Laboratory Report

Academic year 2020/2021

Report Lab 1 - Integrated Systems Architecture

Design and implementation of a digital filter

Alessio Naclerio ID : S270065
Simone Di Blasi ID : S276570
Stefano Floridia ID : S279930

Contents

1 Reference model development

The first goal of this laboratory is to design an IIR digital filter with a cut-off frequency of 2KHz and a sampling frequency of 10KHz.
The work began by describing a Matlab model of the filter and testing it with two sinusoidal waves of different frequency.
The results obtained were then compared with those obtained from a model written in C by evaluating the performance of the fixed point implementation.
Finally, before moving on to the VSLI implementation, it was evaluated whether the area of the architecture was below -30dB of THD.

1.1 Filter design and coeffcient quantization using Matlab

The filter design begins by defining the order of the filter and the number of bits to represent the coefficients. In our case we have created a filter of order 2 with coefficients on 9 bits.
Using these parameters as an input to a function and setting its cut-off and sampling frequency as required by specifications, we will be able to obtain the transfer function of the IIR filter, both in the quantized and direct form.
The image of the transfer function is shown below.

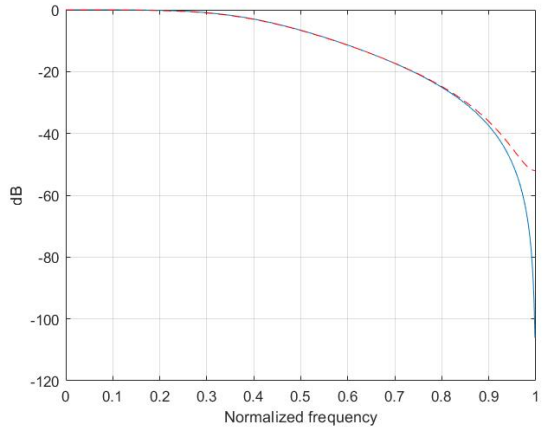

Figure 1: IIR Transfer Function (Red is the quntized funcion).

The function that implements this mechanism allows to save also the coefficients a and b as integers and as quantized parameters (ai, bi, aq, bq).

a_i	256 -95 50
b_i	52 105 52
a_q	1 -0.3711 0.1953
b_q	0.2031 0.4102 0.2031

Tab. Outputs.

1.1.1 Filter Test

To test the functioning of the filter, a signal was created consisting of two sinusoidal waves at two different frequencies, one in band with a frequency of 500 Hz and one out of band with a frequency of 4.5 KHz.
Using the obtained value (x) and the quantized parameters (aq, bq), generated by the IIR filter design function, as input to the Matlab "filter" function which allows to filter the input data x using a defined rational transfer function from the numerator and denominator coefficients bq and aq.
Below is an image depicting the two sinusoidal (x1, x2), the function input ($x = \frac{x_1+x_2}{2}$) and the result of the filtering operation (y).

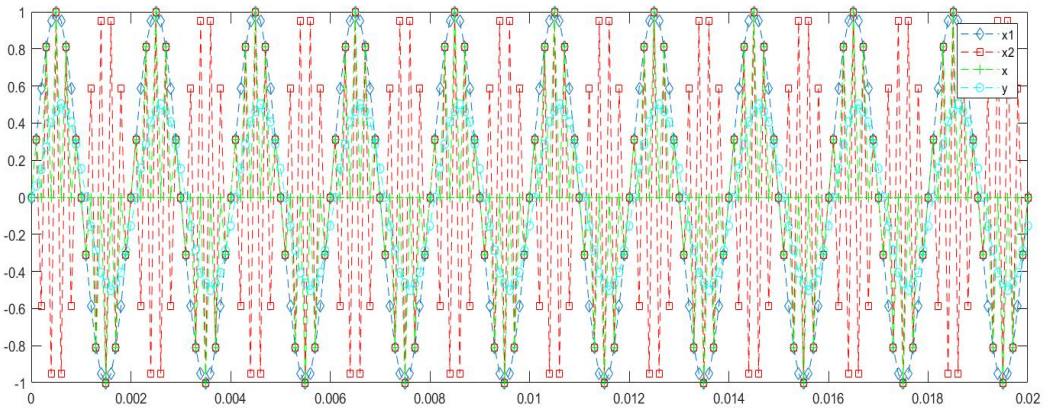

Figure 2: Inputs and test result.

