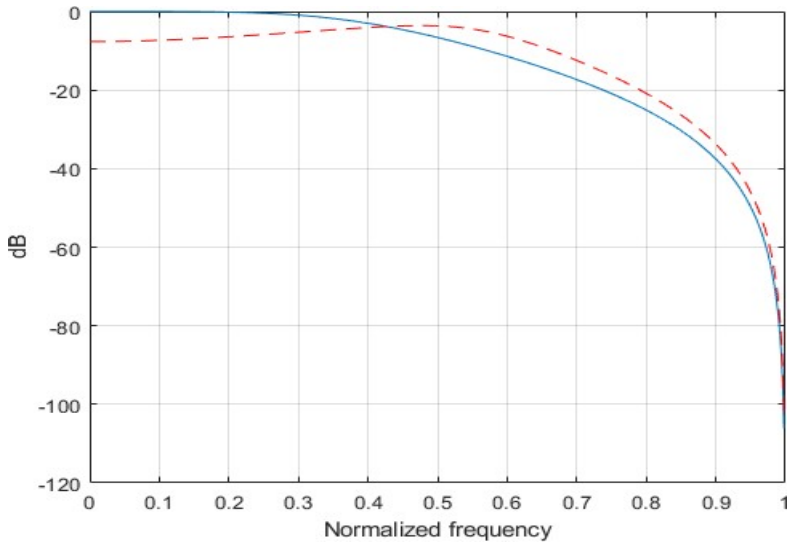

Figure 12: Reference filter and optimized filter with quantized coefficients - transfer functions (Matlab)

Figure 12 shows, in blue, the transfer function of the original 2nd order IIR Filter using the originala and b coefficients coming from the *butter* function. The red line represents, for its part, the transfer function of the optimized, 3rd order IIR Filter using quantized. And, Table 4 reports the coefficients obtained for the optimized filter using the modified matlab code scripts.

Table 4: 3° IIR Digital Filter – Optimized model coefficients (Matlab)

Value of *i* or *q*	**Feedback a_i**	**Feedback a_q**	**Feed-forward b_i**	**Feed-forward b_q**
0	8192	1.000	1692	0.2065
1	0	0	2758	0.3367
2	2723	0.3324	441	0.0539
3	-593	-0.0724	-626	-0.0763

From Table 4, just like before, only the a_i and b_i coefficients will be used for the C and VHDL implementation, except for a_0 = 8192. Finally, Table 5 shows the first forty output values of the optimized IIR Filter. These were obtainedby running the new *my iir filter.m* script.

Table 5: 3° IIR Digital Filter – first forty output values (Matlab) of the optimized model

i	**Value**	***i***	**Value**
0	0	20	-68
1	522	21	487
2	852	22	1028
3	1331	23	1338
4	1792	24	1731
5	1668	25	1678

6	1752	26	1773
7	1385	27	1377
8	1144	28	1138
9	546	29	550
10	66	30	67
11	-486	31	-488
12	-1029	32	-1029
13	-1340	33	-1339
14	-1732	34	-1732
15	-1679	35	-1679
16	-1774	36	-1774
17	-1378	37	-1378
18	-1139	38	-1139
19	-551	39	-551

The reasons for Table 5 having only the first forty samples are the same as for Table 2. The behavior in terms of output periodicity is exactly the same as described for the reference model as well.

2.4.2 C model: fixed-point representation

Once again, the *myfilterII.c* file was used. The new filter order $N = 3$ and the new a_i and b_i coefficients from Matlab were included in the C code for the optimized filter version. Table 6 shows, to be consistent with all previous samples in the report, the first forty output values coming from the C fixed-point model of the optimized filter.

Table 6: 3° IIR Digital Filter – first forty output values (C) of the optimized model

i	**Value**	*i*	**Value**
0	0	20	-69
1	522	21	485
2	852	22	1027
3	1331	23	1336
4	1791	24	1728
5	1666	25	1677
6	1749	26	1772
7	1383	27	1375
8	1143	28	1135
9	544	29	547
10	64	30	65
11	-487	31	-489
12	-1030	32	-1030
13	-1340	33	-1340
14	-1732	34	-1732
15	-1679	35	-1680
16	-1775	36	-1775
17	-1379	37	-1379
18	-1139	38	-1139
19	-552	39	-552

The entire set of 201 values, from which only 40 are shown in Table 6 to be consistent with previous data, was periodical and sinusoidal. For further details, please refer to the Appendices section.

2.4.3 Matlab and C model comparison

In this case, the maximum absolute error among all 201 output samples, with respect to their respective output valuse from Matlab, was 3 digits (ie. 3 in base 10). It's considered that this difference with respect to Matlab is due to the same reasons as discussed previously for the non optimized filter.

3 VLSI implementation: advanced architecture

3.1 Advanced architecture development

3.1.1 Look-ahead Transform

Following the laboratory instructions, a J-look-ahead transform was applied to the IIR Filter with $J = 1$. Starting from Equation 2, the following was done:

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + a_1y[n-1] + a_2y[n-2] / n \rightarrow n+1$$

$$\Rightarrow y[n+1] = b_0x[n+1] + b_1x[n] + b_2x[n-1] + a_1y[n] + a_2y[n-1]$$

Replacing the expression for $y[n]$ from Equation 2:

$$\Rightarrow y[n+1] = b_0x[n+1]+b_1x[n]+b_2x[n-1]+a_1(b_0x[n]+b_1x[n-1]+b_2x[n-2]+a_1y[n-1]+a_2y[n-2])+a_2y[n-1]$$

Now, expanding:

$$\Rightarrow y[n+1] = b_0x[n+1]+b_1x[n]+b_2x[n-1]+a_1b_0x[n]+a_1b_1x[n-1]+a_1b_2x[n-2]+a_1^2y[n-1]+a_1a_2y[n-2]+a_2y[n-1]$$

Applying the inverse transformation $n \rightarrow n-1$, the following is obtained from the previous equation:

$$\Rightarrow y[n] = b_0x[n]+b_1x[n-1]+b_2x[n-2]+a_1b_0x[n-1]+a_1b_1x[n-2]+a_1b_2x[n-3]+a_1^2y[n-2]+a_1a_2y[n-3]+a_2y[n-2]$$

$$\therefore y[n] = b_0x[n]+(b_1+a_1b_0)x[n-1]+(b_2+a_1b_1)x[n-2]+a_1b_2x[n-3]+(a_1^2+a_2)y[n-2]+a_1a_2y[n-3] \quad (4)$$

Block diagrams and optimization steps for this recursive architecture are shown in Section 5.1.2. Furthermore, Equation 4 represents a third order system with coefficients:

- $a_1 = 0$
- $a_2 = a_1^2 + a_2$
- $a_3 = a_1a_2$
- $b_0 = b_0$
- $b_1 = b_1 + a_1b_0$
- $b_2 = b_2 + a_1b_1$
- $b_3 = a_1b_2$

3.1.2 Post Look-ahead Transform: initial Block Diagrams

Similarly to the case of the original, non-optimized filter version, Figure 13 presents the DFI Block Diagram for the filter after applying the look ahead transform.

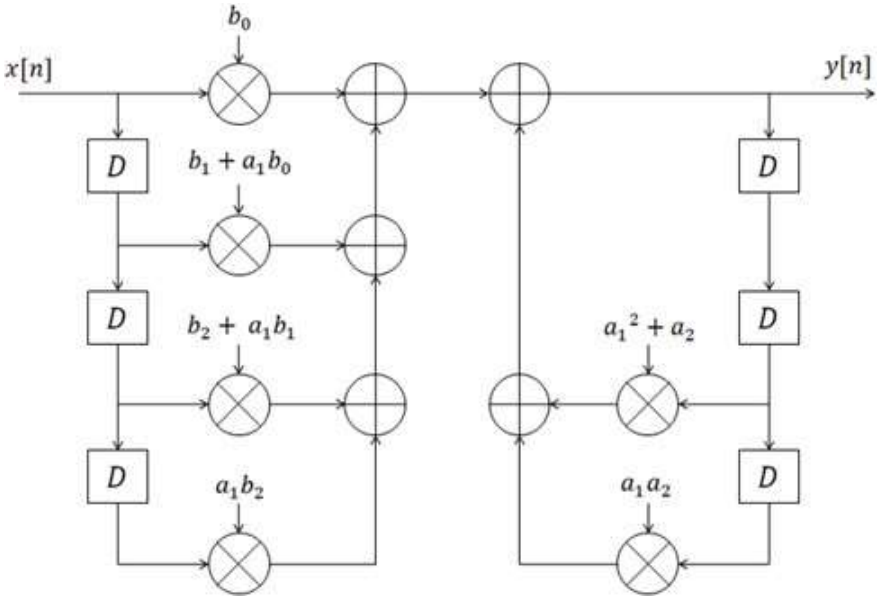

Figure 13: DFI Block Diagram of a generic, causal, 2nd Order IIR Filter after applying one Look-ahead Transform step

Just like before, the DFI Block Diagram from Figure 13 will be used to create the DFII Block Diagram. This is shown in Figure 14.

