Politecnico di Torino

Integrated System Architecture
Design and implementation of a digital filter

First Lab Report

`https://github.com/aledolme19/ISA_LAB.git`

Francesco Babbaro Pierfrancesco Boccardi Alessandra Dolmeta

2021-2022

1 Introduction

The aim of the first laboratory experience is to develop and implement a synthesizable and testable IIR (Infinite Impulse Response) Filter using hardware description languages.
An IIR filter is less complex respect to a FIR one. Indeed, less coefficients than the FIR have to be used to obtain the same behaviour. It is also faster, but differently from a generic FIR, it can be unstable.
The goal of the first laboratory assignment is to use MATLAB and C language models for IIR filter of the second order, test their functionality and develop the same in VHDL.
It is also required to simulate the developed IIR, synthesize it and perfom the place and route operations using cadence tool.
The tools used in the process are:

- **MATLAB**: the MATLAB script for general filter that provides us the right coefficients for a 2nd order IIR filter working on 9 bits, according to the group members' surnames, with a cut-off frequency of 2 kHz
- **C Language**: a general filter model written in C language to behave like MATLAB model, but restricting the number of bit employed to represent signals, in order to have a model closer to VHDL implementation;
- **ModelSim Altera**: VHDL files and testbench with the same functionality of the previous script, simulated in ModelSim Altera to verify their correct behaviour;
- **Synopsys Design Compiler**: logic synthesis of the developed model of filter, performing switching activity based power consumption and area and timing analysis;
- **Cadence Innovus**: the final design was then implemented on silicon using Cadence Innovus tool, which generates the final order of circuit after placing and routing the real library cells.

2 Filter Description

In this first laboratory, we need to design a digital filter with a cut-off frequency of 2 kHz, setting the sampling frequency to 10 kHz.
According to the specification required, given by group surnames, the characteristics of the filters are the following:

- Type: **IIR**
- Order of the filter: **II**
- Number of both I/O bits and coefficients bits: **9**

The schematic is shown in Figure 1 [1].

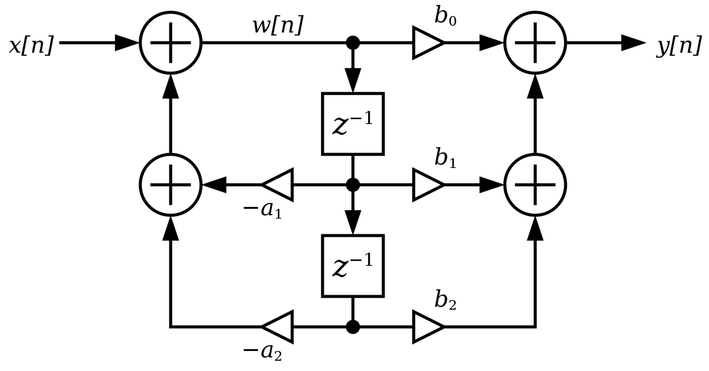


Figure 1: IIR Filter order II

[1]Reference for the figure: Digital biquad filter

3 Reference Model Development

In this first part it's required to design the filter with MATLAB and develop the fixed point model as a C program. Then, we need to evaluate the **THD** parameter (Total Harmonic Distortion), and make sure that it does not become higher than −30 dB, which is the only requirement which is asked.
In the last subsection of this part, the two sets of results are compared and commented.

3.1 MATLAB Model

The MATLAB script of IIR filter has been provided and it has been modified according to the parameters required.
Exploiting MATLAB function *butter*, as shown in the listing below, we obtain the IIR filter's coefficient required for the design, providing as parameters the order of the filter and the cut-off frequency required.
Using these coefficients the real transfer function of the IIR filter can be obtained. However, considering a limited precision for the coefficients (only 9 bits in our case), also the quantized version was considered in order to get a filter behaviour closer to the one will be achieved in hardware implementation.

```
%IIR design
f_cut_off = 2000; % 2kHz
f_sampling = 10000; % 10kHz
f_nyq = f_sampling/2; %% Nyquist frequency
f0 = f_cut_off/f_nyq; %% Normalized cut-off frequency

[b,a]=butter(N, f0); %% get filter coefficients
[h1, w1]=freqz(b,a); %% %% get the transfer function of the designed filter
bi=floor(b*2^(nb-1)); %% convert b coefficients into nb-bit integers
bq=bi/2^(nb-1); %% convert back b coefficients as nb-bit real values
ai=floor(a*2^(nb-1)); %% convert a coefficients into nb-bit integers
aq=ai/2^(nb-1); %% convert back a coefficients as nb-bit real values

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

The transfer function of the desired IIR filter and the transfer function of the quantized filter are shown in Figure 2. The difference between the two is related to **precision**. In fact, MATLAB uses double precision to do all computations, but the C code instead works with a finite precision.

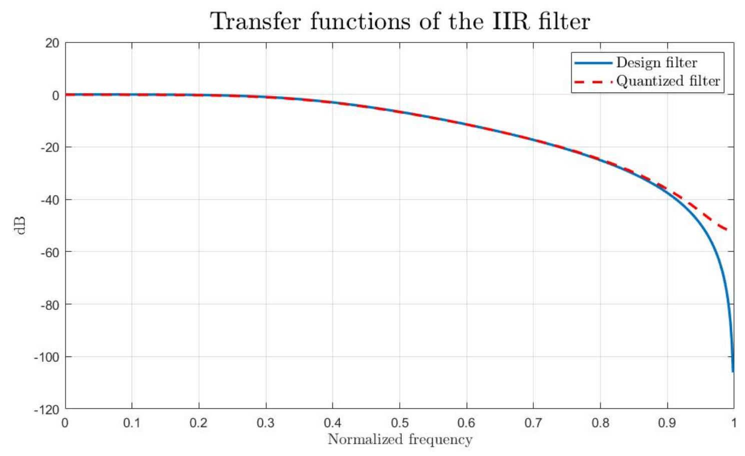


Figure 2: Transfer functions of the IIR filter

For instance, the coefficients that we are going to use are the quantized ones. The behaviour of the filter will not be exactly the desired one, but if we are able to satisfy the requirement on the total harmonic distortion, the specification will be fulfilled.

3.2 C Model

Once we have found and defined the coefficient of the IIR filters, we have modified the fixed point C language model provided, in order to make it behaves the same as the MATLAB model. Moreover, the C model must be written so that it behaves as much as possible as the VHDL model that we are going to actually implement in the Section 4.
The two main requirements we need to fulfil are:

- THD equal to maximum of -30 dB;
- minimize as much as possible the internal architecture of the filter.

The modified C language model is reported below, explaining in the details our choice.

First of all, we need to underline that the default integer format in a C-program usually works with a number of bits higher than the required one (32 bits in some processors). Consequently, each simulation performed with a C-program will give us different results respect to the ones expected, which should be referred to the parallelism of the inputs and the outputs of our filter, so on 9 bits. Therefore, to make C simulation coherent with the VHDL structure, we implemented a function, labelled as `saturation`. This function receives two parameters:

- the integer number that must be controlled;
- the parallelism we want to use.

In this way it is possible to handle integer number in C as if they had the parallelism we chose.

```c
int saturation(int number, int paralellism) {
  int sat_number;
  int max_pos_number = pow(2, paralellism - 1) - 1;
  int max_neg_number = -pow(2, paralellism - 1);

  if(number>max_pos_number) {
    sat_number = max_pos_number;
    printf("Overflow (positive)\n");
  }
  else if(number<max_neg_number) {
    sat_number = max_neg_number;
    printf("Overflow (negative)\n");
  }
  else {
    sat_number = number;
  }

  return sat_number;
}
```

As shown in the code above, `saturation` function first evaluate the maximum and the minimum number which are representable with the given input parallelism, and then compare the number in input with these two values: if the number exceeds the limits, it is saturated (to the lower or the larger value, depending on its sign). Moreover, when the condition of overflow is reached, it prints a message. This is done in order to be sure that the parallelism chosen can be properly used in the hardware description language environment without errors, being sure that the THD obtained in C simulation is respected.
Moreover, we decided also not to truncate the different samples in input. This would lead to a THD closer to the one required, but it would alter the outputs of the filter: we decide that it doesn't worth the prize. The same procedure has

been followed for the different input coefficients: eliminate some of the LSB bit of the coefficient will lead to a lower area and a lower THD, but the frequency response of the filter will be alter. This is why we decide to keep each of the input in the format given.
Here is shown instead the most relevant part of the C-function that actually permed the expression of our IIR filter (Direct Form II), which is:

$$w[n] = x[n] - a_1 w[n-1] - a_2 w[n-2]$$

$$y[n] = b_0 w[n] + b_1 w[n-1] + b_2 w[n-2]$$

