Politecnico di Torino

III Facoltà di Ingegneria

Design, Implementation and Optimization of a second order, 9-bit IIR Filter

Master degree in Electriconic Engineering

Integrated Systems Architectures

Authors: Group 34

github repository link :

https://github.com/alessionaclerio22/Integrated-Systems-Architecture-Labs

Simone Di Blasi, Stefano Floridia, Alessio Naclerio

Contents

1 Reference model development **1**
- 1.1 Filter design using Matlab . . . 1
- 1.2 Fixed point C model . . . 3
  - 1.2.1 THD evaluation . . . 3
- 1.3 Comparison . . . 4

2 Basic IIR Filter **5**
- 2.1 VLSI implementation . . . 5
- 2.2 Architecture Development . . . 5
  - 2.2.1 Design Description . . . 5
  - 2.2.2 First Design Simulation . . . 6
- 2.3 Logic Synthesis . . . 7
  - 2.3.1 Synthesis using Synopsys Design Compiler . . . 7
  - 2.3.2 Switching-activity-based power consumption estimation . . . 8
- 2.4 Place and Route . . . 9
  - 2.4.1 Place and Route with Cadence Innovus . . . 9
  - 2.4.2 Post place and route switching-activity-based power consumption estimation . 10

3 Look-Ahead IIR Filter **11**
- 3.1 Architecture Development . . . 11
  - 3.1.1 Design Description . . . 11
  - 3.1.2 First Design Simulation . . . 13
- 3.2 Logic Synthesis . . . 14
  - 3.2.1 Synthesis using Synopsys Design Compiler . . . 14
  - 3.2.2 Post-synthesis switching-activity-based power consumption estimation . . . 15
- 3.3 Place and Route . . . 16
  - 3.3.1 Place and Route with Cadence Innovus . . . 16
  - 3.3.2 Post place and route switching-activity-based power consumption estimation . 16

CHAPTER 1

Reference model development

The 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 has started by describing a Matlab model of the filter and testing it with a signal made up of two sinusoidal waves at different frequencies. The obtained results have been compared with those retrieved from a fixed-point model written in C. Finally, before moving on to the VSLI implementation, it has been evaluated whether the THD of the C implementation was lower than -30dB.

1.1 Filter design using Matlab

The design has started by defining the order and the number of bits of the filter. Using the formulas provided in the documentation of the assignment, a second order, 9-bit IIR filter has been obtained. Passing these parameters to "*myiir_design.m*" and setting the desired cut-off and sampling frequency, the transfer function of the designed filter and the quantized transfer function have been retrieved. The graph of both is shown in figure 1.1.

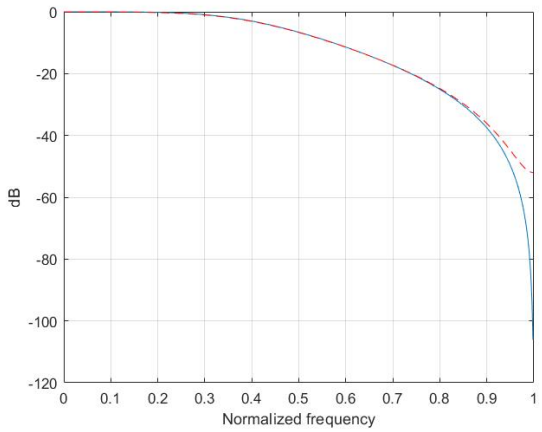

Figure 1.1: IIR Transfer Function (the quantized one is in red).

"*myiir_design.m*" also allows to save the coefficients a and b as 9-bit integers and as 9-bit real values. The first ones have been stored in *ai* and *bi*, the latter in *aq* and *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

Coefficients

In order to test the behavior of the filter, a signal has been created in "$my\_iir\_filter.m$". It consists of two sinusoidal waves at two different frequencies, one in band with a frequency of 500Hz and one out of band with a frequency of 4.5KHz. The values of the resulting signal x and the coefficients aq and bq, generated by the IIR filter design function, have been used in the Matlab "filter" function. This function allows to filter the input data x using a rational transfer function defined by the numerator and denominator coefficients bq and aq. 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) is shown in figure 1.2.