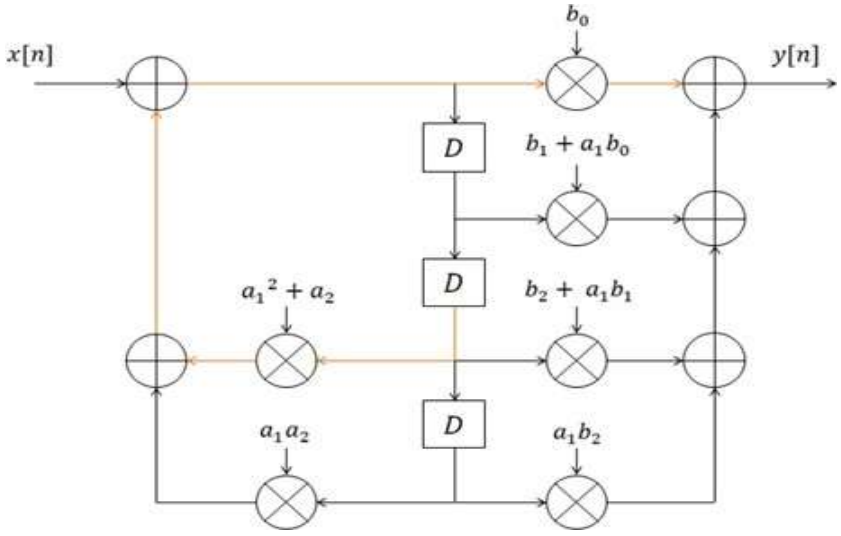

Figure 14: DFII Block Diagram of a generic, causal, 2nd Order IIR Filter after applying one Look-ahead Transform step

Figure 14 is important because it will be the structure to which the optimization steps will be applied. Similarly to before, here the proof that there's space for optimization is that:

$$t_{cp} = 2T_m + 3T_a > T_\infty = (T_m + 2T_a)/2$$

In fact, T_∞ is even lower than for the case of the original 2nd Order IIR Filter. This tends tohappen with a non universal technique such as the Look-ahead Transform. Just like before, a parallel processing version was developed using a formal mathematical method. However, it was considered to be more complicated in terms of implementation and code, so another option was sought. The specific steps are explored in the following subsection.

3.1.3 Post Look-ahead Transform: optimization methods

With the objective of decreasing t_{cp}, two universal optimization methods were applied. The first was pipelining and the second register retiming. The specific way in which these were performed are shown in Figure 15 and Figure 16.

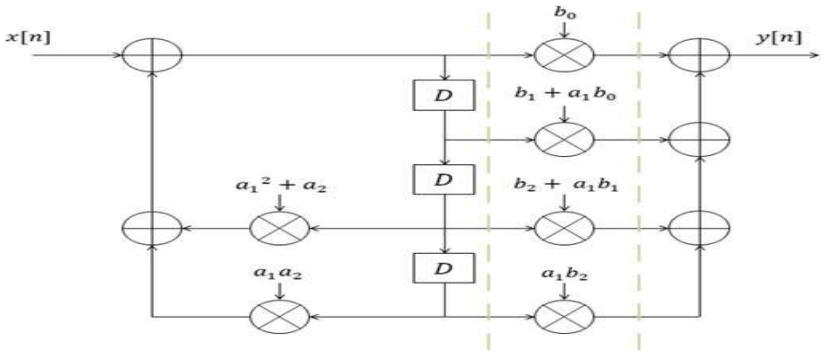

Figure 15: Feed-forward cutsets considered for pipelining the DFII Block Diagram of the laboratory IIR Filter

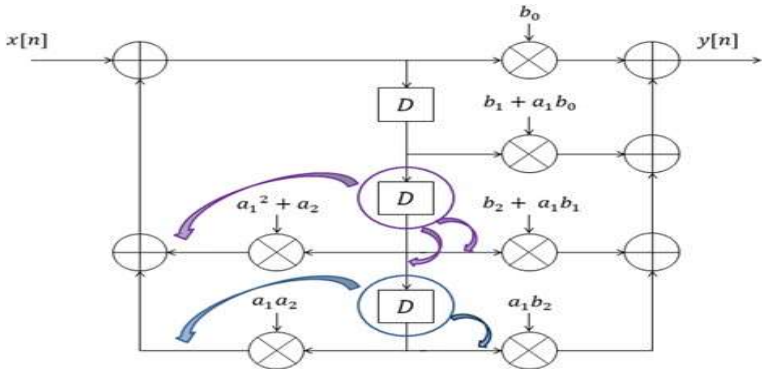

Figure 16: Register retiming method applied to the laboratory IIR Filter

Figure 15 highlights, in green, the two specific feed-forward cutsets used to pipeline the DFII Block Diagram. Similarly, but using different colors, Figure 16 indicates, specifically, which registers were relocated to perform the retiming and where these were moved. The specific locations are pointed at by the tip of the arrows.

3.1.4 Final optimized version

After applying the previously described optimization steps, the final, optimized version for the initial 2nd Order IIR Filter was obtained. This final version is shown in Figure 17.

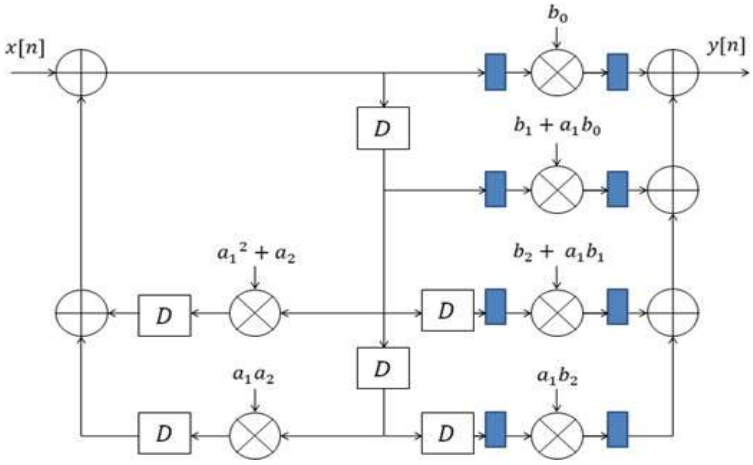

Figure 17: DFII Block Diagram of the optimized laboratory IIR Filter

Pipeline registers are colored in blue in Figure 17 to distinguish them from the original registers. All other registers are indicated in their retimed positions or original positions (in the case of the one that wasn't relocated).For this new optimized version, $T_\infty = (T_m + 2T_a)/2$ again. However, depending on the values of T_m and T_a, there are three possible values for t_{cp}. These will be denoted as t_{cp1}, t_{cp2}, t_{cp3}, where:

- $t_{cp1} = T_m$
- $t_{cp2} = 2T_a$
- $t_{cp3} = 3T_a$

Definitely, $t_{cp3} > t_{cp2}$, so in reality the two possible candidates are t_{cp1} and t_{cp3}. Without the use of software and the specific execution times of the multipliers and adders, it's not possible to continue optimizing any further. Also, it was asked to not perform fine grain pipelining, since also this can be done in software. Therefore, for these reasons, manual optimization was stopped at this point and so the structure in Figure 17 was coded in VHDL next.

3.1.5 VHDL: optimized IIR Filter

As it's possible to see in the Block Diagram after optimization, there are a lot of delay elements thus in this case it was decided to make a component register and thus connect it wherever it's needed. This leads to a very easy code which is easy to understand. Next, a section of a process for the register in the VHDL code is shown.

```
process(RST_n,VIN,CLK) begin
```

```
        if RST_n = '0' then
                q <= (others => '0');
        elsif CLK'event and CLK = '1' and VIN = '1' thenq <= d;
        end if;
    end process;
```

Above is a general code for the register component, which will latch the data to the output port when there is a rising pulse and Vin is high. Next, another section of the code is shown.

```
entity IIR_ADV is
  port (
        CLK       : in    std_logic;
        RST_n     : in std_logic;
        VIN       : in    std_logic;
        DIN           : in    std_logic_vector(13 downto 0);B0
                      : in    std_logic_vector(13 downto 0); B1
                      : in    std_logic_vector(13 downto 0); B2
                      : in    std_logic_vector(13 downto 0); B3
                      : in    std_logic_vector(13 downto 0); A1
                      : in    std_logic_vector(13 downto 0); A2
                      : in    std_logic_vector(13 downto 0);
        VOUT      : out std_logic;
        DOUT      : out std_logic_vector(13 downto 0));end
IIR_ADV;
```

The previous part is similar to the unoptimized model. The only differences are that the new coefficients are calculated already and are converted to bits as required. Following are the components initialized. These include a subtractor, multiplier, adder, and aregister.

```
component reg
port(
    RST_n     : in    std_logic;
    VIN       : in    std_logic;
    d     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);CLK :
    IN STD_LOGIC; -- clock.
    q     : OUT STD_LOGIC_VECTOR(13 DOWNTO 0) -- output
);
END component;

component mulxport
(
    mult1 : in std_logic_vector (13 downto 0); mult2 : in
    std_logic_vector  (13  downto  0);  prod:  out
    std_logic_vector (13 downto 0));
end     component;

component sumx

port (
    add1 : in std_logic_vector (13 downto 0); add2 : in
    std_logic_vector  (13  downto  0);  sum1:     out
    std_logic_vector (13 downto 0));
end component;

component     subx
  port (
    sub1 : in std_logic_vector (13 downto 0); sub2 : in
    std_logic_vector  (13  downto  0);  rel1:  out
    std_logic_vector (13 downto 0));
```

```
end component;
```

After the initialization all the components are mapped to the signals using port map. The basic idea can be seen from the code segment bellow:

```
REGISTER4: reg port map (RST_n,VIN,reg5,CLK,reg4_reg);
        p9: mulx port map(reg4_reg, B3, temp4);
        REG4_1: reg port map (RST_n,VIN,temp4,CLK, mul4_reg);
```

All the things that were discussed until now are responsible for the arithmetic execution of the system. The sequential part in this design consists only of providing a sample at every rising clock at the input port and taking data from the output port, as seen next.

```
process(CLK,RST_n,VIN)
                begin
                if RST_n = '0' then
                        VOUT <= '0';
                        xin    <= (others => '0');
                elsif CLK'event and CLK = '1' and VIN = '1' thenVOUT <=
                        '0';
                        xin(13 downto 0) <= DIN;VOUT
                        <= '1';
                end if;
        end process;
```

3.2 Simulation

3.2.1ModelSim Timing Diagrams

The main difference in the timing of the optimized structure is the little delay in the first result sample. This happens because many registers were introduced to increase the throughput. It's important not to get confused in this case as the pipelining will increase the throughput but will increase the delay until the first output sample as well.