```c
#define N 2 /// order of the filter
#define NB 9  /// number of bits
#define NB_MULT 18 // multiplier parallelism 2.16
#define NB_FIRST_ADD 10 // first (upper-left) adder parallelism 2.8
#define NB_ADD 8 // other adder parallelism 2.6
#define N_TRUNC_MULT 10

int myfilter(int x)
{
  {...}

  /// compute feed-back and feed-forward
  fb = 0;
  ff = 0;
  for (i=0; i<N; i++) {
    fb_mult = sw[i]*-a[i];
    fb_mult = saturation(fb_mult, NB_MULT);
    fb += (fb_mult >> N_TRUNC_MULT);
    fb = saturation(fb, NB_ADD);

    ff_mult = sw[i]*b[i];
    ff_mult = saturation(ff_mult, NB_MULT);
    ff += (ff_mult >> N_TRUNC_MULT);
    ff = saturation(ff, NB_ADD);
  }

  /// compute intermediate value (w) and output sample
  w = x + (fb << (NB_FIRST_ADD - NB_ADD));
  w = saturation(w, NB_FIRST_ADD);
  b0_mult = w*b0;
  b0_mult = saturation(b0_mult, NB_MULT);

  y = b0_mult >> N_TRUNC_MULT;
  y += ff;
  y = y << (NB_FIRST_ADD - NB_ADD);
  y = saturation(y, NB);