In addition, the input and output samples are saved as quantized data represented on 9 bits as integer values, in two text files.
If we use the quantized output data to evaluate the total harmonic distortion we get a THD value equal to -52.53dB.
The graph is shown below.

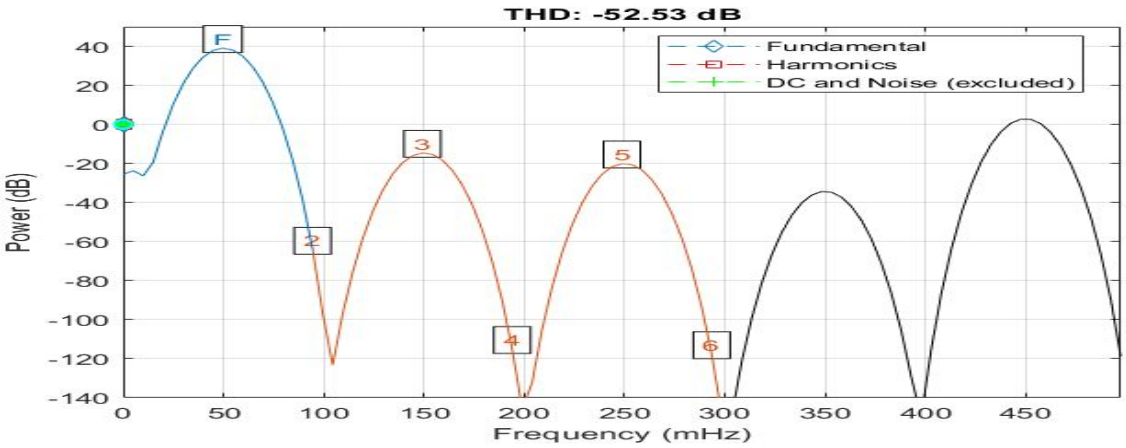

Figure 3: THD of the Matlab model

1.2 Fixed point C model

In this section, a fixed-point model of the filtering operation is implemented through a program written in C.
This is done using the following formula:

$$\sum_j x_{i-j} * b_j - \sum_k y_{i-k} * a_k$$

Direct form II (canonical direct form) for IIR filters is used to implement this formula.
The program is able to cyclically read the input file generated by the Matlab model and calculate the filtered results. In order to do this, it uses a shift register and evaluates feed-back and feed-forward and the various intermediate values.
At the same time as reading, the program prints the results obtained in another file.
The code in order to emulate the behavior of the filter needs initial parameters to be fixed, such as:

- Filter order
- Number of bits
- Coefficient b_0 (first value of b_i)
- b array (with the reaming 2 values of b_i)
- a array (with the last 2 values of b_i)

The program allows to reduce the number of bits after each multiplication, to reduce the bitwidth (and therefore the area) of the filter.

1.2.1 THD

Taking the output file of the fixed-point model made in C and on this data we can evaluate the THD.
The total harmonic distortion (THD) function allows to have the sinusoidal signal in dBc at real values passed to it.
The graph of the function is shown below.