The relevant timing diagrams are shown in Figure 18 and Figure 19.

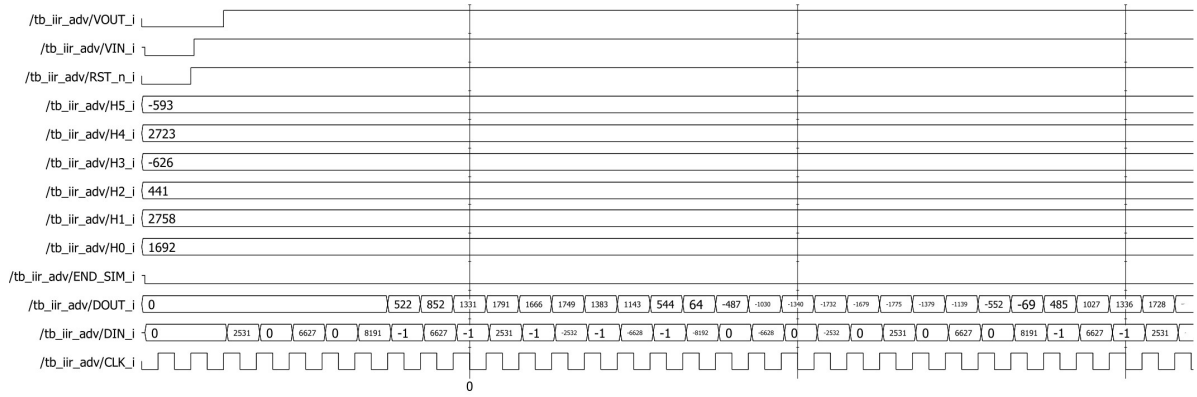

Figure 18: optimized Filter Timing Diagram

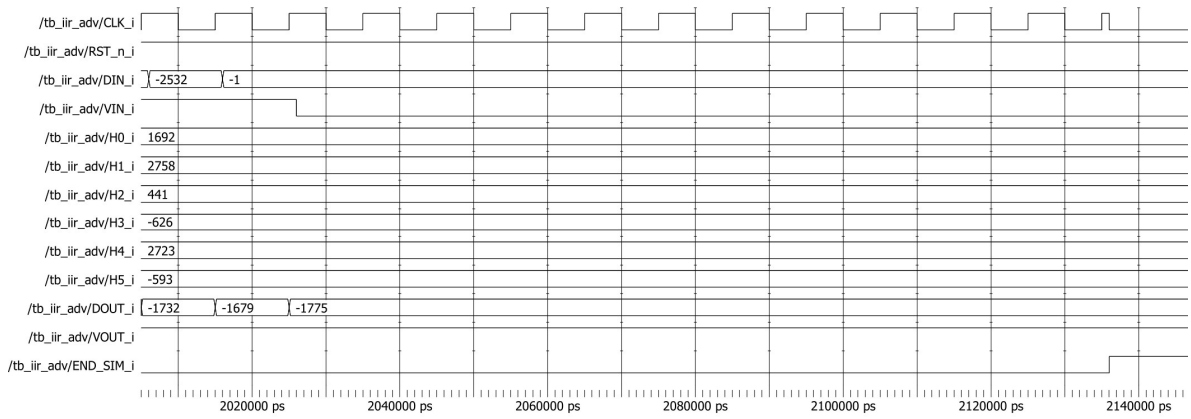

Figure 19: optimized Filter Timing Diagram with extra signal

Figure 19 is just to show that the END SIM is working desired. This is always important, to have such signals, to get a status update about process completion by the system. Finally, just like before, Figure 20 is shown here to highlight that the behavior with respect to the VIN signal is also working for the case of the optimized filter.

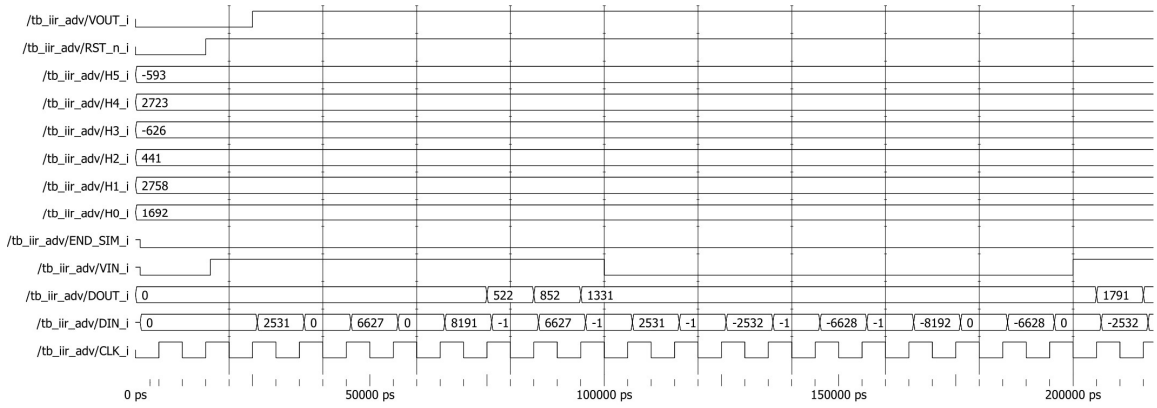

Figure 20: Behavior with respect to the VIN signal - Optimized IIR Filter version

3.2.2 ModelSim and C result comparison

As was previously discussed for the case of the reference IIR Filter, the result between the C and VHDL should be equal as they both use a particular number of bits to represent the data. Once again, after simulation and output data retrievement, it was verified that this was indeed the case for the case of the optimized IIR Filter. This means that the optimized version of the designed filter also has the same outputs as the C results.

3.3 Implementation

3.3.1 Logic synthesis

The steps that was taken in the unoptimized version will be exactly repeated, except for changing some names only, since the design will be optimized and changed. As done before, several attempts were done to check for the maximum frequency. However, it was found out that the slack equals to 0 when clock is 3 nanosecond. Furthermore, this proves thatthe design is highly improved since it changed from 6 nanosecond in the unoptimized version to 3 nanosecond in this version. Moreover, the timing report shows the following:

```
clock MY_CLK (rise edge)                                  3.00       3.00
clock network delay (ideal)                               0.00       3.00
clock uncertainty                                        -0.07       2.93
REG51/q_reg[13]/CK (DFFR_X1)                              0.00       2.93 r
library setup time                                       -0.04       2.89
data required time                                                   2.89
---------------------------------------------------------------------------
data required time                                                   2.89
data arrival time                                                   -2.88
---------------------------------------------------------------------------
slack (MET)                                                          0.00
```

While the area report shows the following:

```
Library(s) Used:

    NangateOpenCellLibrary (File: /home/isa12/Desktop/ISA2018/lab1/Optimized/Fm/Synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:              116
Number of nets:               513
Number of cells:               72
Number of references:          31

Combinational area:       5266.002016
Noncombinational area:    1122.520036
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          6388.521973
Total area:                 undefined
1
```

As expected, the area had to increase, since additional resources were used out to increase the performance, like additional registers. However, for the maximum frequency divided by 4, the timing report shows the following:

```
clock MY_CLK (rise edge)                                 12.00      12.00
clock network delay (ideal)                               0.00      12.00
clock uncertainty                                        -0.07      11.93
REG61/q_reg[13]/CK (DFFR_X1)                              0.00      11.93 r
library setup time                                       -0.04      11.89
data required time                                                  11.89
---------------------------------------------------------------------------
data required time                                                  11.89
data arrival time                                                   -3.21
---------------------------------------------------------------------------
slack (MET)                                                          8.67
```

As explained before, the area will be expected to decrease. The following is the area report for the maximum frequency divided by 4:

```
Library(s) Used:

    NangateOpenCellLibrary (File:
/home/isa12/Desktop/ISA2018/lab1/Optimized/Fm_divide4/Synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:              116
Number of nets:               513
Number of cells:               72
Number of references:          30

Combinational area:       5140.450020
Noncombinational area:    1122.520036
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          6262.970215
Total area:                 undefined
1
```

Now the power is also expected to increase, since more area acquired and obviously more dynamic power, as explained before. The following is power report for the maximum frequency divided by 4:

```
Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)


Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
IIR_ADV               5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 537.1018 uW   (58%)
  Net Switching Power  = 391.2656 uW   (42%)
                         ---------
Total Dynamic Power    = 928.3674 uW  (100%)

Cell Leakage Power     = 127.2722 uW
```

Definitely, if the power report was done for the maximum frequency it, would have been higher than the previous results, for the reasons explained before.

4 Place and Route

After verifying the results and synthesizing using *Synopsys* tools, it is possible to place and route using *Cadence* tool the design using the Verilog hierarchy exported by *Synopsys* for the maximum frequency divided by 4 for the optimized and the unoptimized versions. As mentioned in the document file for lab 1, provided by professor Maurizio, the steps to follow to obtain the correct results. The first thing done was to prepare the setup file, the following is what was modified in the customizable section:

set IN_DIR "../Synopsys/netlist"
set TopLevelDesign "IIR_CA"
set in_verilog_filename "$IN_DIR/IIR.v"
set in_sdc_filename "$IN_DIR/IIR.sdc"

Then steps will be followed as mentioned in the documents file. However, these steps will be done twice, for optimized and unoptimized version. The layout of the physical design for the unoptimized version is shown in Figure 21 after runningoptimization techniques using *Innovus* and checking the timing constraints using *Pre Clock-Tree- Synthesis*, and used *Clock-Tree-Synthesis* to reduce the loads and the delay of the clock tree.