  {...}
}
```

The IIR filter functions has been splitted in different operations, in order to simulate the behaviour of the filter in VHD-language and to understand which are the minimum number of bits that are necessary in the internal structure, exploiting the `saturation` function and reminding the two requirements discussed previously. We have decided to use different parallelism inside the architecture, in order to reduce as much as possible the overall area, a not being forced to use the worst scenario for all the possible interconnection.

The final choice in terms of parallelism are shown in Figure 3.

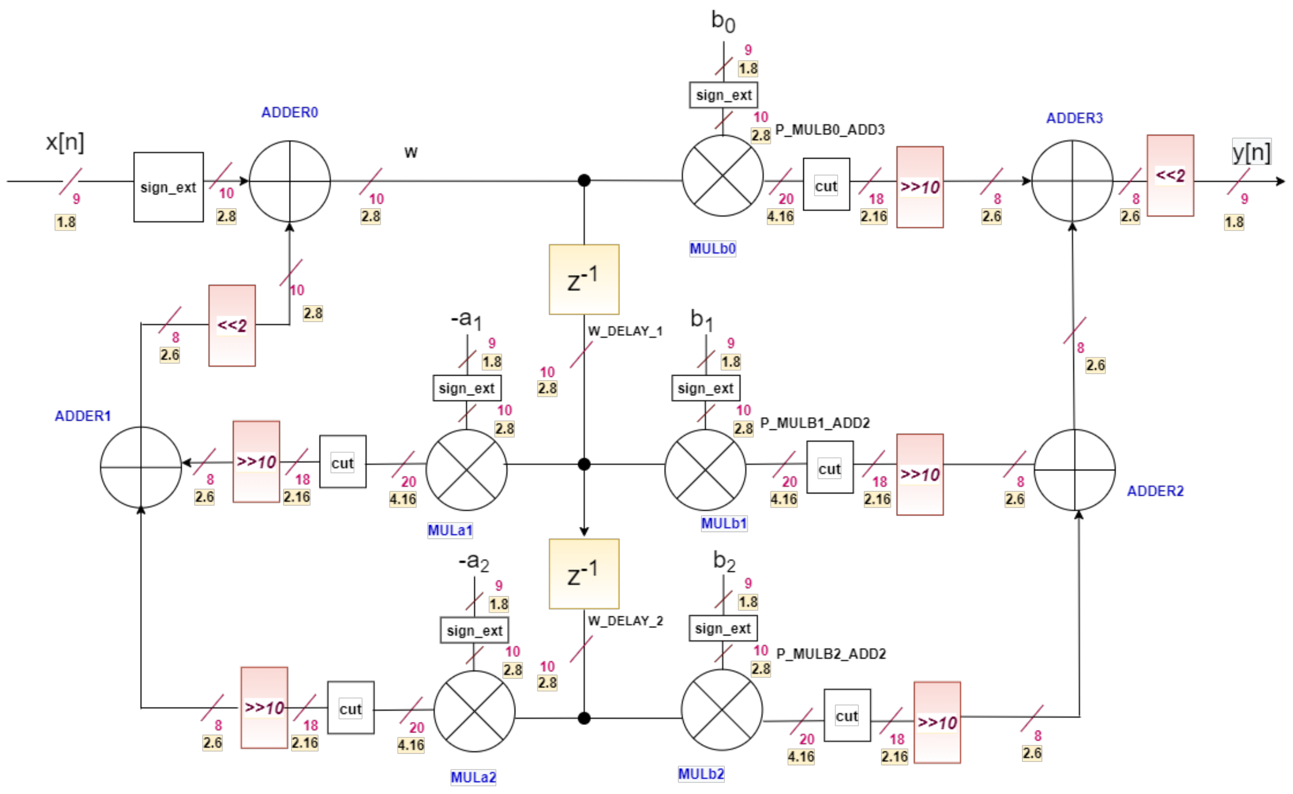


Figure 3: IIR Filter order II internal structure and parallelism

Moreover, simulating different possible conditions, we have verified that the numbers of bit that have been chosen for the different internal signals can be further reduce. This reduction can be performed if we imagine the filter to work only with the input samples that were provided. However, we decide to implement a general version of the IIR filter, considering an input range of [-1, 1).

Doing so, we imagine the worst possible scenario: since the input samples are given by the arithmetic mean between two sinusoidal signal, the worst scenario is referring to the case in which, for multiple clock cycles, both the sines are at their maximum amplitude. This is like considering a step signal constant and equal to 1 (the closest number to 1). The parallelism has been chosen in order to properly cover this case.

Refering to Figure 3, the parallelism of each wire is coloured in magenta, while the fixed point format is highlighted in yellow, to make the explanation clearer.

3.3 THD results

Using Matlab the THD was evaluated, so the results are shown in Figure 4.

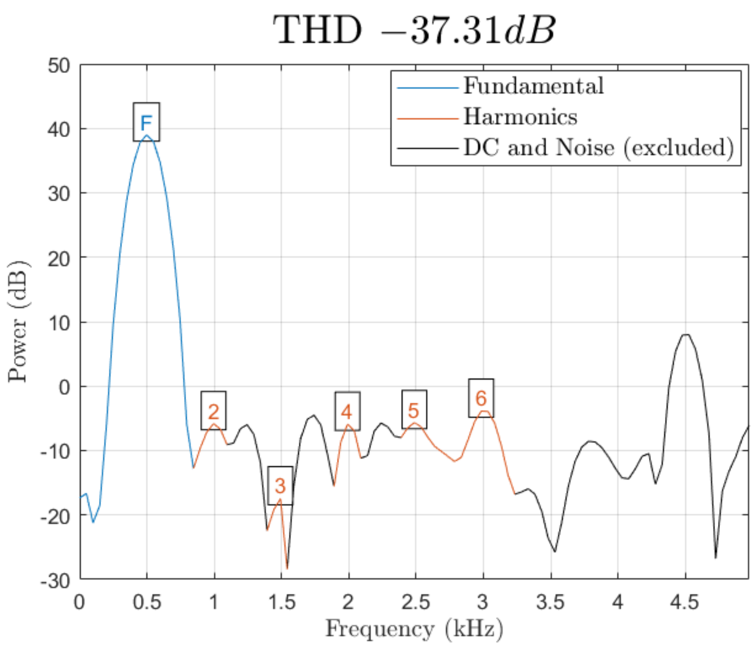


Figure 4: THD result from MATLAB

As required, we obtained a THD value which is not higher to -30 dB, therefore the constraint was satisfied.

4 VLSI Implementation

The filter interface is shown in Figure 5: the input samples (DIN) enters every clock cycle whenever a validation signal VIN is high. The output DOUT contains the result of the filter and VOUT is a validation signal. When a new data is computed by the filter it is loaded in the output register and VOUT is asserted to 1.
According to the constraints, all input/output data and signals are loaded/stored in registers, for this purpose one input register, named INPUT_REG, is implemented. INPUT_REG register samples at every clock cycle on the rising edge of the clock a new data at the input port, in this way whenever VIN is asserted and DIN is in the input port, the architecture is able to store the new sample and it decides either to start to process or not.

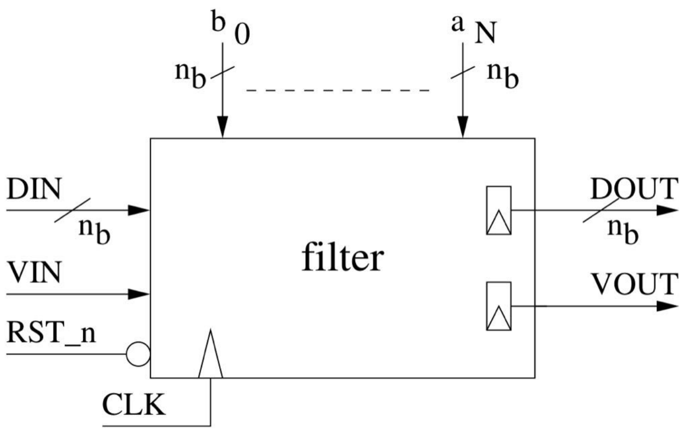


Figure 5: Top-Level Entity Interface

The **IIR** receives as input:

- DIN: the sample involved in expression in that specific clock cycle. It has a 9-bit parallelism;
- A1 and A2: coefficient involved in the feedback part of the filter;
- B0, B1 and B2: coefficients of moving-average part of the filter;
- VIN: signal that enables the DATA_PATH blocks and the input register;
- RST_N: asynchronous active-low signal that allows to reset multiplier's internal registers;
- CLK: timing signal for the correct functioning of the component.

As concerns the outputs, they are represented by:

- DOUT: 9-bit product;
- VOUT: flag activated (high) when there is a valid result on the output. This has been implemented by simply let the VIN signal through two registers.

The sketch of the VHDL structure we have implemented is the one shown in Figure 6.

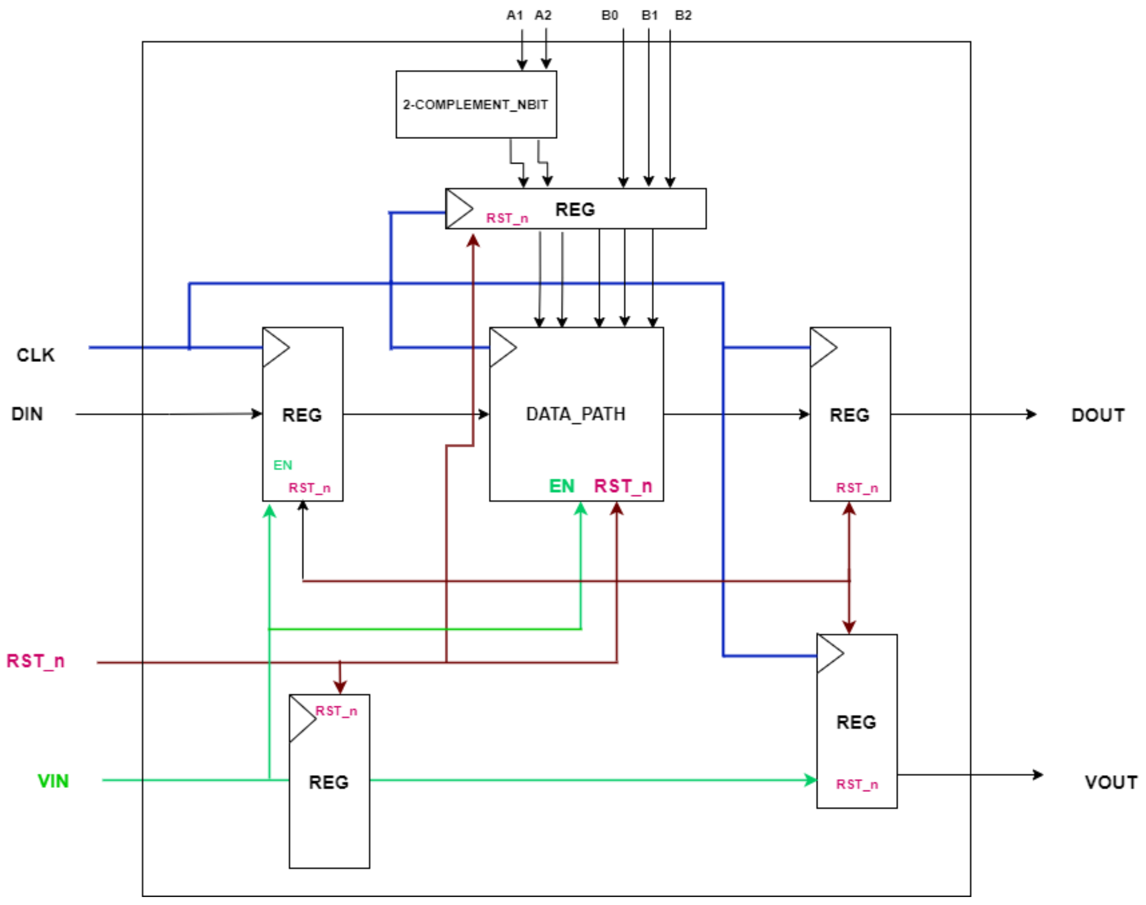


Figure 6: Top-Level VHDL sketch

The two main blocks that are used are `2-COMPLEMENT_NBIT` and `DATAPATH`.
The first one has been added in order to correctly perform the subtraction needed for the IIR feedback chain. The `2-COMPLEMENT_NBIT` is making the 2-complements of the two coefficients, `A1` and `A2`.
This component can be easily avoid by simply inserting the two coefficient already with the minus sign. However, we have decided to use a `2-COMPLEMENT_NBIT` convert so that the coefficients obtained by **MATLAB** can be directly provided as input of IIR filter, without any kind of transformation.

In terms of timing, this component is not introducing any further delay. In fact, despite the presence of one level of registers for each of the five input coefficients, the `2-COMPLEMENT_NBIT` is completely combinatorial. The evaluation of the 2-complement of `A1` and `A2` which can be critical is the one of the first cycle: however, while the conversion is made and the correct coefficient are put in input to the `DATAPATH`, the first input samples is processed by the input register. So, there are no timing problem. In Figure 6, the five registers just mentioned have been represented as one single register for the sake of simplicity. B-coefficients of the filters have no need to be modified.
Moreover, the register is used in order to avoid to increase further the critical path, so the latter depends only on the longest combinational path inside the `DATA_PATH` block.
Finally, they are sent to the `DATA_PATH` block, which performs the expression of the IIR filter.

The timing analysis in shown in Figure 7.

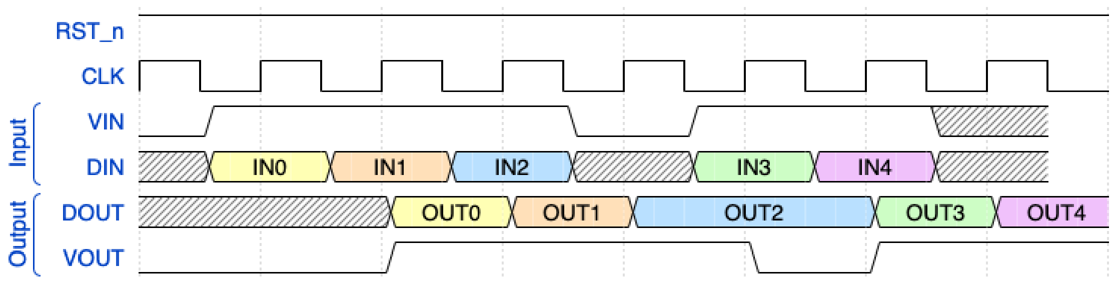
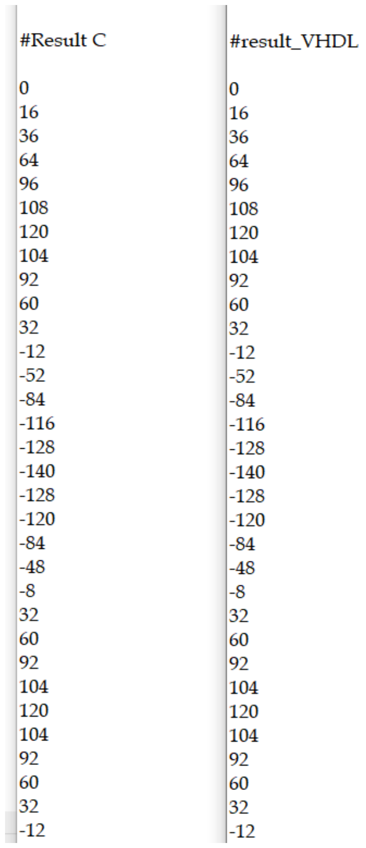


Figure 7: Timing Analysis

As shown in Figure, the system works as expected.
In fact, if `VIN` is low the `DATA_PATH` and the input register stop working, and the output remains unchanged until `VIN` turns high again. Moreover, when `VIN` is low, the testbench has been designed in order to stop reading the input file. Doing so, as shown in the Figure, we do not lost any input values, "*freezing*" it until the `VIN` signal turns high again.

4.1 Simulation

To verify the correctness of the VHDL and the timing in Figure 7 a simulation is performed on ModelSim Altera. The output results of VHDL simulation are compared with the output results of reference C-model.
The two results are identical, as shown in Figure 8, which shows some parts of the two output files results.



#Result C	#result_VHDL
0	0
16	16
36	36
64	64
96	96
108	108
120	120
104	104
92	92
60	60
32	32
-12	-12
-52	-52
-84	-84
-116	-116
-128	-128
-140	-140
-128	-128
-120	-120
-84	-84
-48	-48
-8	-8
32	32
60	60
92	92
104	104
120	120
104	104
92	92
60	60
32	32
-12	-12

Figure 8: Comparison between C-results and Modelsim Altera results

5 Logic Synthesis

In this section, we are required to synthesize the filter with Synopsys Design Compiler. Firstly, it performed the synthesis using Synopsys Design Compiler. Secondly, we performed the same analysis but including the result of the back-annotation process.

5.1 Logic Synthesis without Back-Annotation

First of all, we need to find the maximum clock frequency of our design, forcing a 0 ns clock period.
Applying a **null-clock** to a purely combinatorial architecture corresponds to creating a virtual clock signal, which is associated to a no-real clock pin. Therefore, the optimization potentialities of the synthetizer are quite constrained. This is evident from the obtained results in terms of delay and power.

In order to get the **maximum clock frequency** f_M the Design Compiler must be forced to use a clock with a period equal to zero, in this way the total time required by the architecture to perform all operations in the critical path will be shown. In particular:

$$t_{CLK \to Q} + t_{comb} + t_{setup} = -t_{slack}$$

Since the slack is defined as positive, its value will be certainly negative, in fact the Design Compiler reports a slack equal to -2.52 ns. This is the **critical path**.
The maximum allowed frequency is:

$$f_M = \frac{1}{t_{cp}} = 396.83\text{MHz}$$

To be sure to respect the time constraints related to the maximum frequency, is required to divide by 4 the frequency obtained.
Finally, the value of clock frequency obtained is 99.2 MHz (10.08 ns of period).

In terms of **area**, with a synthesis with 0 ns clock and by putting then the actual period (10.08 ns) **without re-synthesizing**, the circuit we obtain has a cell area equal to 3041.44 μm^2, which is divided into:

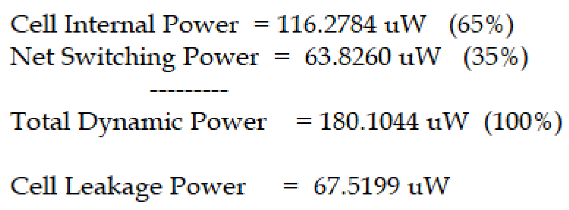
- Combinational area: 2588.18 μm^2
- Buf/Inv area: 243.92 μm^2
- Non-combinational area: 453.26 μm^2

Since the implemented filter is of order II, we obtain a combinational logic which occupies more area than the non-combinational one. In fact, in our architecture there are five multipliers and four adders, and even if they are described in a behavioral way in VHDL, they are still bigger (especially the multipliers) than the registers.
However, the report discussed is obtained for a clock with a period of 10.08 ns, but without making the re-synthesis of the circuit.
If instead if the period is set to 10.08 ns, **re-synthesizing** the circuit, we obtain a cell area equal to 2576.48 μm^2, which is divided into:

- Combinational area: 2124.27 μm^2
- Buf/Inv area: 86.71 μm^2
- Non-combinational area: 452.2 μm^2

We obtain in this way a **reduction in the area of almost the 15%**. This new result is due to the fact that the optimization performed by the Design Compiler changes depending on the clock frequency that is required. In fact a synthesis using more restrictive constraints for clock frequency leads to a netlist with faster gates, that could mean gates with larger transistors. On the hand, when a lower clock frequency is imposed the constraints are more relaxed, so more low

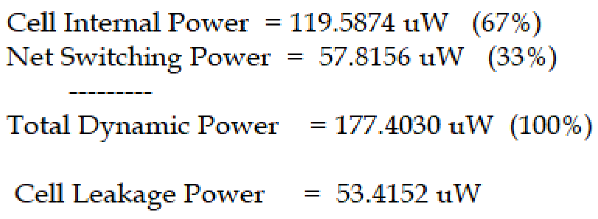
power gates can be used, compared to the previous case, that might have a smaller area.
Moreover, we need to underline the fact that the value of the interconnect area is still *undefined*, because at this step of the design flow interconnections are not evaluated yet by the tools, therefore the final total area is reported as undefined too.
Regarding the **power** needed by the architecture, we reported the two different result obtained when performing the synthesis with 0 ns clock, and setting then the clock to 10.08 ns clock [case (A), Figure 9a] and the synthesis directly performed inserting the clock equal to 10.08 ns [case (B), Figure 9b].