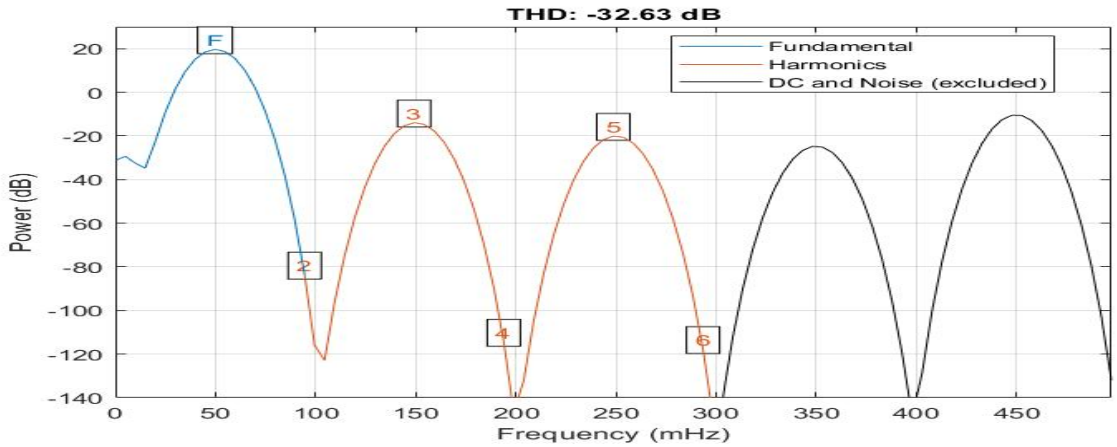

Figure 4: THD of the fixed-point model

As you can see from the image, the THD value for this model is -32.63dB the is lower then -30dB that is the constraint.

1.3 Comparison

It can be seen that in the two models used we have different results. This can be seen by directly analyzing the values present in the output files but also by looking at their THDs. This is due to the fact that for the C model we are using fixed-point operators while the Matlab model uses floating point ones.

2 VLSI implementation

2.1 Starting architecture development

According to the specifications given, now is time to develop a VHDL model following the architecture's requirements described in the previous chapter. A testbench file "$tb\_fir.v$" has been written, as requested, and saved inside the "tb" folder.
The goal for this part is to simulate and implement the VHDL model described.

2.2 Simulation

In this sections the aim is to simulate the filter, with the help of ModelSim, in order to compare the results produced now with what we obtained with the C model, and be sure that both are identical. After having performed the simulation following the steps showed in "*Documents.pdf*", the results has been saved in the file "*results.txt*" and, if we compare it with the content of the file "*output.txt*", that is the file containing the output of the C model, it is possible to notice that the two files are identical.
As requested by the laboratory assignment, it is possible to show the waveform of the simulation, highlighting that the filter is working correctly even when VIN is 0.

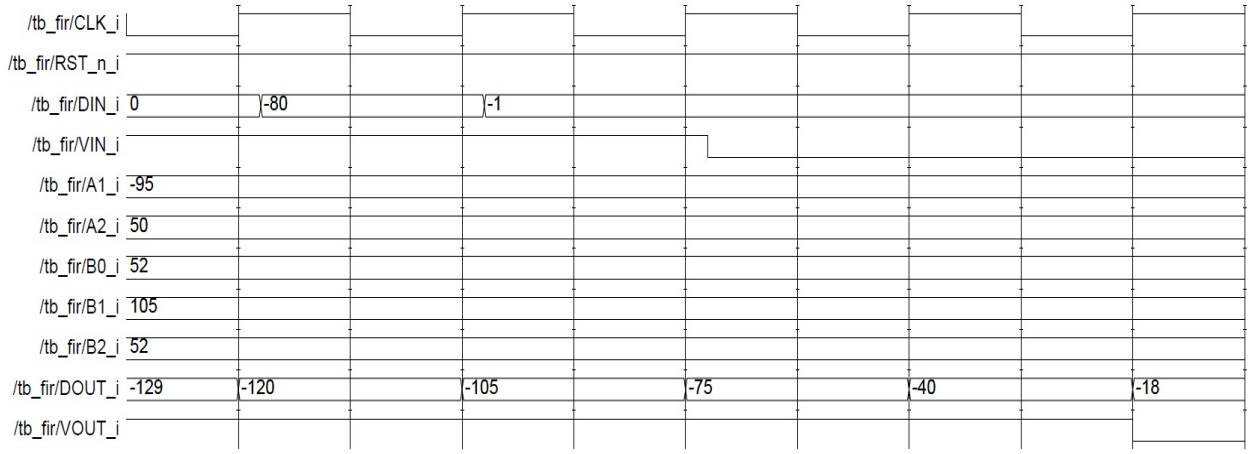

Figure 5: IIR Filter waveform

The image shown is a portion of the complete waveform saved as "$sim\backslash IIR.pdf$"

2.3 Implementation

Terminated the design simulation, next step is to perform the logic synthesis with Synospys, setting as work environment the folder syn. Then the last part is devoted to place and route the filter using Cadence SOC Innovus.