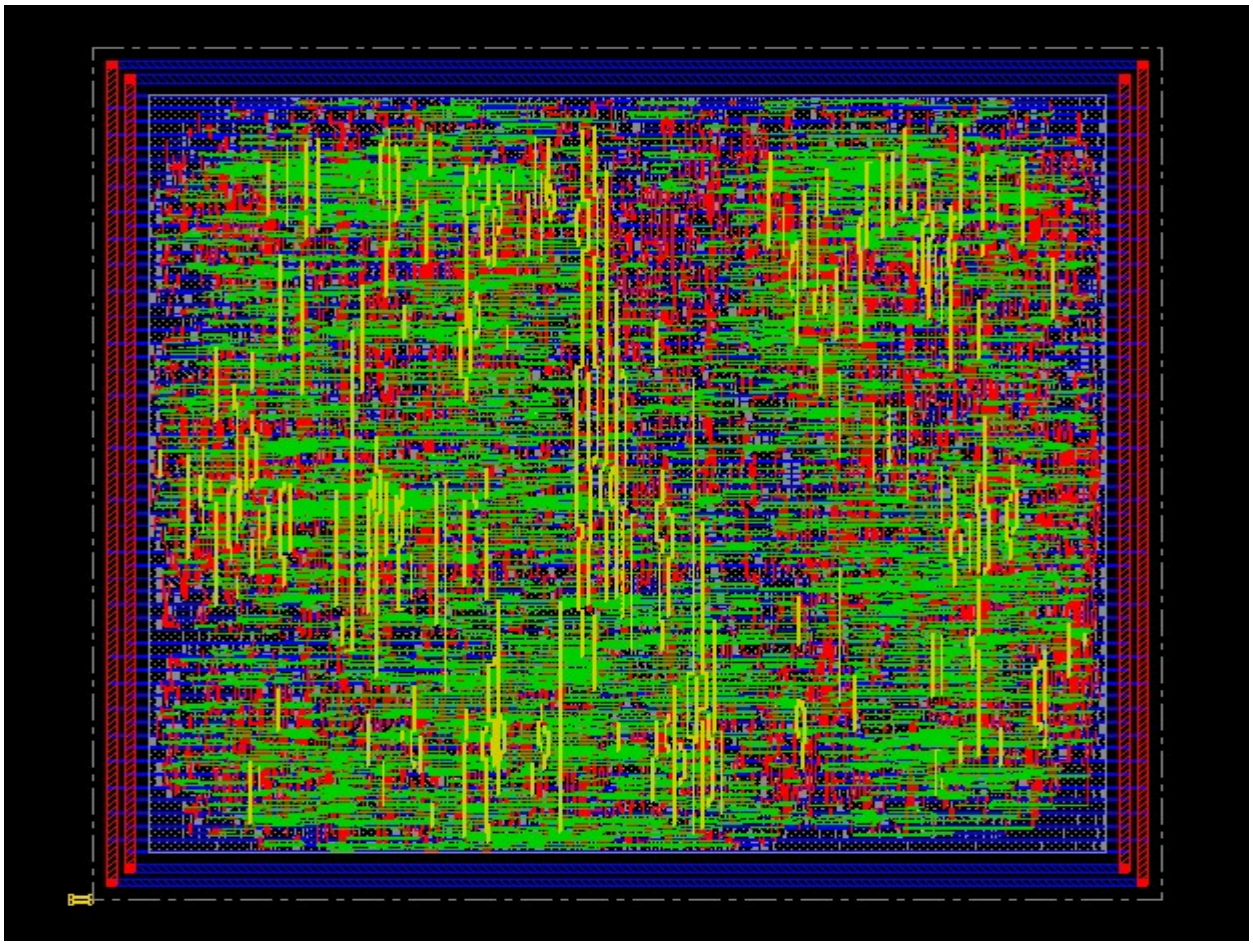

Figure 21: layout of the physical design of the unoptimized version

After analyzing time and verification, the gate count can be exported which showed the area after optimization using *Innovus*. However, the output was the following:

Gate area 0.7980 µm²
Level 0 Module IIR_CA
Gates= 5217
Cells= 2029
Area= 4163.2 µm

This means the area was reduced to 4163.2 μm^2 from 4192.626270 μm^2. For the optimized version, the same was done. In Figure 22 the layout of the physical design for the optimized version is shown.

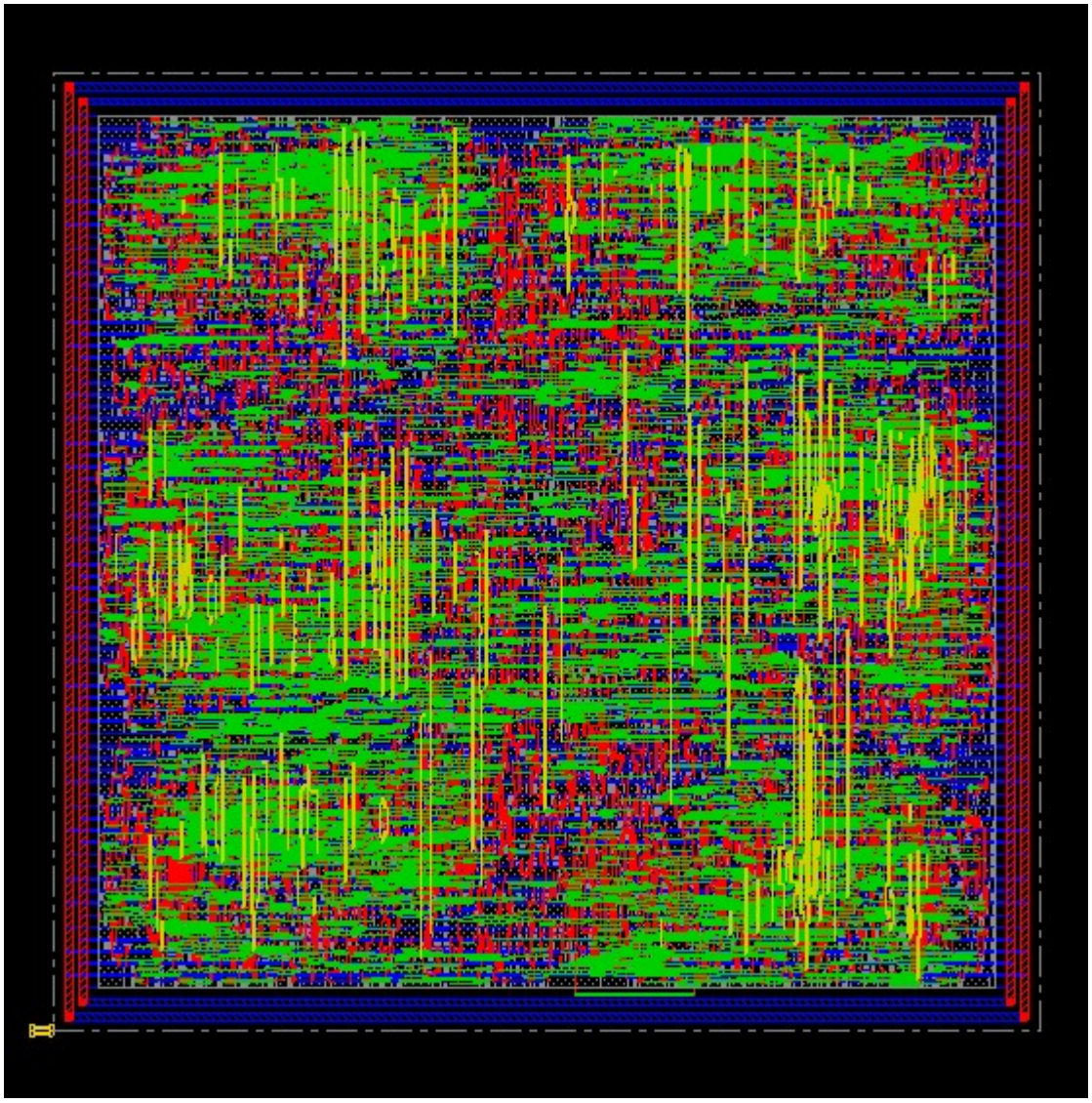

Figure 22: layout of the physical design of the optimized version

Then the gate count was the following:

Gate area 0.7980 Then μm^2
Level 0 Module IIR ADV
Gates= 7803
Cells= 2982
Area= 6227.3 μm^2

This means the area was reduced to 6227.3 μm^2 from 6262.970215 μm^2.
The new netlist and the delays of the netlist for both versions, optimized and the unoptimized can be saved to be used to calculate the new power consumption after optimizing the area. This canbe

done by simulating the new netlist using ModelSim as explained in Chapter 7.
After finishing the simulation, it was required to export a *VCD* file from *ModelSim* which include the switching power information, which is going to be used on *Innovus* to export the power report

from *Innovus*.
The power report for the unoptimized version shows the following:

	Internal power	*Switching power*	*Total power*	*Leakage power*
Total (2029 of 2029)	*0.7468*	*0.6942*	*1.526*	*0.08482*
Total Capacitance	*1.293e-11 F*			

While the power report for the optimized version shows the following:

	Internal power	*Switching power*	*Total power*	*Leakage power*
Total (2982 of 2982)	*1.181*	*0.8278*	*2.135*	*0.1266*
Total Capacitance	*1.8e-11 F*			

Power in the optimized version will increase since area has increased. However, for timing reports using the maximum clock divided by 4 exported from Innovus, the unoptimized version shows the following:

Path Groups: MY_CLK
Analysis View: MyAnView

Other End Arrival Time	*0.000*
- External Delay	*0.500*
+ Phase Shift	*24.000*
- Uncertainty	*0.070*
= Required Time	*23.430*
- Arrival Time	*4.764*
= Slack Time	*18.667*
Clock Rise Edge	*0.000*
+ Input Delay	*0.500*
= Beginpoint Arrival Time	*0.500*

While the optimized version shows the following:

Path Groups: MY_CLK
Analysis View: MyAnView

Other End Arrival Time	*0.000*
- Setup	*0.046*
+ Phase Shift	*12.000*
- Uncertainty	*0.070*
= Required Time	*11.884*
- Arrival Time	*2.617*
= Slack Time	*9.267*
Clock Rise Edge	*0.000*
+ Input Delay	*0.500*
= Beginpoint Arrival Time	*0.500*

In both cases, *Cadence* was able to optimize the design much better in area and timing point of view than Synopsys, allowing less area, and also allowing more slack.