```
Cell Internal Power  = 116.2784 uW   (65%)
Net Switching Power  =  63.8260 uW   (35%)
                  ---------
Total Dynamic Power    = 180.1044 uW  (100%)

Cell Leakage Power     =  67.5199 uW
```

(a) Synthesis with 0 ns clock + set 10.08 ns clock



```
Cell Internal Power  = 119.5874 uW   (67%)
Net Switching Power  =  57.8156 uW   (33%)
                  ---------
Total Dynamic Power    = 177.4030 uW  (100%)

Cell Leakage Power     =  53.4152 uW
```

(b) Synthesis with 10.08 ns clock

Figure 9: Power consumption comparison pre-back-annotation

As it can been observed by looking at Figure 9, in the case of the synthesis with 10.08 ns clock there is a **reduction of the total dynamic power of the 2%**, because constraints related to speed are more relaxed, so slower components gate that allows to save more power are chosen by the library during the synthesis. However, the difference is quite small.

5.2 Logic Synthesis with Back-Annotation

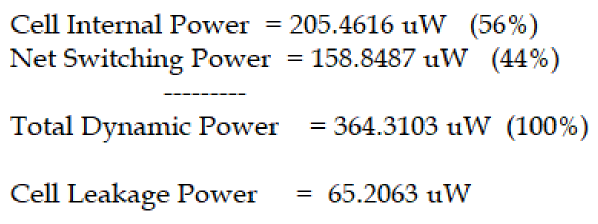
Then, we need to obtain the switching activity.
The procedure starts with the VHDL design. We take our design, and synthesize it with Synopsys, as previously described. We generate an **SDF** file (**Standard Delay Format**), which gives information on the actual delays present in the circuit. The system takes notes of the delays of each instance, based on the point of the circuit in which it is located, on the load, etc. Then it is necessary to simulate the system with the **real activity of the inputs** (with testbench). Thus, the real switching activities are simulated and annotated in the **VCD** (**Value Changed Dump**) file.
With this technique, all transitions of each signal is noted. The important thing is that all the transitions of the inputs are noted, so that they are then correctly propagated within the circuit. The result of the VCD is re-read by Synopsys, and then the final power report is run, which will take into account not only the physical ports, but the transitions of the various signals that are there.
Overall, the entire process is divided into two parts:

- one in the VHDL simulator;
- one in the Synopsys synthesizer.

The main step followed are:

1. after having analyzed, elaborated and compiled the design of the file, the synthesizer generates the **SDF** file, which contains the netlist of the whole circuit. This file contains all circuit delays;
2. once the VHDL simulator has read the SDF file, after running the simulation, it generates a **VCD** file by writing the activities to each node. In this step, the period of the clock in the testbench has been changed and set properly referring to the results obtained;
3. the VCD file is converted to a SAIF file;
4. the synthesizer can read the annotation file (SAIF file) that is returned and update the properties of the nodes with the activities calculated by the VHDL.

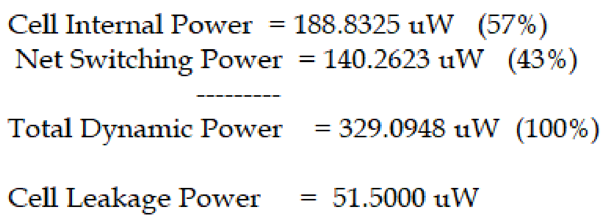
At the end of back annotation process each node is characterized by its real switching activity, so that a more precise power estimation can be done.
The result obtained are shown in Figure 21.