2.3.1 Logic Synthesis

The purpose of the logic synthesis is to find the maximum clock frequency (f_M) at which the operation gives as a result a slack equal to zero.

In order to do so the script "*syn.tcl*" has been run several times, each time adding the value of the slack obtained to the previous value of the clock period, used to run the synthesis. This operation is done until the slack equal zero is obtain. The clock period obtained is 3.37ns so $f_M = 296.7$MHz.

With this frequency,an area report is generated running the script "*post_syn.tcl*". The generated file is called "*area_reportfM.txt*" and the main observable points are:

- Combinational area: 2046.86999 µm^2
- Buf/Inv area: 127.946000 µm^2
- Noncombinational area: 202.160007 µm^2
- Total cell area: 2249.030004 µm^2

As Requested by the Laboratory assignment, the script "*syn.tcl*" has been run one more time, with a clock period four times grater. A new area report is generated with the name "*area_reportfM_4.txt*", and we can observe some differences with the previous results:

- Combinational area: 1815.716006 µm^2
- Buf/Inv area: 92.568000 µm^2
- Noncombinational area: 202.160007 µm^2
- Total cell area: 2017.876012 µm^2

The outcome of the frequency constraint in the second synthesis is that the total area is lower, and in particular what has been reduced is the combinational area.

Last point is to perform a switching activity power estimation at a clock frequency $f_{clk} = f_M/4$. According to the informations given in the file "*Documents.pdf*", a ModelSim simulation is launched using scripts saved in the folder "*sim*". First script to be run is "*compile_postsyn.sh*" for compiling the "*tb*" folder, the post-synthesis netlist and initialize the simulation. Then the script "*run_postsyn.do*" that opens a file "*.vcd*" and then the simulation is launched for 3µs. The switching activity power estimation has been performed and the "*.vcd*" has been converted into a "*.saif*" file and saved inside the "*saif*" directory. The following steps are performed in the Synopsys Design Compiler where the script "*power.tcl*" is run, the file "*.saif*" is read together with the post-synthesis netlist. The result is a power report named as "*power_reportfM_4.txt*". The main elements to be observed on this report are:

- Cell Internal Power = 172.2397 µW
- Net Switching Power = 150.9063 µW
- Total Dynamic Power = 323.1460 µW
- Cell Leakage Power = 40.3169 µW

2.3.2 Place Route

Before starting the Innovus environment, a new directory called "*Innovus*" has been created where all the results of the place and route are saved. The aim is to evaluate the design area, simulate the netlist and estimate the power consumption, using "*Innovus*" and beign sure that the rsults obtained are the same of the previous sections.

For this part all the steps described in the "*Documents.pdf*" file have been followed.

3 Look Ahead IIR Filter VLSI implementation

The aim of this part of the laboratory has been to develop, simulate, synthesize, place and route a second order, 9-bit IIR filter modified with the look-ahead technique. Power estimation based on post-synthesis and post place and route switching-activity has been carried out. The project has been organized in folders as suggested.

3.1 Architecture Development

3.1.1 Design Description

Starting from the basic IIR filter, the J-look-ahead technique (with J = 1) has been applied. The main benefit of this modification on the circuit is that other universal techniques can be implemented in the resulting netlist, thus enhancing performance. In this case, pipeline registers have been introduced. This causes the creation of pipeline stages with the great advantage of controlling the delay of a path between two registers. Thanks to this modification, it is possible to reach a condition in which the **critical path delay** T_{cp} is equal to the **loop iteration** T_{∞}.