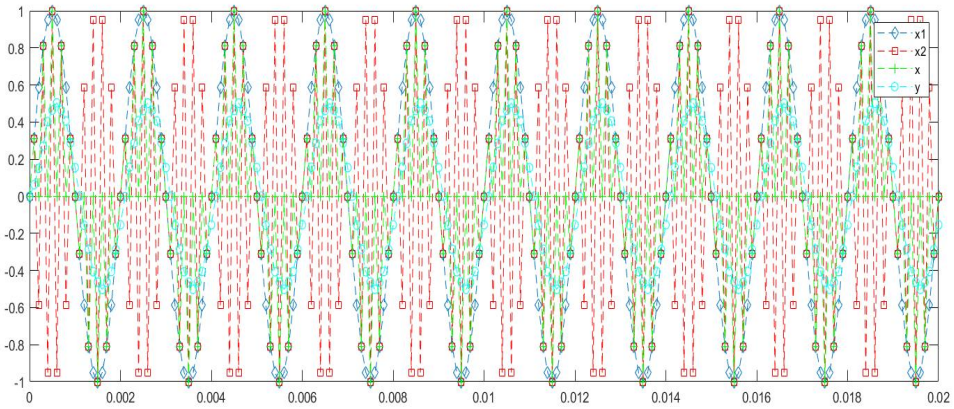

Figure 1.2: Inputs and output waveforms.

In addition, the input and output samples have been saved in two text files, "$samples.txt$" and "$resultsm.txt$". Using the quantized output data to evaluate the total harmonic distortion, a THD value equal to -52.53dB is obtained. The graph is shown below in figure 1.3.

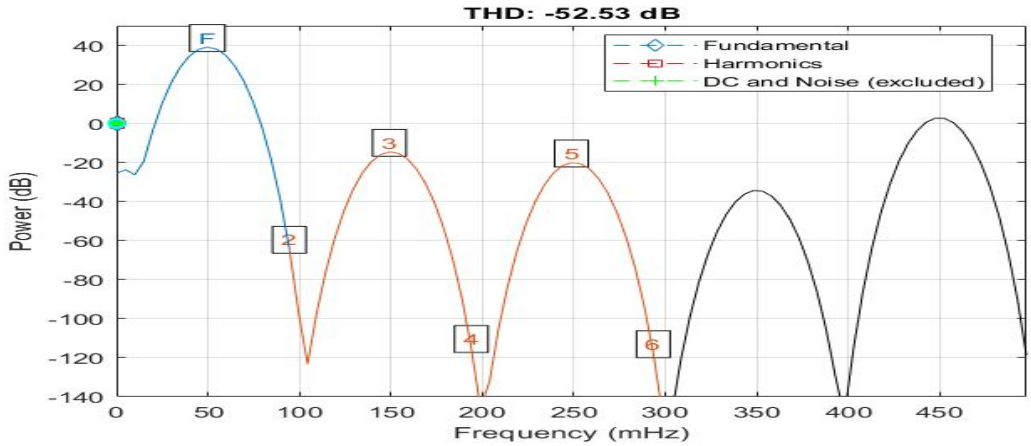

Figure 1.3: THD of the Matlab model

1.2 Fixed point C model

In this section, a fixed-point model of the filtering operation has been implemented by means of a program written in C. Direct form II (canonical direct form) has been chosen for the IIR filter architecture. Hence, the main formulas for this structure are:

$$w = x[n] - A1 * w[n-1] - A2 * w[n-2]$$
$$y[n] = B0 * w[n] + B1 * w[n-1] + B2 * w[n-2]$$

In order to emulate the behavior of the filter, the C program 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)

All these parameters have been taken from the Matlab results. Then, it continuously reads the input samples file generated by the Matlab model, "*samples.txt*", and calculate the filtered results. In order to do this with a fixed point approach, it uses a 8-bit right shift after the multiplication between the coefficients and the variable sw, which acts a shift array for the various w contributions. The C code also evaluates feed-back and feed-forward sum (fb and fb) in order to calculate the final result. The input x is added to fb, the resulting w is multiplied by $b0$ giving y, which is also the final result after being added ff. In the end, the program prints the results obtained in another file, "*output.txt*".

1.2.1 THD evaluation

Taking the output file of the fixed-point model made in C, "*output.txt*", the resulting THD has been evaluated. The Matlab function for the total harmonic distortion, *thd*, accepts an array made up of the values present in the text file mentioned before. The graph of the function is shown in figure 1.4.