5 Power Switching

The switching activity is used to check for the power consumption by generating some information using *ModelSim* in a file with .saif extension, using the help of *Synopsys* it was possible to generate the *NangateOpenCellLibrary.saif* as explained in the library technology thatis going to be used by *ModelSim* to generate the switching activity information for both versions, the optimized and the unoptimized. For generating this *.saif* file, it was required to simulate the filters, the optimized and unoptimized. Basically, the process is so similar for generating the *.saif* file.
First it was required to launch *ModelSim*, by executing these commands in the terminal in a directory where init_msim6.2g file is located in:

source init_msim6.2g
vsim

After launching *ModelSim*, a new project can be done and the necessary files can be loaded and compiled, and put their compilation information in the work directory; the files required where: the netlist exported by *Synopsys* where the hierarchy is combined as explained in Chapter 3.3.1, the testbench in which it is going to be modified to detect the switching activity and read the technology library and then it can export the *IIR.saif*, the clock generator, the data maker which contains the coefficients of the filters and the process of reading the samples, and the data sink which is used to export the results got in a text file.
Before compilation, the testbench modification will basically be by adding these lines before ending the module:

```
initial begin
        $read_lib_saif("./saif/NangateOpenCellLibrary.saif");
        $set_gate_level_        monitoring("on");
        $set_ toggle_ region(UUT);
        $toggle_start;
end
always @ ( END_SIM_i ) begin
        if (END _SIM _i) begin
        $toggle_stop; $
         toggle_ report("./saif/IIR_back.saif", 1.0e-9, "tb_ iir.UUT");
        end
end
```

These files can be compiled using the following commands:

vcom -93 -work ./work ./clk_gen.vhd
vcom -93 -work ./work ./data_maker_new.vhd
vcom -93 -work ./work ./data_sink.vhd
vlog -work ./work ./IIR.v
vlog -work ./work ./tb_iir.v

Then for simulating, the .sdf file which was generated by Synopsys as explained in chapter 3.3.1 has to loaded for the test bench since it contains delay information of the netlist. However, these commands were used to simulating and then generating the IIR.saif file:

vsim -L /software/dk/nangate45/verilog/msim6.2g work.tb_iir
vsim -L /software/dk/nangate45/verilog/msim6.2g -sdftyp /tb iir/UUT=./IIR.sdf work.tb_iir
vsim -L /software/dk/nangate45/verilog/msim6.2g -sdftyp /tb iir/UUT=./IIR.sdf -pli /soft-

ware/synopsys/syn_current/auxx/syn/power/vpower/lib-linux/libvpower.so work.tb_iir
It is required now to run the simulation through *ModelSim* to generate the results and the switching activity information can be exported too in the *IIR.saif* file.

In the optimized version the same was exactly done except naming the IIR entity IIR_adv, in both testbench and ModelSim. However, the results obtained were exactly the same for both versions. In Figures 23 and 24 samples of results for both unoptimized and optimized versions are shown, respectively. Both these samples and the entire set of samples match the VHDL and C output results for both filter versions.

Figure 23: Unoptimized version results in power-switching

Figure 24: optimized version results in power-switching

6 Post Power Switching

The same purpose done in Power Switching chapter is going to be done here as well, but this part is going to be more precious since it is going to take place after preparing the layout. *SAIF* file was generated after simulating using *ModelSim*. However, it is not going to change that much this time, except that, this time *VCD* (Value ChangeDump) will be generated to be used by *Cadence*, and the Testbench willremain the same as the original version.

After generating the netlist from *Cadence*, *ModelSim* will use it and use the *SDF* file as well to gen- erate the *VCD* file. These commands were used in ModelSim to start simulating the unoptimized version:

```
vcom -93 -work ./work ./clk_gen.vhd
vcom -93 -work ./work ./data_new.vhd
vcom -93 -work ./work ./data_sink.vhd
vlog -work ./work ./IIR_CA.v
vlog -work ./work ./tb_iir.v
vcd file ../vcd/design.vcd
vcd add /tb_iir/UUT/*
```

For the optimized version the same script was used except obviously changing the filter which is used. During this process the results after generating the netlist using *Cadence* were checked using *ModelSim*. In Figures 25 and 26, samples of the unoptimized version and the optimized version will be shown respectively. These samples, as well as the entire set of results, match the results obtained before from C and VHDL for both filter versions.

Figure 25: Unoptimized version results in Post Power-Switching

Figure 26: optimized version results in Post Power-Switching

7 Matlab filter comparison: non-optimized and optimized versions

A script called *comparison.m* was created to visually compare the Matlab output values. The previously mentioned output values are the output values of the reference filter model (2° IIR) and the optimized filter (3° IIR) after applying the Look-ahead Transform. Figure 27 on the next page illustrates both waveforms, normalized, as previously indicated. The vertical axis, horizontal axis, time range and time steps for this graph are the same as for the graphs in previous sections.

It's possible to see that both waveforms are very similar in amplitude, have the same frequency but appear to be slightly shifted. For this group's particular filter, this transformed the original filter into a 3rd order filter, with different coefficients. It's probably the difference in order, and also the value of the new coefficients, that most likely pro- duces the shift between both waveforms as seen in Figure 27.

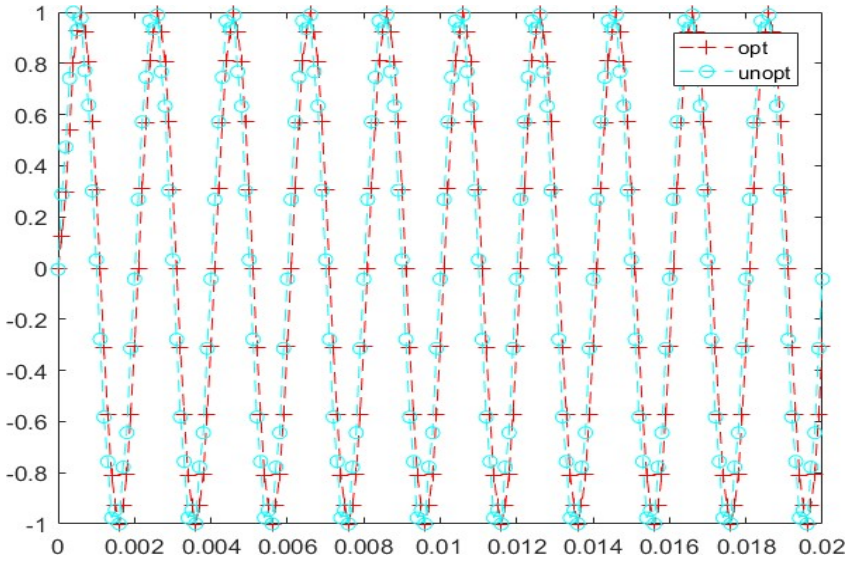

Figure 27: Optimized and non-optimized filter Matlab waveforms (normalized)

In addition to Figure 27, Figure 28 on the following page shows the comparison of both waveforms

again. However, this time, the waveforms are not normalized which makes it possible to better appreciate the difference of the Matlab output values of the non optimized and optimized versions. The reasons why it's thought that this difference occurs was already explained. As a final comment regarding this figure, this plot was built using the same script that was previously mentioned.

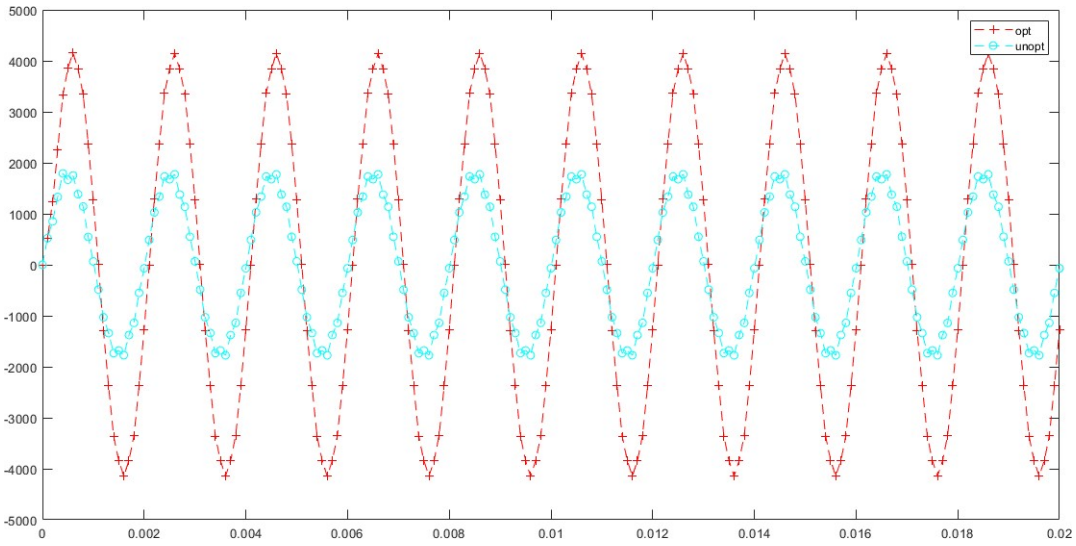

Figure 28: Optimized and non-optimized filter Matlab waveforms

Finally, the comparison of the output waveforms coming from VHDL/C is shown below in Figure 29. The same script was used. The only difference is that, for this case, .txt files containing the outputs from ModelSim (which were the same as the C outputs) were used to make the graph.