```
Cell Internal Power  = 205.4616 uW   (56%)
Net Switching Power  = 158.8487 uW   (44%)
                    ---------
Total Dynamic Power    = 364.3103 uW  (100%)

Cell Leakage Power     =  65.2063 uW
```

(a) Synthesis with 0 ns clock + set 10.08 ns clock



```
Cell Internal Power  = 188.8325 uW   (57%)
 Net Switching Power  = 140.2623 uW   (43%)
                    ---------
Total Dynamic Power    = 329.0948 uW  (100%)

Cell Leakage Power     =  51.5000 uW
```

(b) Synthesis with 10.08 ns clock

Figure 10: Power consumption comparison after-back-annotation

First of all, making a comparison between the two different synthesis analysed, as it can been observed by looking at Figure 10, in the case of the synthesis with 10.08 ns clock there is a **reduction of the total dynamic power of almost the 10%**, for the same reason explained previously.
However, the most relevant comparison that must be performed is the one related to the power result pre-back-annotation and post-back-annotation, for both the two cases analysed.

In the case of the system with a period of 10.08 ns but the synthesis performed with respect to the 0 ns-clock period, we obtain:

- the total dynamic power is doubled, where:
  - the cell internal power has almost doubled (from 116.27 µW to 205.46 µW);
  - the net switching power is increased from 63.82 µW to 158.85 µW ;
- the cell leakage power is diminish of the 3%.

In the case of the system with a period of 10.08 ns, we obtain:

- the total dynamic power has increased of the 185%, where:
  - the cell internal power has almost doubled (from 119.59 µW to 188.83 µW);
  - the net switching power is increased from 57.82 µW to 140.26 µW ;
- the cell leakage power is diminish almost of the 4%.

As expected, in both of the structure there is an overall increase of the power consumption. The back annotation, as stated at the beginning of the Section, is a process in which we are making a more realistic estimation of the performance of our circuit, since we are taking into account the **actual delays present in the circuit** and the **real switching activities**. What diminish in both cases, even if of a small quantity, is the cell **leakage power**.
In our circuit, the leakage power is the 18% of the total dynamic power in the case of the circuit with a period of 10.08 ns but the synthesis performed with respect to the 0 ns-clock period, and the 16% of the total dynamic power in the case of the circuit with a period of 10.08 ns.
Obviosuly, the small this amount is, the better.
However, in our design, we obtain quite high value of leakage power since no power-optimization has been applied. The focus of this laboratory is in fact the performance in terms of area and throughput, while the power-consumption aspect has been slightly overshadowed.

6 Place & Route

All the previous discussions do not take into account the interconnection and routing of the system.
For this reason a further power report is obtained using **Cadence SOC Innovus**.

We have decided to physical layout of the first circuit analyzed in the previous Section, so the one in which the first synthesis is done with a clock of 0 ns, and then it is imposed the actual clock period of 10.08 ns, without re-synthesizing. Since there were not specific-specification reported on the laboratory requirement, we decided to implement the most general solution, which is the first between the two possibility just mentioned.

Innovus considers not only the capacitance of each block of the design as done in the previous steps, but also parasitic resistances and capacitances due to the interconnections are used to evaluate each power consumption contribution.
The most important steps are:

- **Placement**:the cells that compose the operator are placed;
- **CTS optimization**: try to optimize the design in order to satisfy timing constraints;
- **Route**: it generates the connections among the cells;
- **PostRoute optimization**: it tries to optimize the design.

The first step is to import our synthesized design in the form of a verilog netlist generated by Design Vision as well as the SDC file containing the timing constraints.
After that, we have to structure the **floorplan** by setting, among other things, the core aspect ratio, which was set to 1.0 (a square). This first value is fundamental as it affects the placement, the routing, the frequency and a few additional parameters. Moreover, core margins were fixed to 5 µm.
After that, **power rings** are placed all around the core. They will be necessary to distribute VDD and GND to the whole die. These lines run along the top layers (metal9 and metal10) which are dedicated to power supply, as shown in Figure 11.

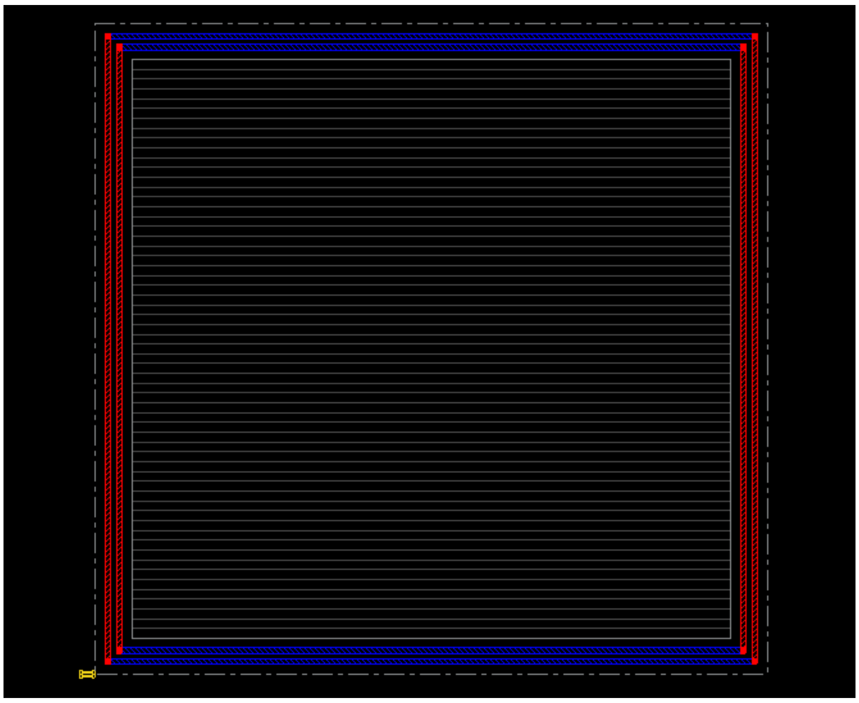


Figure 11: Power supply physical design of the final IIR

Then **placement** can take place. This process takes some time. It is also carried out more than once to improve the outcome.
A good placement is the one that minimizes the area and wiring costs by putting interacting cells close to each other. This in turn reduces delays and power dissipation. The placement operation, in principle, employs an iterative algorithm that divides the available area into smaller regions whose boundaries represent constraints which dictate where cells can be placed. This is done at each loop. Once the regions are small enough the algorithm assigns a row for each cell and the placement is complete. For the standard cell placement, metal layers from 1 to 8 are used. Clock nets are also positioned.