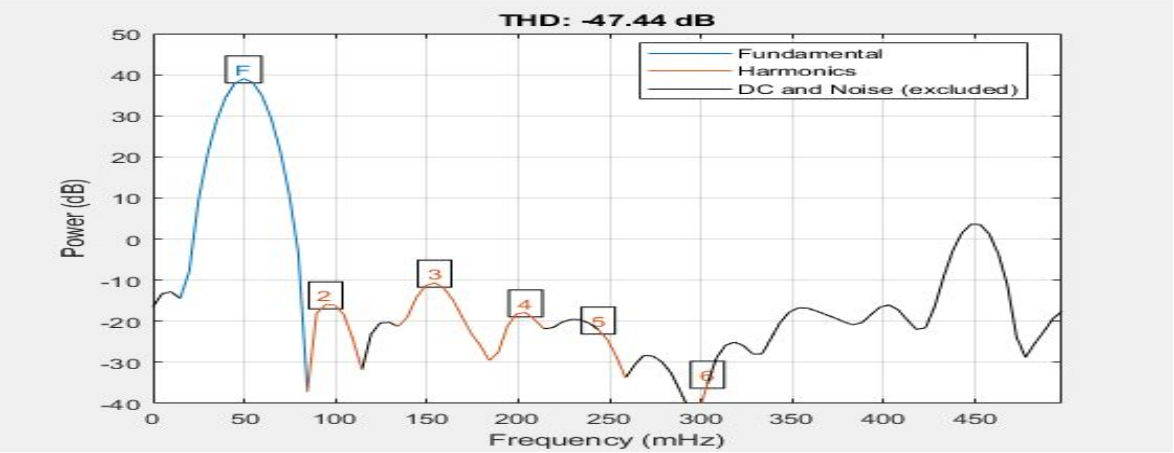

Figure 1.4: THD of the fixed-point model

As it can be seen from the image, the resulting THD is -47.44dB, which is lower than -30dB, as requested by the assignment. In order to possibly reduce the internal area, a lower-bit right shift in the C code has been tried. Unfortunately, this has led to a final THD higher in absolute value than the one obtained with the 9-bit shift. In the end, the initial configuration has been chosen also for the VLSI implementation.

1.3 Comparison

It can be seen that in the two models used, Matlab and C, different results have been retrieved. This can be seen both by directly analyzing the values present in the output files but also by looking at their THDs. This is due to the fact that a fixed-point approach has been used for the C model, while the Matlab one uses floating point representation.

CHAPTER 2

Basic IIR Filter

2.1 VLSI implementation

The goal of this part is to develop a VHDL model following the architecture requirements for the IIR filter described in the previous chapter. The structure has been simulated, synthesized, placed and routed. Moreover, a 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 by the laboratory assignment.

2.2 Architecture Development

2.2.1 Design Description

The IIR filter structure has been derived from the C model implementation. Both the internal architecure and the external interface parallelism is 9 bits. As already mentioned before, the structure of the filter has been designed to implement the direct form II, which relies on the following equations in order to obtain the result:

$$w = x[n] - A1 * w[n-1] - A2 * w[n-2]$$
$$y[n] = B0 * w[n] + B1 * w[n-1] + B2 * w[n-2]$$

As regards the VHDL entity, it receives the sample DIN, the coefficients $A1$, $A2$, $B0$, $B1$ and $B2$ on 9 bits and the enable signal VIN as inputs. It produces $DOUT$ as output result and $VOUT$ as output valid signal. As requested by the assignment, two registers have been placed, IN_REG and OUT_REG, the first storing the input and the latter containing the output.
With respect to the VHDL netlist described in the file "$IIR\_filter.vhd$" and shown in figure, the signals Din_out_reg, $Dout\_ext$, w and $sw[i]$ correspond respectively to $x[n]$, $y[n]$, $w[n]$, $w[n-i]$ in the previous formula. Regarding the generation of signals $sw[i]$, which are the delayed replicas of $w[n]$, two registers have been exploited. The contributions related to the feed-forward and feed-back networks have been computed in the following manner:

$$psum\_b = A1 * sw[0] + A2 * sw[1]$$
$$psum\_f = B1 * sw[0] + B2 * sw[1]$$

Since the result of a multiplication is on 18 bits, it should be rigth-shifted of 8 positions, which corresponds to $NBIT - 1$. This is performed each time such an operation is required in the architecture by selecting bits ranging from $2 * NBIT - 2$ down to $NBIT - 1$. Then, the signal $psum\_b$ has to be

subtracted from Din_out_reg in order to produce w, while $psum\_f$ has to be added to the multiplication of w and $B0$ to generate $Dout\_ext$.