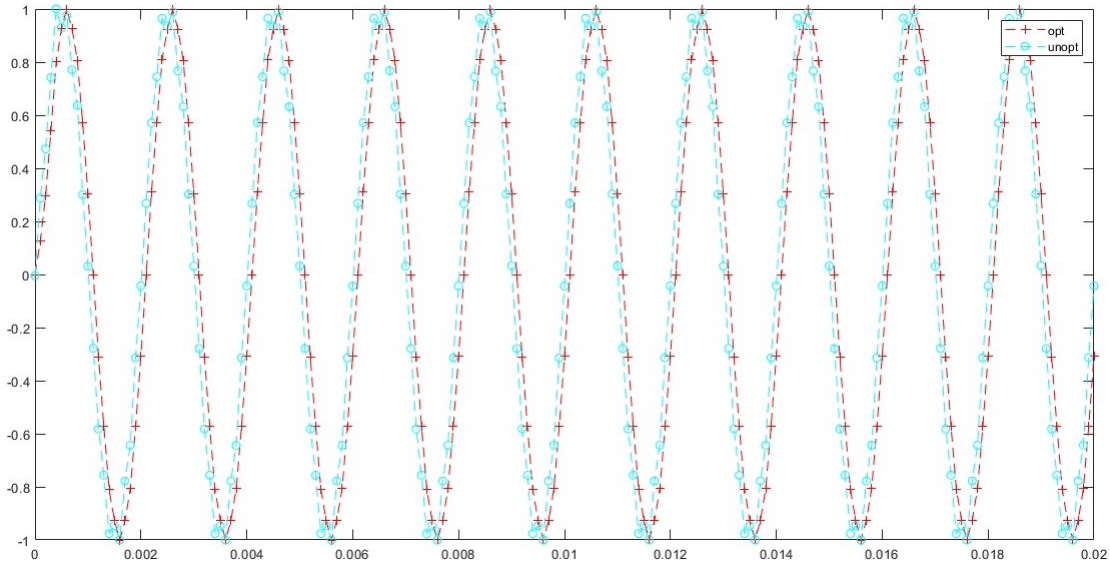

Figure 29: Optimized and non-optimized filter VHDL/C waveforms

Figure 29 shows the output waveform of the optimized filter in red, and the output waveform of the reference model in light blue. Just like before, these are extremely similar, and their difference (phase

shift) might be explained by the same reasons as indicated before when discussing Figure 27.Like in this first case, the VHDL/C results seem acceptable.

Report on Laboratory 2 will deal with the non-optimized and optimized output comparisons in more detail when comparing the VHDL results to the optimized filter using different multiplier architectures.

8 Conclusion

In conclusion, it can be said that the goal of designing a reference digital filter model and its advanced architecture counterpart, using the Look-ahead Transform and other universal optimization techniques, was met. In particular, for both cases, the output results obtained in Matlab, C and VHDL were as expected. C and VHDL results were verified to match and the error between the Matlab results and the C/VHDL pairs was small as expected (either 2 least significant digits in base 10 for the non optimized case and 3 for the optimized filter). As explained, this is due to the way in which integer values were built in Matlab in comparison to C and VHDL.

Moreover, the VHDL design proved to be adequate in terms of obtained results. This is due to the fact that simulations carried out using ModelSim matched the behavior obtained in C, as laboratory teams were instructed to achieve during the laboratory sessions. Furthermore, for both cases, activities related to synthesis, place and routing, power switching and post power switching were carried out successfully and gave good results, as already discussed extensively and in detail in the report. Just to highlight an important achieved result, for example, it was possible to go from a clock pe- riod of 6 ns for the non optimized version to 3 ns for the optimized version, having a slack value of 0. Finally, the result difference obtained between the original 2nd Order IIR Filter in DFII and the optimized, 3rd Order IIR Filter in DFII can be explained due to the different coefficient values between both versions and the new filter order. However, as it was mentioned, a more in depth behavior analysis of the optimized filter will be carried out in Laboratory 2 when designing different multiplier architectures. In this case, all these results will be compared to the original output results of the non optimized filter from C/VHDL, as well as the optimized one (inVHDL/C).

9 Appendixes

9.1 Appendix A. VHDL: reference filter

```
library ieee;
use ieee.std_logic_1164.all; use
ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;use
ieee.std_logic_textio.all; use
ieee.std_logic_signed.all;

library std;
use  std.textio.all;

entity IIR_CA is
  port (
    CLK       : in    std_logic;
    RST_n     : in    std_logic;
         VIN       : in  std_logic;
         DIN                    : in    std_logic_vector(13 downto 0); --14 bit data input
         B0                     : in    std_logic_vector(13 downto 0);
         B1                     : in    std_logic_vector(13 downto 0);
         B2                     : in    std_logic_vector(13 downto 0);
         A1                     : in    std_logic_vector(13 downto 0);
         A2                     : in    std_logic_vector(13 downto 0);VOUT
              : out std_logic;
    DOUT      : out std_logic_vector(13 downto 0));end

IIR_CA;

architecture bhv of IIR_CA is
         signal fb             : signed(13 downto 0);
         signal ff             : signed(13 downto 0);
         signal w              : signed(13 downto 0); signal
         reg1                    : signed(13 downto 0);
         signal reg2             : signed(13 downto 0);
                        signal temp1: signed(27 downto 0); --If we have n bit the multiplication can
                                                           --leads to 2*n bits.
         signal temp2: signed(27 downto 0); signal
         temp3: signed(27 downto 0); signal temp4:
         signed(27 downto 0); signal temp5:
         signed(27 downto 0); signal mul1 :
         signed(13 downto 0); signal mul2 :
         signed(13 downto 0); signal mul3 :
         signed(13 downto 0); signal mul4 :
         signed(13 downto 0); signal mul5 :
         signed(13 downto 0);
         signal xin: std_logic_vector(13 downto 0);
```

```
        signal DOUT_temp: std_logic_vector(13 downto 0);
begin
        temp1 <= reg1*signed(A1);--multi mul1<=
        temp1(26 downto 13);--shifttemp2 <=
        reg2*signed(A2);
        mul2    <= temp2(26 downto 13); --taking only 14 bit of MSB
        temp3 <= reg1*signed(B1); mul3
                <= temp3(26 downto 13);
        temp4 <= reg2*signed(B2); mul4
                <= temp4(26 downto 13);
        temp5 <= w*signed(B0);
        mul5    <= temp5(26 downto 13);fb
        <= -mul1-mul2;--feedback ff <=
        mul3+mul4;--feedforwardw
                <= signed(xin)+fb;
        DOUT_temp <= conv_std_logic_vector(ff+mul5,14); --Converting it to vector 14 bits
        DOUT <= DOUT_temp(13 downto 0);
        process(CLK,RST_n,VIN)
                begin
                if RST_n = '0' then -- Everything 0 if reset
                        reg1 <= (others => '0'); reg2
                        <= (others => '0'); VOUT  <=
                        '0';
                        xin    <= (others => '0');
                elsif CLK'event and CLK = '1' and VIN = '1' thenVOUT <= '0';
                        xin(13 downto 0) <= DIN;
                        reg2 <= reg1; -- In proces this will model after a FF so no
                                                        --need to add an extra FF.
                        reg1 <= w;
                        VOUT <= '1';
                end if;
        end process;
end bhv;
```

9.2 Appendix B. VHDL and Verilog testbench: reference filter

```
//`timescale 1ns

module tb_iir ();

    wire CLK_i; wire
    RST_n_i;
    wire [13:0] DIN_i;
    wire VIN_i;
    wire [13:0] H0_i;
    wire [13:0] H1_i;
    wire [13:0] H2_i;
    wire [13:0] H3_i;
    wire [13:0] H4_i;
    wire [13:0] DOUT_i;wire
    VOUT_i;
    wire END_SIM_i;

    clk_gen CG(.END_SIM(END_SIM_i),
                    .CLK(CLK_i),
                .RST_n(RST_n_i));

    data_maker SM(.CLK(CLK_i),
                    .RST_n(RST_n_i),
                    .VOUT(VIN_i),
                    .DOUT(DIN_i),
                    .H0(H0_i),
                    .H1(H1_i),
                    .H2(H2_i),
                    .H3(H3_i),
                    .H4(H4_i),
                    .END_SIM(END_SIM_i));

    IIR_CA UUT(.CLK(CLK_i),
                .RST_n(RST_n_i),
                .DIN(DIN_i),
                .VIN(VIN_i),
                .B0(H0_i),
                .B1(H1_i),
                .B2(H2_i),
                .A1(H3_i),
                .A2(H4_i),
                .DOUT(DOUT_i),
                .VOUT(VOUT_i));

    data_sink DS(.CLK(CLK_i),
                    .RST_n(RST_n_i),
```

```
                .VIN(VOUT_i),
                .DIN(DOUT_i));

    initial begin
        $read_lib_saif("./saif/NangateOpenCellLibrary.saif");
        $set_gate_level_monitoring("on");
        $set_toggle_region(UUT);
        $toggle_start;
    end

    always @ ( END_SIM_i ) beginif
        (END_SIM_i) begin
            $toggle_stop;
            $toggle_report("./saif/IIR_back.saif", 1.0e-9, "tb_iir.UUT");
        end
    end

endmodule
```