The starting point for the application of the look-ahead technique is the direct form II initial equation for the IIR filter, which is reported here:

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

where:

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

Then, the formula has to be written for $y[n+1]$:

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

At this point, $w[n]$ has to be substituted in the equation above:

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

Going back to $y[n]$:

$$y[n] = b_0 w[n] + b_1(x[n-1] + a_1 w[n-2] + a_2 w[n-3]) + b_2 w[n-2]$$

The last equation could be expressed in another form, where b_1 explicitly multiplies the operands inside the parenthesis, that is to say:

$$y[n] = b_0 w[n] + b_1 x[n-1] + a_1 b_1 w[n-2] + a_2 b_1 w[n-3] + b_2 w[n-2]$$

New coefficients have been created, such as a_1b_1 and a_2b_1. As for the VHDL implementation, the first formula has been adopted. This choice allows a more precise calculation of the output, but it also results in a greater area, due to a further multiplier to be inserted. The latter equation, would have been more convenient from the point of view of area minimization, but not as good as the first one regardin precision.

after applying the modifications induced by the look-ahead technique, it is possible to observe that the feedback network has not been changed: on the other hand, the feedforward one has been replaced according to the derived equation. This has caused an inevitable increasing in area occupation, with the addition of 2 multipliers, 1 adder and 1 subtractor with respect to the basic implementation. Moreover, two new registers have been introduced, one delaying the input $x[n]$ in order to create $x[n-1]$, and another providing $w[n-3]$ from $w[n-2]$. These two sequential elements are depicted in green in figure 3.1.

Along with the increased area, a new critical path has been created by applying this technique, which is the one going from the register providing $sw[2]$ to the output register. For this reason, pipelining has been adopted as a solution. Thus, a 4-stage pipelined architecture has been derived in order to reach the best performances. This final implementation has allowed to obtain a path's delay between two sequential elements which is lower or equal to the iteration bound T_∞. This last parameter is defined as the maximum among the loop bound and in this case is :

$$T_\infty = \frac{2T_{mult}+T_{sub}}{1} = 2T_{mult} + T_{sub}$$

Without pipelining, the critical path delay would be much greater than the iteration bound, and this means that there can be optimizations to be applied. After pipelining the netlist, it can be observed that a path's delay between two registers is never greater than the iteration bound, so no further modifications are feasible.

As regards the rest of the circuit, an input and an output one have been exploited to store both DIN and $DOUT$. Moreover, a simple control unit has been designed in order to make the pipeline work properly. It uses delayed versions of the input VIN as enable signals to the pipeline registers. If they are asserted (equal to '1'), the sequential elements provide in output what they have in input, otherwise they give the previous values as outputs. This means that, if an input arrives and VIN is '0', a sort of stall is introduced in the pipeline and DIN is not processed.

3.1.2 First Design Simulation

After designing the architecture, a ModelSim simulation has been carried out. Files uploaded on Portale della Didattica, such as "*data_maker_new.vhd*", "*data_sink.vhd*", "*clk_gen.vhd*" and "*tb_fir.v*", have been modified according to our architecture and exploited. They are all contained in the folder named *tb*. Moreover, following the indications written in "*Documents.pdf*", two scripts have been prepared in order to make the process faster and adaptable to other eventual designs. These scripts are "*compile.sh*", to be run in a terminal inside the *sim* folder, and "*run.do*", to be run in the ModelSim shell after it opens up. The first one has allowed

compiling the *.vhd* and *.v* files, as well as initializing the simulation. The latter scripts is very simple and it has been used to add signals to the wave panel and run the simulation for 3µs. A similar approach has been used also for post-synthesis and post-routing simulation for switching activity.

At the end of the simulation, the file "*results.txt*" has been compared to the one generated by the C model implementation, "*output.txt*", with the Linux shell command "$diff$". No differences have been reported, which is exactly what it had been expected, since the chosen implementation could allow a better precision.