Moreover, a small control unit has been desinged in order to properly process input data. If the enable signal VIN is '1', in the next clock cycle $Vin\_delayed$ will be '1', thus allowing the shift of data between w, $sw[0]$ and $sw[1]$. In this case, the output $DOUT$ is a valid one, so $VOUT$ will be set to '1' along with the providing of $DOUT$. On the other hand, if VIN is equal to '0', it means that DIN must not be processed, so $Vin\_delayed$ will be '0', thus preventing the shift of data. Also, $VOUT$ will be '0', meaning that $DOUT$ is not a valid one. An image of the filter architecture is shown in figure 2.1.

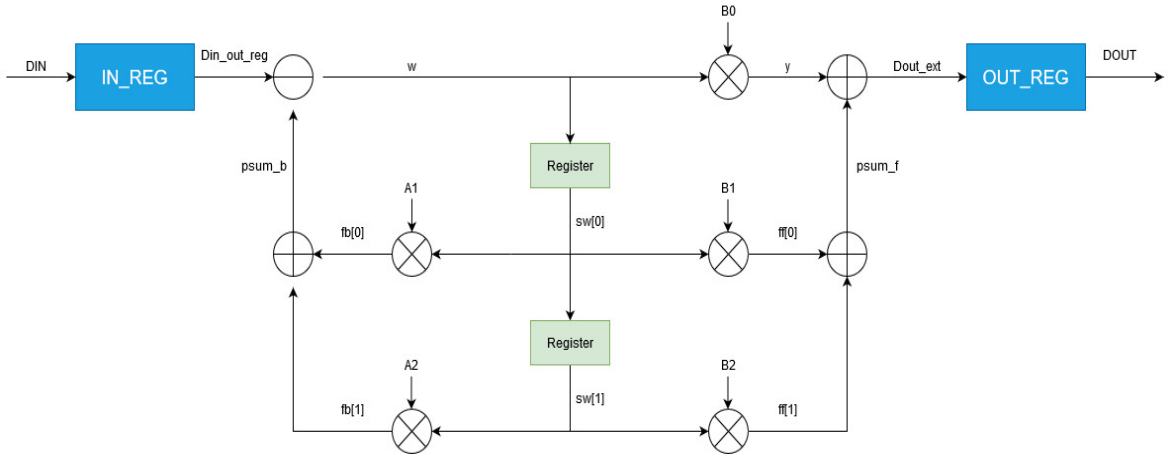

Figure 2.1: IIR Filter architecture

2.2.2 First Design Simulation

After the design phase, the filter has been simulated using ModelSim, in order to compare the results produced by the VLSI implementation with the ones obtained with the C model. They have to fully correspond. The simulation has been carried out following the steps showed in "*Documents.pdf*", creating a script called *compile.sh*, to be run in a terminal inside the *sim* folder, and another one called *run.do*, to be run inside ModelSim. The first one allows compiling *.vhd* and *.v* files and initialize the simulation, while the second is used to add signals to the wave panel and run the simulation. The files used to simulate the filter have been taken from the ones uploaded on the Portale della Didattica, such as "*data_maker_new.vhd*", "*data_sink.vhd*", "*clk_gen.vhd*" and "*tb_fir.v*". They have been properly changed in order to test our specific component. The simulation results have been saved in "*results.txt*" and, if compared 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. The Linux shell command $diff$ has been used for this purpose.

As requested by the laboratory assignment, an extract of the waveform of the simulation is shown in figure 1.1, highlighting that the filter is working correctly even when VIN is 0. As a matter of fact, when $DOUT$ is equal to -18, $VOUT$ is '0', because this value must not be considered as valid. Moreover, in figure 2.3, the behaviour of an intermidiate change of VIN is shown. It can be seen how the shifting of data is stopped and $VOUT$ is correctly set to '0'. The first simulation has been carried out with the testbench files mentioned before and it has been used to generate "*results.txt*". The latter simulation has been done using another testbench "*tb_iir.vhdl*".