9.3 Appendix C. VHDL: optimized filter

9.3.1 Main File

```
--file edited by rashid on 04/11/2018
--Edited by sawan on 05/11/2018
library ieee;
use ieee.std_logic_1164.all; use
ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;use
ieee.std_logic_textio.all; use
ieee.std_logic_signed.all;

library std;
use std.textio.all;

entity IIR_ADV is
  port (
              CLK       : in    std_logic;
         RST_n     : in std_logic;
         VIN       : in    std_logic;
         DIN          : in    std_logic_vector(13 downto 0);B0
                    : in    std_logic_vector(13 downto 0); B1
                    : in    std_logic_vector(13 downto 0); B2
                    : in    std_logic_vector(13 downto 0); B3
                    : in    std_logic_vector(13 downto 0); A1
                    : in    std_logic_vector(13 downto 0); A2
                    : in    std_logic_vector(13 downto 0);
              VOUT      : out std_logic;
              DOUT      : out std_logic_vector(13 downto 0));end
IIR_ADV;

architecture bhv of IIR_ADV is

  component reg
port(
      RST_n      : in      std_logic;
     VIN         : in      std_logic;
     d     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);CLK :
     IN STD_LOGIC; -- clock.
     q     : OUT STD_LOGIC_VECTOR(13 DOWNTO 0) -- output
);
END component;

component mulxport
(
     mult1 : in std_logic_vector (13 downto 0); mult2 : in
     std_logic_vector   (13   downto   0);   prod:   out
     std_logic_vector (13 downto 0));
```

```
end component;

component      sumx
port (
      add1 : in std_logic_vector (13 downto O); add2 : in
      std_logic_vector  (13  downto  O);  sum1:      out
      std_logic_vector (13 downto 0));
end component;

component      subx
   port (
      sub1 : in std_logic_vector (13 downto 0); sub2 : in
      std_logic_vector  (13  downto  0);  rel1:    out
      std_logic_vector (13 downto 0));
end component;

            signal              fb            : STD_LOGIC_VECTOR(13 downto O); signal
                               ff             : STD_LOGIC_VECTOR(13 downto O); signal
                               w              : STD_LOGIC_VECTOR(13 downto O); signal
                               reg2             : STD_LOGIC_VECTOR(13 downto 0); signal
                               reg3             : STD_LOGIC_VECTOR(13 downto 0); signal
                               reg4             : STD_LOGIC_VECTOR(13 downto 0); signal
                               reg5             : STD_LOGIC_VECTOR(13 downto 0); signal
                               reg6             : STD_LOGIC_VECTOR(13 downto 0); signal
                               reg7             : STD_LOGIC_VECTOR(13 downto 0); signal
                               reg1_reg              :  std_logic_vector(13 downto O);signal
                                reg2_reg             :  std_logic_vector(13 downto O);signal
                                reg3_reg             :  std_logic_vector(13 downto O);signal
                                reg4_reg             :  std_logic_vector(13 downto O);signal
                               temp1            : STD_LOGIC_VECTOR(13 downto 0); signal
                               temp2              : STD_LOGIC_VECTOR(13 downto 0); signal
                               temp3              : STD_LOGIC_VECTOR(13 downto 0); signal
            temp4           : STD_LOGIC_VECTOR(13 downto 0); signal          temp5          :
            STD_LOGIC_VECTOR(13 downto 0); signal          temp6          :
            STD_LOGIC_VECTOR(13  downto  0);
            signal              mul1_reg             : std_logic_vector(13 downto 0); signal
                               mul2_reg             : std_logic_vector(13 downto 0); signal
                               mul3_reg             : std_logic_vector(13 downto 0); signal
                               mul4_reg             : std_logic_vector(13 downto 0); signal
                               xin                    : std_logic_vector(13 downto 0);signal
            sig_ff1: std_logic_vector(13 downto 0);
begin         signal sig_ff2: std_logic_vector(13 downto 0);



            REGISTER1: reg port map (RST_n,VIN,w,CLK,reg1_reg);p6: mulx port
            map(reg1_reg, B0, temp1);
```

```
        REG1_1: reg port map (RST_n,VIN,temp1,CLK,mul1_reg);          --After the register
                                                --which is after B0 coefficient

        REGISTER2: reg port map (RST_n,VIN,reg2, CLK ,reg2_reg);p7: mulx
        port map(reg2_reg, B1, temp2);
        REG2_1: reg port map (RST_n,VIN,temp2,CLK,mul2_reg);          --After the register
                                                --which is after B1 coefficient

        REGISTER3: reg port map (RST_n,VIN,reg3,CLK,reg3_reg);p8: mulx port
        map(reg3_reg, B2, temp3);
        REG3_1: reg port map (RST_n,VIN,temp3,CLK,mul3_reg);          --After the register
                                                --which is after B2 coefficient

        REGISTER4: reg port map (RST_n,VIN,reg5,CLK,reg4_reg);p9: mulx port
        map(reg4_reg, B3, temp4);
        REG4_1: reg port map (RST_n,VIN,temp4,CLK, mul4_reg);

        p11: mulx port map(reg2, A1, temp6);p10:
        mulx port map(reg4,A2, temp5);SUB1: subx port
        map(reg6,reg7,fb);

        s6:sumx port map(mul1_reg,mul2_reg,sig_ff1); s7:sumx
        port map(mul3_reg,mul4_reg,sig_ff2); s8:sumx port
        map(sig_ff1,sig_ff2, ff);
        s9: sumx port map(fb,xin,w); DOUT
        <= ff;

        REG11: reg port map (RST_n,VIN,w,CLK,reg2); REG21: reg port
          map (RST_n,VIN,reg2,CLK,reg3); REG31: reg port map
          (RST_n,VIN,reg2,CLK,reg4); REG41: reg port map
          (RST_n,VIN,reg4,CLK,reg5);
           REG51: reg port map (RST_n,VIN,temp5,CLK,reg6);REG61:
           reg port map (RST_n,VIN,temp6,CLK,reg7);

        process(CLK,RST_n,VIN) begin
                if RST_n = '0' then
                        VOUT <= '0';
                        xin  <= (others => '0');
                elsif CLK'event and CLK = '1' and VIN = '1' thenVOUT <=
                        '0';
                        xin(13 downto 0) <= DIN;VOUT
                        <= '1';
                end if;
        end process;
end bhv;
```

9.3.2 Multiplexer

```
library ieee;
use ieee.std_logic_1164.all; use
ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;use
ieee.std_logic_textio.all; use
ieee.std_logic_signed.all;

library std;
use  std.textio.all;

entity   mulx   is
  port  (
     mult1 : in std_logic_vector (13 downto O); mult2 : in
     std_logic_vector   (13   downto   O);   prod:   out
     std_logic_vector ( 13 downto O));
end mulx;

architecture str1 of mulx is
signal temp:std_logic_vector (27 downto 0); begin
     temp <= SIGNED (mult1)*SIGNED(mult2);prod <=
     temp(26 downto 13);
end str1;
```

9.3.3 Register

```
library ieee;
use ieee.std_logic_1164.all; use
ieee.std_logic_arith.all; use
ieee.std_logic_textio.all;use
ieee.std_logic_signed.all;ENTITY reg
IS PORT(
          RST_n     : in    std_logic;
          VIN       : in    std_logic;
          d     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);CLK :
          IN STD_LOGIC; -- clock.
          q     : OUT STD_LOGIC_VECTOR(13 DOWNTO 0) -- output
);
END reg;

ARCHITECTURE description  OF reg ISBEGIN
     process(RST_n,VIN,CLK) begin
          if RST_n = 'O' then
                    q <= (others => 'O');
```

```
        elsif CLK'event and CLK = '1' and VIN = '1' thenq <= d;
        end if;
    end process;
END  description;
```

9.3.4 Subtractor

```
  library ieee;
use ieee.std_logic_1164.all; use
ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;use
ieee.std_logic_textio.all; use
ieee.std_logic_signed.all;

library std;
use  std.textio.all;

entity  subx  is
  port (
    sub1 : in std_logic_vector (13 downto 0); sub2 : in
    std_logic_vector  (13  downto  0);  rel1:  out
    std_logic_vector (13 downto 0));
end subx;

architecture str_sub of subx is
  --signal rel_temp:std_logic_vector (14 downto 0);
  begin
    rel1 <= - SIGNED (sub1) - SIGNED(sub2);
    --rel1 <= rel_temp (14 downto 1);
end str_sub;
```

9.3.5 Addition

```
library ieee;
use ieee.std_logic_1164.all; use
ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;use
ieee.std_logic_textio.all; use
ieee.std_logic_signed.all;

library std;
use  std.textio.all;

entity  sumx  is
  port (
    add1 : in std_logic_vector (13 downto O); add2 : in
    std_logic_vector  (13  downto  O);  sum1:  out
    std_logic_vector (13 downto 0));
```

```
end sumx;

architecture str of sumx is
  --signal sum_temp:std_logic_vector (14 downto 0);
  begin
     sum1 <= SIGNED (add1)+ SIGNED(add2);
     --sum1 <= sum_temp (14 downto 1);
end str
```

9.4 Appendix D. VHDL and Verilog testbench: optimized filter

```
//`timescale 1ns

module tb_iir_adv ();wire

    CLK_i;
    wire RST_n_i;
    wire [13:0] DIN_i;
    wire VIN_i;
    wire [13:0] H0_i;
    wire [13:0] H1_i;
    wire [13:0] H2_i;
    wire [13:0] H3_i;
    wire [13:0] H4_i;
    wire [13:0] H5_i;
    wire [13:0] DOUT_i;wire
    VOUT_i;
    wire END_SIM_i;

    clk_gen CG(.END_SIM(END_SIM_i),
                    .CLK(CLK_i),
                .RST_n(RST_n_i));

    data_maker SM(.CLK(CLK_i),
                     .RST_n(RST_n_i),
                     .VOUT(VIN_i),
                     .DOUT(DIN_i),
                     .H0(H0_i),
                     .H1(H1_i),
                     .H2(H2_i),
                     .H3(H3_i),
                     .H4(H4_i),
                     .H5(H5_i),
                     .END_SIM(END_SIM_i));

    IIR_ADV UUT(.CLK(CLK_i),
                .RST_n(RST_n_i),
                .DIN(DIN_i),
                .VIN(VIN_i),
                .B0(H0_i),
                .B1(H1_i),
                .B2(H2_i),
                .B3(H3_i),
                .A1(H4_i),
                .A2(H5_i),
                .DOUT(DOUT_i),
                .VOUT(VOUT_i));
```

```
    data_sink DS(.CLK(CLK_i),
                 .RST_n(RST_n_i),
                 .VIN(VOUT_i),
                 .DIN(DOUT_i));

    initial begin
        $read_lib_saif("./saif/NangateOpenCellLibrary.saif");
        $set_gate_level_monitoring("on");
        $set_toggle_region(UUT);
        $toggle_start;
    end

    always @ ( END_SIM_i ) beginif
        (END_SIM_i) begin
            $toggle_stop;
            $toggle_report("./saif/IIR_back.saif", 1.0e-9, "tb_iir_adv.UUT");
        end
    end

endmodule
```