The first optimization happens at this point. It is a **post-CTS** optimization. This kind of improvement is performed only on the datapath and it includes checking for DRVs and setup and hold times, area and power optimization, congestion reduction.
The actual routing takes place now. By using the **NanoRoute algorithm**, parameters such as the number of nets, the area occupied by the wires, the total wire-length, the congestion, the coupling etc. are optimized.
A **second optimization is performed following routing**: setup and hold times are verified.
After this second improvement, what's left to do is place the **filler cells**. These cells are needed to ensure wells' continuity which is beneficial to the overall design's fabrication. First of all, transistors placed at the edge of the n-well (the type commonly used) suffer from an unwanted effect known as WPE (well proximity effect) which worsens performances of these transistors. The second problem is that discontinuity calls for the need of tap connections which take up more space in accordance with the design rules. Moreover, mask generation is also easier when continuity is guaranteed.
After this step, the place & route of the design is complete.
In the following part, timing and integrity of our circuit have to be analyzed. To do so, the **parasitics** (i.e. resistances and capacitances) **for each metal wire are extracted**. From timing reports, it has been observed that the slack for the setup time is quite large as the clock frequency used is by far lower than the maximum value. The hold time, however, as previously said is very close to the limit for the majority of paths. Despite this, **no negative slack times have been recorded**.
The final step consists in looking for **geometry and connectivity violations** and none have been spotted.
Now the circuit has to be simulated again to get an even more realistic power report.

Result of Innovus synthesis in shown in Figure 12.

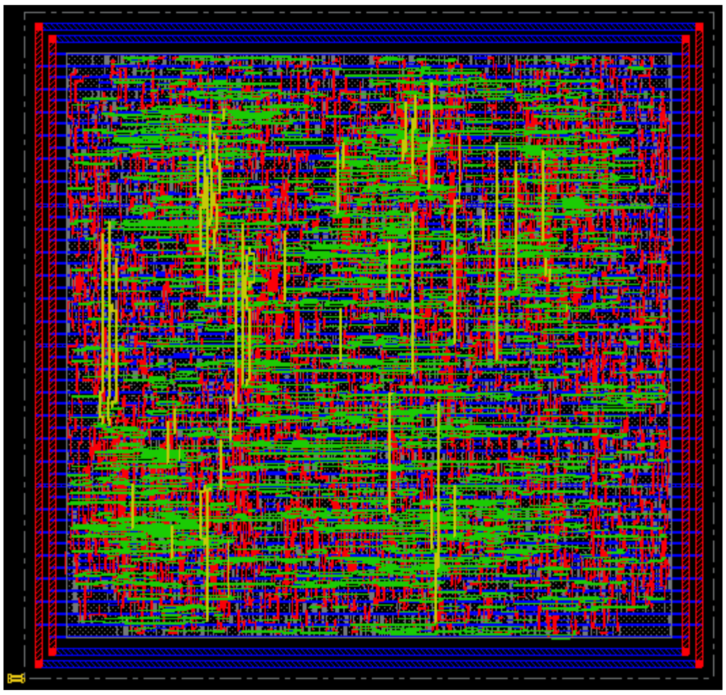


Figure 12: Layout of the IIR filter

Innovus consider not only the capacitance of each block of the design as done in the previous step, but also parasitics resistance and capacitance due to the interconnections are used to evaluate each power consumption contribution.
After performing placing and routing operations, we evaluate for the last time the power consumption of our filter considering new backannotation data provided by a Modelsim simulation of the new netlist generated by Innovus.

Power	Value [mW]	Percentage
Total Internal Power	0.6039	51.4%
Total Switching Power	0.5106	43.4%
Total Leakage Power	0.0613	5.2%
Total Power	1.1758	100%

Table 1: Total Power Consumption after Place & Route

7 Advanced Architecture Development

7.1 Reference Model Development

Look-ahead (**LA**) techniques are highly effective in attaining high speed VLSI implementation of recursive digital filters. This non-universal technique can be applied in presence of loops, when universal methods are not effective. As a consequence a more complex architecture is achieved, compared to the original DFG, but now it is possible to employ one of the universal methods in order to increase the throughput.
In the original DFG of IIR (Dircet Form II), let's focus on the feedback part: the critical path $T_{cp} = 2T_{ADD} + T_{MUL}$ and it is equal to T_{∞}, so universal techniques are not effective to increase the throughput. So a **1-look-ahead** was employed.

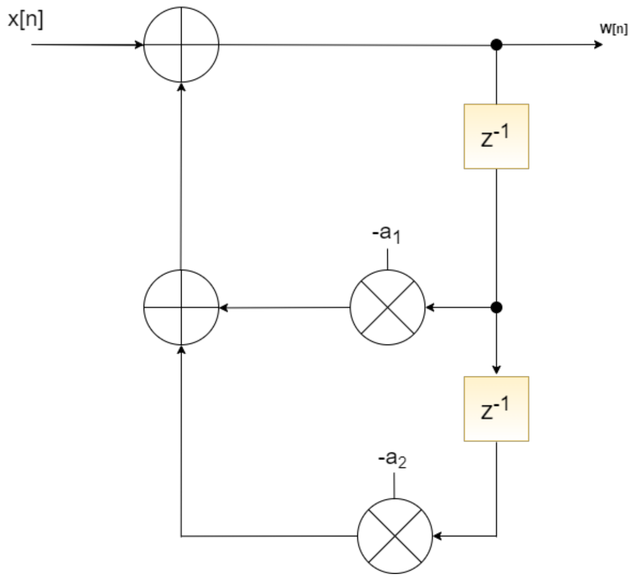


Figure 13: Feedback part of IIR Direct Form II

Looking at the feedback part of Direct Form II of IIR, the equation that describe the behavior of this loop, as shown in Figure 13 is

$$w[n] = x[n] - a_1 w[n-1] - a_2 w[n-2] \tag{1}$$

Starting from the equation 1 it is possible to obtain the expression of $w[n]$ at the previous sampling instant ($w[n-1]$):

$$w[n-1] = x[n-1] - a_1 w[n-2] - a_2 w[n-3] \tag{2}$$

Now, substituting the equation 2 in the equation 1, we can obtain a new expression for $w[n]$:

$$w[n] = x[n] - a_1 x[n-1] + (a_1^2 - a_2) w[n-2] + a_1 a_2 w[n-3] \tag{3}$$

Therefore, the feedback part in Direct Form II of IIR can be replaced with a new architecture, as suggested in equation 3. Obviously, coefficients are different from the initial ones, since they are obtained by combining them (i.e., $a_1^2 - a_2, a_1 a_2$). However 9-bits are still enough to represent them without affecting their precision.

The new schematic which represent the expression of the equation 3, is shown in Figure 14.

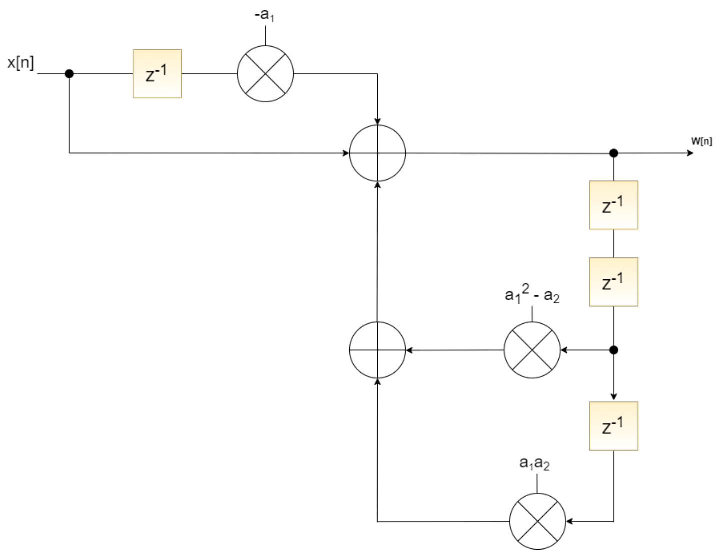


Figure 14: Feedback after 1-look-ahead

Computing the new value of T_∞ we can observe that it is lower than critical path. Differently from the initial configuration, here **universal technique can be employed**. The new DFG of IIR filter after **1-look-ahead** becomes the one shown in Figure 15.

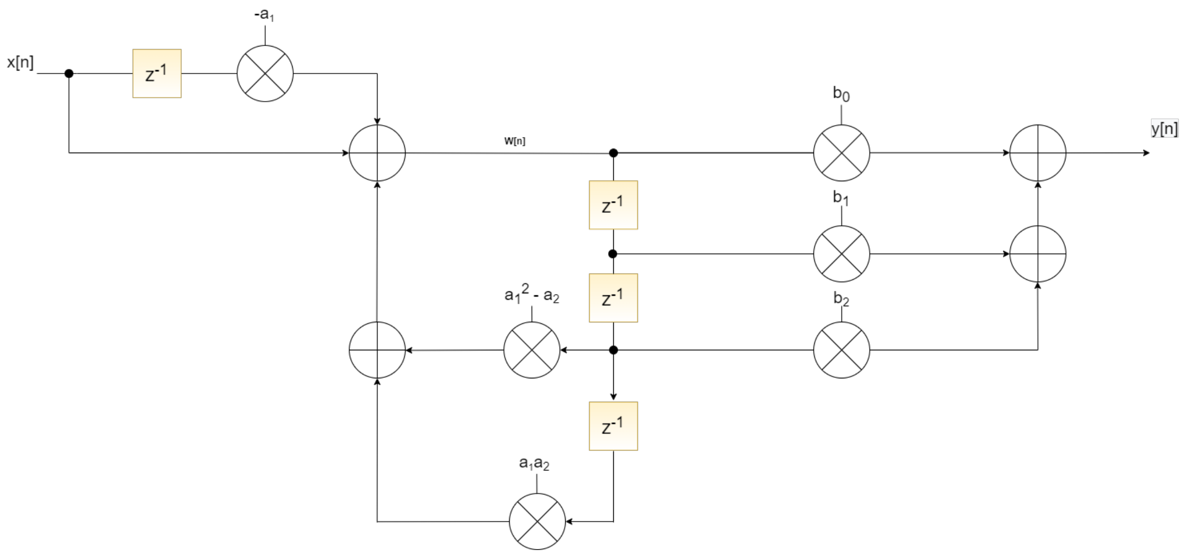


Figure 15: IIR architecture after 1-look-ahead

The overall critical path now is:

$$T_{cp} = 3T_{ADD} + 2T_{MUL}$$

So, a possible approach can be applying **retiming** in order to move some registers along the critical path to reduce it.

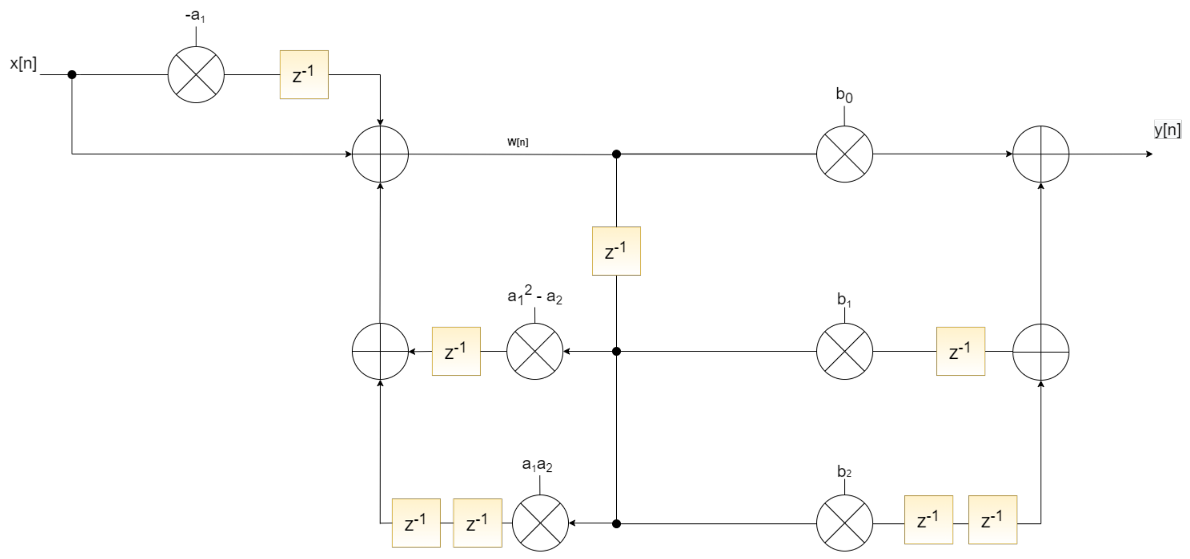


Figure 16: IIR architecture after 1-look-ahead and retiming

Moreover, some **pipeline registers** can be inserted along feed-forward cut-sets, again with the aim of minimizing the critical path.
Finally, all these optimizations led to the final architecture, in Figure 17, that minimize throughput of IIR filter.

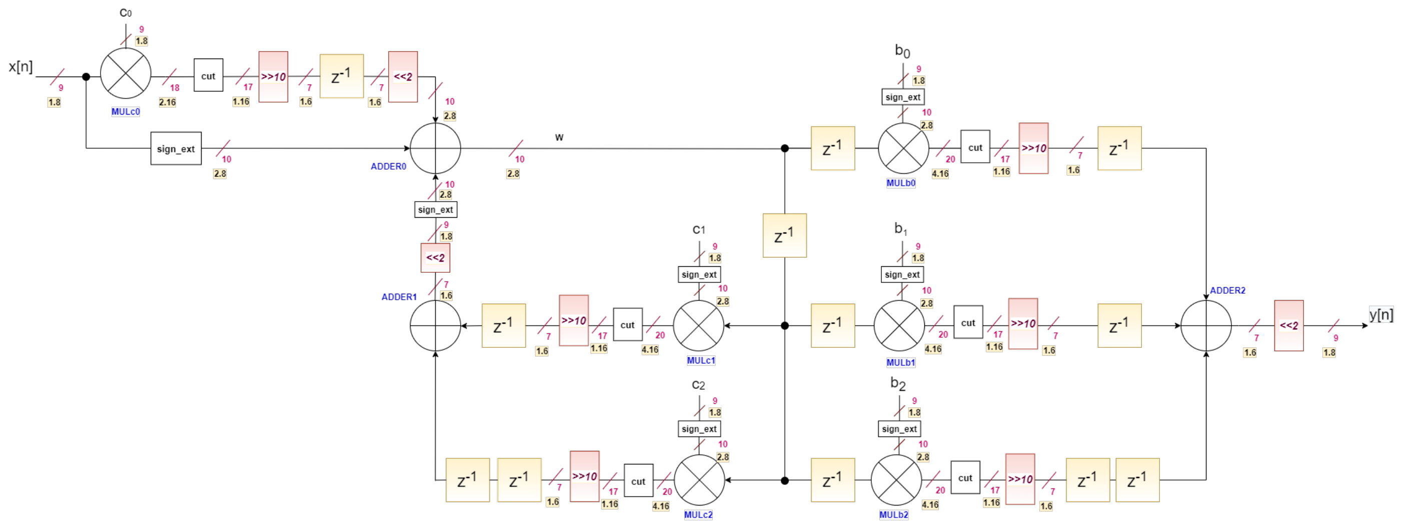


Figure 17: IIR architecture after 1-look-ahead, retiming and pipelining