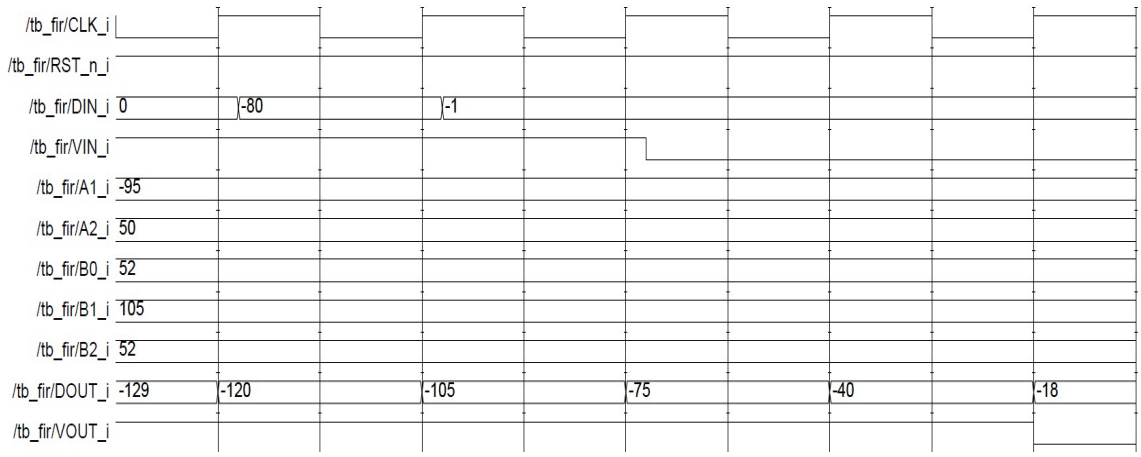

Figure 2.2: IIR Filter waveform

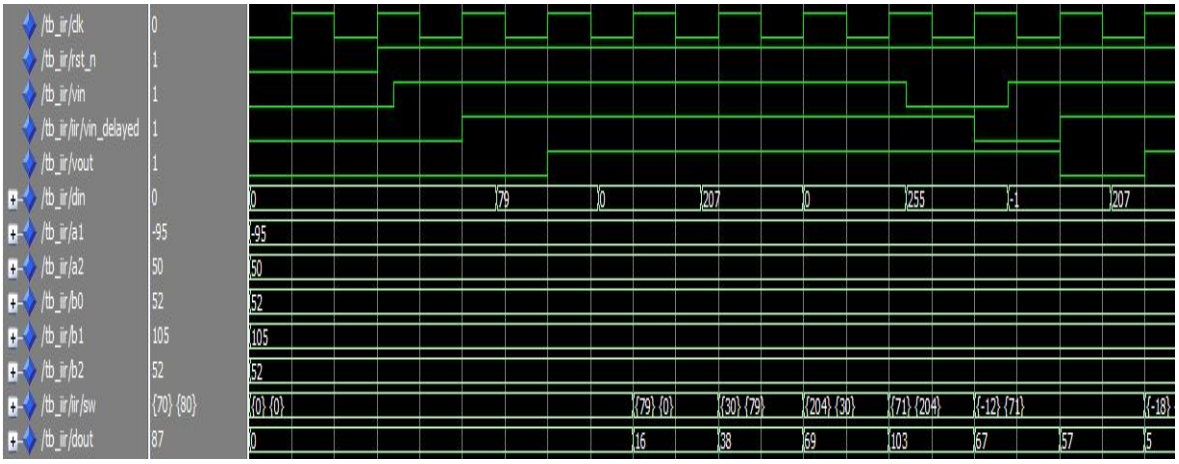

Figure 2.3: IIR Filter waveform

2.3 Logic Synthesis

After the design simulation, the next step requires performing the logic synthesis with Synospys Design Compiler, setting as work environment the folder *syn*. Moreover, a switching-activity-based power estimation has been carried out.

2.3.1 Synthesis using Synopsys Design Compiler

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

The purpose of the logic synthesis is to find the maximum clock frequency (f_M), which is the one at which the command *report_timing* indicates that the slack is zero followed by the string (MET). In order to do that, the script "*syn.tcl*" has been run several times. This script allows performing the synthesis according to the commands elencated in "*Documents.pdf*". Each time this operation has been done, the value of the obtained slack has been added to the previous value of the clock period,

until a synthesis with slack equal to zero has been reached. The final clock period is 3.37ns, so the maximum frequency is $f_M = 296.74$MHz.