In order to have a better view on how the circuit works, the waveforms panel have been observed. When DIN is provided alongside with VIN, it is considered as a valid input and it will be provided to the internal datapath by the output of the initial register. Then, the four delayed replicas of VIN ($Vin\_delayed\_1$, $Vin\_delayed\_2$, $Vin\_delayed\_3$ and $Vin\_delayed\_4$) are set to '1' in their respective clock cycle in order to make both the shifting registers and the pipeline work properly. After these phases, the output $DOUT$ is provided at the output asserting $VOUT$. From the clock cycle in which the input is provided and the one $DOUT$ is given, 4 additional clock cycles are present. Thus, there are 3 more clock cycles with respect to the basic IIR Filter implementation. However, once the pipeline is full, one output at each period is produced.

On the other hand, if VIN is equal to zero when a certain input is provided, none of its internal replicas are activated in the next clock cycles. This causes a stall in the pipeline and in the shifting and the output will be a random one with $VOUT$ not asserted, which means it is not a valid one.

3.2 Logic Synthesis

After the first design simulation phase, Synopsys Design Compiler has been exploited in order to carry out the logic synthesis. The folder used to perform the following operations has been named *syn*. After this first part, a switching-activity-based power estimation has been carried out.

3.2.1 Synthesis using Synopsys Design Compiler

In the above mentioned folder, Synopsys Design Compiler has been launched. Indications on how to setup this phase have been taken from the file "*Documents.pdf*" and they have been used to create some *.tcl* scripts.

The first goal of this part is to obtain the minimum clock period for the architecture. In order to do this, the synthesis must be performed multiple times, each time deleting previous synthesis files and changing the clock constraint. For this purpose, the script "*syn.tcl*" has been run multiple times, each time adding the slack reported by the command "*report_timing*" to the previous clock period value, until a null slack has been obtained.

After few iterations, a clock period of 2.21ns has been achieved. Always with this synthesis, an area report has been generated using the script "*post_syn.tcl*", which contains a small

procedure that can take a string as input and set the file name according to this parameter. Looking at the generated file, "*area_reportfM.txt*", the various contributions to the total area have been observed:

- Combinational area: 2965.102004 μm^2
- Buf/Inv area: 184.870000 μm^2
- Noncombinational area: 833.378024 μm^2
- Total cell area: 3798.480028 μm^2

Then, the file "*syn.tcl*" has been modified in order to synthesize the same circuit with a clock contraint equale to four times the one used before. Thus, the clock period has been set to 8.84ns and the synthesis has been run. Also in this case, an area report has been generated ("*area_reportfM_4.txt*") and it shows the following characteristics:

- Combinational area: 2781.030011 μm^2
- Buf/Inv area: 173.166000 μm^2
- Noncombinational area: 833.378024 μm^2
- Total cell area: 3614.408035 μm^2

It can be observed that the second synthesis' total area is lower than the first one. In particular, while the noncombinational area is the same in both, the combinational one is the contribute that changes the most. This is probably related to the less strict contraint on the clock period, which allows mapping the operations on smaller blocks.

At the end of this phase, files useful for the switching acticity estimation have been produced. For this purpose, the same script "*post_syn.tcl*" has been used in order to generate $IIR\_FILTER\_ADVANCED.sdc$, $IIR\_FILTER\_ADVANCED.sdf$ and IIR_FILTER_ADVAN These files have been put in the *netlist* folder and have been exploited in the next steps. As regards the script, it has been run only once producing both the area report and the already mentioned files.

3.2.2 Post-synthesis switching-activity-based power consumption estimation

This phase aims both at verifying that the post synthesis circuit behaves as the one designed and at evaluating the switching activity, which will be used for the power consuption estimation by Synopsys Design Compiler.

First of all, a ModelSim simulation has been carried out by means of scripts in the folder *sim*, which have created with the indications reported in the file "*Documents.pdf*". The first one to be run inside the Linux shell is "*compile_postsyn.sh*", which allows compiling the *tb* folder files, the post-synthesis netlist and initialize the simulation. The second one is "*run_postsyn.do*", in which a *.vcd* file is opened and simulation launched for 3µs.