The DFG reported in Figure 17 is the best in terms of **throughput**.
In fact, the critical path:

$$T_{cp} = T_{MUL}$$

Since one sample can be filtered at each clock cycle the **throughput** is:

$$Th = \frac{1}{T_{cp}} = \frac{1}{T_{MUL}}$$

As we can see in Figure 17, the coefficients b_0, b_1 and b_2 are the same of non-optimized version.

However, there are also some new coefficients, defined in the following way:

$$c_0 = -a_1$$

$$c_1 = a_1^2 - a_2$$

$$c_2 = a_1 a_2$$

All the comments that have been done in Section 3.2 for what concern the parallelism are still valid. A C-model has been created in order to simulate the VHDL behaviour of the filter and to control which is the lower parallelism that can be used in order to not have overflow. Moreover, as in Figure 3, in Figure 17 the parallelism of each wire is coloured in magenta, while the fixed point format is highlighted in yellow, to make the explanation clearer.
Finally, as required, we obtain a THD value which is not higher to -30 dB, equal to -35.56 dB.

7.2 VLSI Implementation

The filter interface is shown in Figure 5, and it is the same of the not-optimized version. The sketch of the VHDL structure we have implemented is the one shown in Figure 18.

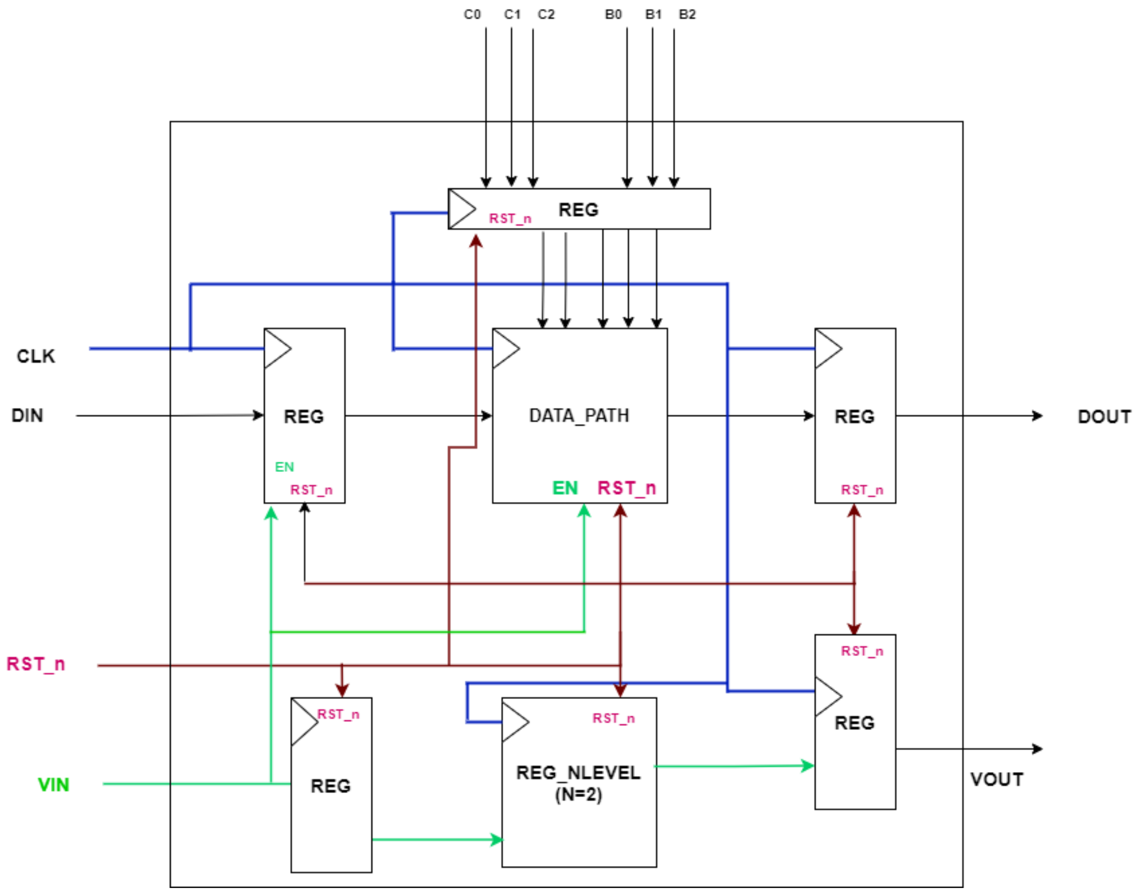


Figure 18: Top-Level VHDL sketch look-ahead

The two main block that is used is the `DATAPATH`.
Here, the `2-COMPLEMENT_NBIT` component is not used anymore, but the new coefficient `c_0`, `c_1` and `c_2` are directly pass to their register in the right format. Again, as in Figure 6, the five registers for the five input coefficients have been represented as one single register for the sake of simplicity, and again B-coefficients of the filters have no need to be modified.
Moreover, in order to save power, only some of the internal registers of the `DATAPATH` has the enable signal. In particular,

the seven registers that are in the forward branches, related to the different calculus with B-coefficients, have not the enable signals, since *freezing* all the signals coming from the input and in feedback will be enough to *freeze* the filter execution until `VIN` comes back to 1 again.
Respect to the previous case, as it can be seen from Figure 17, a 3-inputs adder is necessarily needed. This 3-inputs adder is so exploited also at the output node. This has been done only because it was directly required to not choose the architecture of adders and multipliers, but to use "+" and "-". So, since the implementation of their architecture is not under our control, we are not sure of which option will has between the best performance, between an adder with three inputs or two adder with two inputs, and we decide, for simplicity, to re-use the same adder of the input, imagine we would have a possible reduction of the critical path.
The timing analysis in shown in Figure 19.

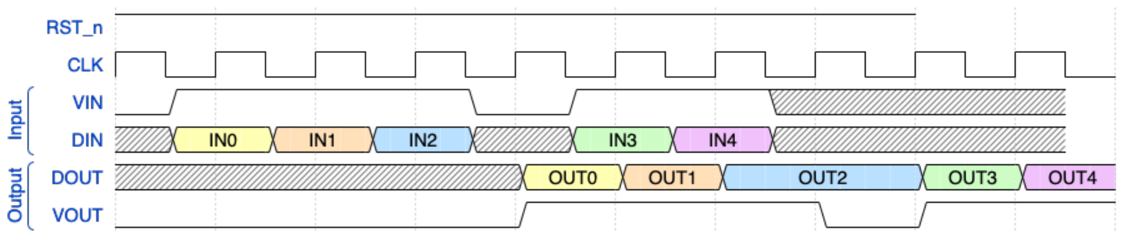
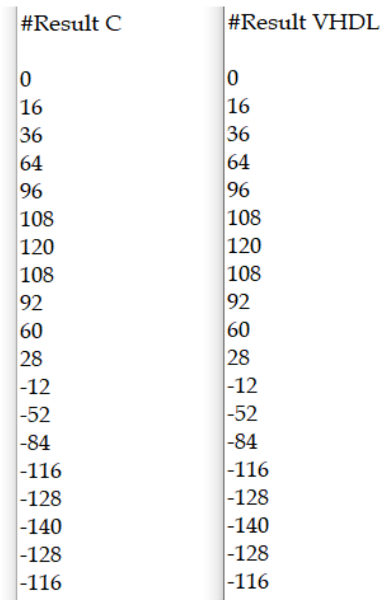


Figure 19: Timing Analysis

As shown in Figure, the system works as expected.
In fact, if `VIN` is low the `DATA_PATH` and the input register stop working, and the output remains unchanged until `VIN` turns high again. Here, differently from Figure 7, we have a latency of three clock cycles, due to the internal structure `DATAPATH`.

7.3 Simulation

To verify the correctness of the VHDL and the timing in Figure 19 a simulation is performed on ModelSim Altera. The output results of VHDL simulation are compared with the output results of reference C-model. The two results are identical, as shown in Figure 20, which shows some parts of the two output files results.



#Result C	#Result VHDL
0	0
16	16
36	36
64	64
96	96
108	108
120	120
108	108
92	92
60	60
28	28
-12	-12
-52	-52
-84	-84
-116	-116
-128	-128
-140	-140
-128	-128
-116	-116

Figure 20: Comparison between C-results and Modelsim Altera results

7.4 Logic Synthesis

First of all, we need to find the maximum clock frequency of our design, forcing a 0 ns clock period. Design Compiler reports a slack equal to -1.15 ns. This is the **critical path**.
The maximum allowed frequency is:

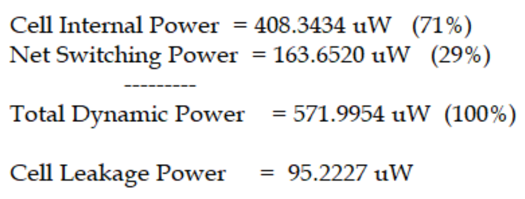
$$f_M = \frac{1}{t_{cp}} = 869.56\text{MHz}$$

To be sure to respect the time constraints related to the maximum frequency, is required to divide by 4 the frequency obtained.
Finally, the value of clock frequency obtained is 217.39 MHz (4.6 ns of period). In terms of **area**, with a synthesis with 0 ns clock and by putting then the actual period (4.6 ns) **without re-synthesizing**, the circuit we obtain a cell area equal to 4228.87 µm^2, which is divided into:

- Combinational area: 3305.31 µm^2
- Buf/Inv area: 362.56 µm^2
- Non-combinational area: 923.55 µm^2

In this analysis, for the same reason explained in the previous sections, we will carry on only the **not re-synthesised** analysis of the circuit. Comparing the new result with the ones of the first version of the filter, we obtained an increase in terms of area of almost the 28%. Again, since the implemented filter is of order II, we obtain a combinational logic which occupies more area than the non-combinational one. However, having applied the look-ahead technique, the pipeling technique and the re-timing, there is an increase of the non-combinational area of almost the 50% and an increase of the buf/inv area of the 33%.

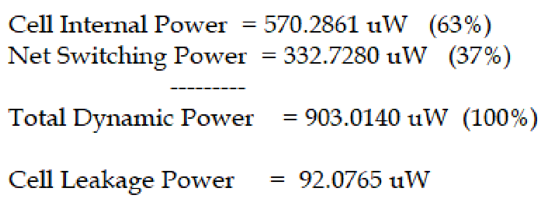
Regarding the **power** needed by the architecture, when performing the synthesis with 0 ns clock, and setting then the clock to 4.6 ns clock we obtained the result reported in Figure 21a.