9.5 Appendix G. Matlab *myiir_design.m* script (optimized filter)

```
function [bi, ai, bq, aq]=myiir_design(N,nb)
%%  function  myiir_design(N,nb)
%% N is order of the filter
%% nb is the number of bits
%% bi,ai taps represented as integers
%% bq,aq  quantized  taps

close all;

f_cut_off = 2000; % 1kHz
f_sampling = 10000; %  10kHz

f_nyq = f_sampling/2; %% Nyquist frequenc
f0 = f_cut_off/f_nyq; %% Normalized cut-off frequency

[b,a]=butter(2, f0); %% get filter coefficients
[h1, w1]=freqz(b,a); %% %% get the transfer function of the designed filter

bi=floor(b*2^(nb-1));
ai=floor(a*2^(nb-1));

bq=bi/2^(nb-1); %% convert back b coefficients as nb-bit real values
aq=ai/2^(nb-1);

a0=aq(1,1);
a1=aq(1,2);
a2=aq(1,3)+(aq(1,2)*aq(1,2));
a3=aq(1,2)*aq(1,3);

b0=bq(1,1);
b1=bq(1,2)+(aq(1,2)*bq(1,1));
b2=bq(1,3)+(aq(1,2)*bq(1,2));
b3=aq(1,2)*bq(1,3);

bq=[b0,b1,b2,b3];
aq=[a0,0,a2,a3];

bi=floor(bq*2^(nb-1)); %% convert back b coefficients as nb-bit real values
ai=floor(aq*2^(nb-1))

[h2, w2]=freqz(bq,aq); %% get the transfer function of the quantized filter

%% show the transfer functions
plot(w1/pi, 20*log10(abs(h1)));hold on;
plot(w2/pi,  20*log10(abs(h2)),'r--');
```

```
grid on;
xlabel('Normalized frequency');
ylabel('dB');
```

9.6 Appendix H. Matlab *comparison.m* script

```
fs=10000 %% sampling frequency
f1=500;     %% first sinewave freq (in band)
f2=4500; %% second sinnewave freq (out band)


T=1/500; %% maximum period
tt=0:1/fs:10*T; %% time samples

x1=sin(2*pi*f1*tt); %% first sinewave
x2=sin(2*pi*f2*tt); %% second sinewave

x=(x1+x2)/2; %% input signal

Y = dlmread('results_optimized.txt');
Y2 = 2 * (Y - min(Y))/(max(Y) - min(Y)) -1;M =
dlmread('results_unoptimized.txt');
M2 = 2 * (M - min(M))/(max(M) - min(M)) -1;

%% plots
figure
plot(tt,M2, 'r--+');hold
on
plot(tt, Y2, 'c--o');hold on

legend('opt', 'unopt')hold
off
```

9.7 Appendix I. Reference 2nd Order IIR Filter: output results (C)

i	Value	*i*	Value	*i*	Value
0	0	40	-1269	80	-1269
1	522	41	-4	81	-4
2	1238	42	1293	82	1293
3	2245	43	2370	83	2370
4	3324	44	3361	84	3361
5	3848	45	3839	85	3839
6	4154	46	4144	86	4144
7	3841	47	3840	87	3840
8	3343	48	3344	88	3344
9	2375	49	2375	89	2375
10	1268	50	1268	90	1267
11	3	51	3	91	2
12	-1293	52	-1293	92	-1294
13	-2371	53	-2371	93	-2371
14	-3362	54	-3362	94	-3361
15	-3841	55	-3841	95	-3839
16	-4147	56	-4147	96	-4145

17	-3842	57	-3842	97	-3841
18	-3347	58	-3347	98	-3347
19	-2377	59	-2377	99	-2377
20	-1270	60	-1270	100	-1269
21	-5	61	-5	101	-4
22	1293	62	1293	102	1293
23	2370	63	2370	103	2370
24	3361	64	3361	104	3361
25	3839	65	3839	105	3839
26	4144	66	4144	106	4144
27	3840	67	3840	107	3840
28	3344	68	3344	108	3344
29	2375	69	2375	109	2375
30	1267	70	1267	110	1267
31	2	71	2	111	2
32	-1295	72	-1295	112	-1295
33	-2372	73	-2372	113	-2372
34	-3362	74	-3362	114	-3362
35	-3839	75	-3839	115	-3839
36	-4145	76	-4145	116	-4145
37	-3841	77	-3841	117	-3841
38	-3347	78	-3347	118	-3347
39	-2377	79	-2377	119	-2377

i	Value	*i*	Value	*i*	Value
120	-1269	160	-1270	200	-1269
121	-4	161	-5	-	-
122	1293	162	1293	-	-
123	2370	163	2370	-	-
124	3361	164	3361	-	-
125	3839	165	3839	-	-
126	4144	166	4144	-	-
127	3840	167	3840	-	-
128	3345	168	3345	-	-
129	2376	169	2376	-	-
130	1268	170	1268	-	-
131	2	171	2	-	-
132	-1294	172	-1295	-	-
133	-2371	173	-2372	-	-
134	-3361	174	-3362	-	-
135	-3839	175	-3839	-	-
136	-4145	176	-4145	-	-
137	-3841	177	-3841	-	-
138	-3347	178	-3347	-	-
139	-2377	179	-2377	-	-
140	-1269	180	-1269	-	-
141	-4	181	-4	-	-
142	1293	182	1293	-	-
143	2370	183	2370	-	-
144	3361	184	3361	-	-
145	3839	185	3839	-	-
146	4144	186	4144	-	-
147	3840	187	3840	-	-
148	3344	188	3345	-	-
149	2375	189	2376	-	-
150	1268	190	1269	-	-
151	3	191	3	-	-
152	-1293	192	-1293	-	-
153	-2371	193	-2371	-	-
154	-3362	194	-3361	-	-
155	-3841	195	-3839	-	-
156	-4147	196	-4145	-	-
157	-3843	197	-3842	-	-
158	-3347	198	-3347	-	-
159	-2377	199	-2377	-	-

9.8 Appendix J. Optimized 3rd Order IIR Filter: output results (C)

i	Value	*i*	Value	*i*	Value
0	0	40	-69	80	-69
1	522	41	485	81	485
2	852	42	1027	82	1027
3	1331	43	1336	83	1336
4	1791	44	1728	84	1728
5	1666	45	1677	85	1677
6	1749	46	1772	86	1772
7	1383	47	1376	87	1375
8	1143	48	1135	88	1135
9	544	49	547	89	547
10	64	50	65	90	65
11	-487	51	-489	91	-489
12	-1030	52	-1030	92	-1029
13	-1340	53	-1340	93	-1340
14	-1732	54	-1732	94	-1732
15	-1679	55	-1680	95	-1679
16	-1775	56	-1775	96	-1775
17	-1379	57	-1379	97	-1379
18	-1139	58	-1139	98	-1139
19	-552	59	-552	99	-552
20	-69	60	-69	100	-69
21	485	61	485	101	485
22	1027	62	1027	102	1027
23	1336	63	1336	103	1336
24	1728	64	1728	104	1728
25	1677	65	1677	105	1676
26	1772	66	1772	106	1772
27	1375	67	1376	107	1376
28	1135	68	1135	108	1135
29	547	69	547	109	548
30	65	70	65	110	66
31	-489	71	-489	111	-489
32	-1030	72	-1030	112	-1030
33	-1340	73	-1340	113	-1340
34	-1732	74	-1732	114	-1732
35	-1680	75	-1680	115	-1680
36	-1775	76	-1775	116	-1775
37	-1379	77	-1379	117	-1379
38	-1139	78	-1139	118	-1139
39	-552	79	-552	119	-552

i	Value	*i*	Value	*i*	Value
120	-69	160	-69	200	-69
121	485	161	485	-	-
122	1027	162	1027	-	-
123	1336	163	1336	-	-
124	1729	164	1729	-	-
125	1677	165	1677	-	-
126	1772	166	1772	-	-
127	1376	167	1376	-	-
128	1136	168	1136	-	-
129	547	169	547	-	-
130	65	170	65	-	-
131	-489	171	-489	-	-
132	-1029	172	-1030	-	-
133	-1340	173	-1340	-	-
134	-1732	174	-1732	-	-
135	-1679	175	-1679	-	-
136	-1775	176	-1775	-	-
137	-1379	177	-1379	-	-
138	-1139	178	-1139	-	-
139	-552	179	-552	-	-
140	-69	180	-69	-	-
141	485	181	485	-	-
142	1027	182	1027	-	-
143	1336	183	1336	-	-
144	1728	184	1728	-	-
145	1677	185	1676	-	-
146	1772	186	1772	-	-
147	1375	187	1376	-	-
148	1135	188	1136	-	-
149	547	189	548	-	-
150	65	190	66	-	-
151	-489	191	-489	-	-
152	-1030	192	-1030	-	-
153	-1340	193	-1340	-	-
154	-1732	194	-1732	-	-
155	-1680	195	-1679	-	-
156	-1775	196	-1775	-	-
157	-1379	197	-1379	-	-
158	-1139	198	-1139	-	-
159	-552	199	-551	-	-