Power Groups	Internal Power	Switching Power	Leakage Power	Total Power
io pad	0.0000	0.0000	0.0000	0.0000
memory	0.0000	0.0000	0.0000	0.0000
black box	0.0000	0.0000	0.0000	0.0000
clock network	0.0000	0.0000	0.0000	0.0000
register	127.5701	41.0388	1.3869e+04	182.4783
sequential	0.0000	0.0000	0.0000	0.0000
combinational	209.5592	189.9424	5.7214e+04	456.7157
Total	337.1293µW	230.9812µW	7.1084e+04nW	639.1940µW

Table 1: switching-activity-based power report for $\frac{f_M}{4}$ IIR gilter design

After checking that the waveforms in the wave panel corresponds to those obtained for the first design simulation, the switching-activity-based power estimation has been performed. As first step, the *.vcd* file has been converted to a *.saif* file and saved in the *saif* folder. Then, "*power.tcl*" has been executed inside Synopsys Design Compiler. This script contains a small procedure which reads the *.saif* file and the post-synthesis netlist and it produces a power report. The name of this output file is set according to the input string parameter that the procedure receives. In this case, it is "*power_reportfM_4.txt*".

Various power contributions have been gathered from this report and here are shown:

- Cell Internal Power = 337.1294µW;
- Net Switching Power = 230.9812µW;
- Total Dynamic Power = 568.1106µW;
- Cell Leakage Power = 71.0838µW;

As regard the Total Power, the main contribution is surely the Total Dynamic Power, which is much greater than the Cell Leakage Power.

3.3 Place and Route

During this last phase, the main goal has been to use Cadence Innovus to perform the place and route operations. These steps have been performed in the *innovus* folder and applied on the IIR Filter netlist derived from the synthesis with clock frequency equal to $\frac{f_M}{4}$. Then, a switching-activity-based power estimation has been carried out in a similar way as mentioned in the previous sections, but using Innovus.

3.3.1 Place and Route with Cadence Innovus

Following the commands elencated in the file "*Documents.pdf*", Cadence Innovus has been exploited in order to obtain the placed and routed design for the IIR filter. The files "*design.globals*" and "*mmm_design.tcl*" have been copied in a folder called innovus and the program launched from this directory. During the execution of the various steps mentioned

in the *.pdf* file, in particular after the route phase, *.slk* and *.tarpt* reports concerning Setup and Hold time have been generated. These have been checked in order to verify that no negative slack had been obtained. After that, both connectivity and geometry have been checked and no errors have showed up. In the end, the post place and route verilog netlist ($IIR\_FILTER\_ADVANCED.v$) has been saved along with $IIR\_FILTER\_ADVANCED.sdf$. These last two file have been exploited in the next phase.

3.3.2 Post place and route switching-activity-based power consumption estimation

This phase goal has been to verify the correct behaviour of the post place and route netlist and to estimate the power consumption based on the information gathered abbout the switching activity. For the first part of this phase ModelSim has been used, and then Cadence Innovus has been exploited for the power consumption calculation. This is very similar to what has been done for the post-synthesis netlist.

First of all, a simulation has been carried out using ModelSim by means of scripts created with the indications reported in the file "*Documents.pdf*". The first one to be run inside the Linux shell has been "*compile_postroute.sh*", which allows compiling the tb folder files, the post place and route netlist and initialize the simulation. The second one has been "*run_postroute.do*", in which a *.vcd* file is opened and simulation launched for 3µs.

After checking the corrispondence between the waveforms of the current simulation and the first design simulation ones, the switching-activity-based power estimation has been performed. After reopening the innovus design saved in the previous steps, using the created *.vcd* file, a power report has been produced using Cadence Innovus.

Various power contributions have been gathered from this report and here are shown:

- Total Internal Power = 0.95788998µW;
- Total Switching Power = 0.71007244µW;
- Total Leakage Power = 0.07131811µW;
- Total Power = 1.73928053µW;