```
Cell Internal Power  = 408.3434 uW   (71%)
Net Switching Power  = 163.6520 uW   (29%)
                     ---------
Total Dynamic Power    = 571.9954 uW  (100%)

Cell Leakage Power     =  95.2227 uW
```

(a) Power result pre-back annotation



```
Cell Internal Power  = 570.2861 uW   (63%)
Net Switching Power  = 332.7280 uW   (37%)
                     ---------
Total Dynamic Power    = 903.0140 uW  (100%)

Cell Leakage Power     =  92.0765 uW
```

(b) Power result post-back annotation

Figure 21: Power consumption comparison pre- and post-back-annotation

Respect to the not-optimized version, reported in Figure 10a, we have an increase of the 248%! The power consumption after the back-annotation process of the optimized version is more than double respect to the one of the first IIR filter implementation.
This was predictable, since our optimization was done in terms of throughput, and not in terms of power, and consequently we have add registers and operator which are obviously consuming more power.
At the end of back annotation process, each node is characterized by its real switching activity, so a more precise power estimation can be done, obtaining the result reported in Figure 21b.
Respect to the simulation without back-annotation reported in Figure 21a, there is an increase of the 158%!
This means that in the first analysis Design Compiler reports an underestimation of the actual consumption of our circuit, as in the previous cases.

7.5 Place & Route

All the previous discussions do not take into account the interconnection and routing of the system. In order to get the physical layout of the filter we have followed the same steps which are described in Section 6.
Result of Innovus synthesis in shown in Figure 22.

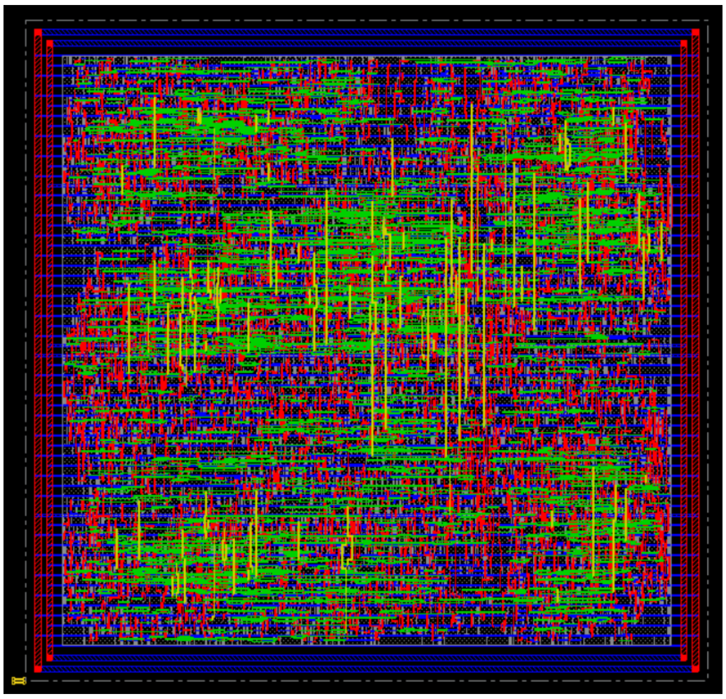


Figure 22: Layout of the IIR filter

After performing placing and routing operations, we evaluate for the last time the power consumption of our filter, we obtain a total power consumption equal to 1.614 [mW]. This is distributed as shown in Table 2.

Power	Value [mW]	Percentage
Total Internal Power	0.936	58.02%
Total Switching Power	0.592	36.66%
Total Leakage Power	0.086	5.31%
Total Power	1.614	100%

Table 2: Total Power Consumption after Place & Route

Respect to the not-optimized version, reported in Figure 12, we have an increase of about 137.3%! As stated before, the new structure of IIR filter has been designed in order to optimize the frequency response, which is instead being improved of the 219%. In fact, we have obtained a possible working frequency of 217.39 MHz instead of 99.2 MHz found with the common IIR filter. On the contrary, area and power performances are not improved, but this was not between our requirements.