With this frequency, an area report has been generated running the script "*post_syn.tcl*", which contains a small procedure that can take a string as input and set the report file name according to this parameter. The generated file is called "$area\_report fM.txt$" 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 greater ($T_{clk} = 13.48$ns and $f = 74.18$MHz). A new area report has been generated with the name "$area\_report fM\_4.txt$", and some differences can be observed with respect to 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 total area is lower in this second sythesis and, in particular, the combinational area is what has been reduced the most. This may be related to the less strict limit on the speed of the circuit, 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 $IIR\_filter.sdc$, $IIR\_filter.sdf$ and $IIR\_filter.v$. These files have been put in the *netlist* folder and have been exploited in the next step. As regards the script, it has been run only once producing both the area report and the already mentioned files.

2.3.2 Switching-activity-based power consumption estimation

The goal of this step is to perform a switching-activity-based power estimation for the $f_{clk} = f_M/4$ circuit and verify that it behaves as expected using ModelSim.

According to the information given in the file "*Documents.pdf*", a ModelSim simulation has been launched using scripts saved in the folder "*sim*". The first script that has been run inside the Linux shell is "*compile_postsyn.sh*" for compiling the *tb* folder files, the post-synthesis netlist and initialize the simulation. Then, the script "*run_postsyn.do*" has been launched in ModelSim. It opens a file *.vcd* and allows the simulation to start and last for 3µs. After looking at the waves generated by the simulation and being sure that they were correct, the switching activity power estimation has been performed.

After that, the folder *syn* has been selected as working directory and the previously generated *.vcd* has been converted into a $.saif$ file and saved inside the $saif$ directory. The following steps have been performed in the Synopsys Design Compiler where the script "*power.tcl*" has been run. This script reads

the file *.saif* together with the post-synthesis netlist and produces a power report named basing on the string that it receives in input. In this case, the output file has been called "*power_reportfM_4.txt*". The main elements to be observed in 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

It can be noticed that the main contribution to the Total Power is the Total Dynamic Power, while the Cell Leakage Power is the less significant one. Moreover, in Table 2.1, the detailed values of the power report are reported.

Power Groups	Internal Power[µW]	Switching Power[µW]	Leakage Power[nW]	Total Power[µW]
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	31.3364	5.3947	3.3572e+03	40.0883
sequential	0.0000	0.0000	0.0000	0.0000
combinational	140.9034	145.5115	3.6960e+04	323.3747
Total	172.2397	150.9062	4.0317e+04	363.4629

Table 2.1: post-synthesis switching-activity-based power report for $\frac{f_M}{4}$ IIR filter

As it can be seen from Table 2.1, the major contribution to the total power is given by combinational blocks, while register ones' is much lower.

2.4 Place and Route

This last part is devoted to place and route the filter using Cadence Innovus. Before starting the Innovus environment, a new directory called "*innovus*" has been created where all the results have been saved. Furthermore, the obtained netlist has been simulated in order to check its behaviour and power consumption has been estimated based on the switching activity.

2.4.1 Place and Route with Cadence Innovus

Following the commands 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 the *innovus* folder and the program launched from this directory. During the execution of the various steps, 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. An area report has been generated using the option *GateCount* and the correspondent file named "*IIR_Filter.GateCount*". I shows the following parameters:

- Gates = 2496;
- Cells = 1007;
- Gate Area = $0.7980\mu m^2$;
- Area = $1992.1\mu m^2$;

In the end, the post place and route verilog netlist ($IIR\_FILTER.v$) has been saved along with $IIR\_FILTER.sdf$. These last two files have been exploited in the next phase.

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

The goal of this phase to verify the correct behaviour of the post-place-and-route netlist and to estimate the power consumption based on the information gathered about the switching activity. For the first part of this phase ModelSim has been used, and then Cadence Innovus has been reopened 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 = 0.76841301mW;
- Total Switching Power = 0.63860564mW;
- Total Leakage Power = 0.03965208mW;
- Total Power = 1.44667073 mW;

As expected, the total power is much higher than the correspondent value retrieved with Design Compiler. Moreover, in Table 2.2 the detailed values are shown. Also in this case, the major contribution is given by the combinational blocks.

Power Groups	Internal Power[mW]	Switching Power[mW]	Leakage Power[mW]	Total Power[mW]
Sequential	0.0458	0.01073	0.003357	0.05989
Macro	0.0000	0.0000	0.0000	0.0000
IO	0.0000	0.0000	0.0000	0.0000
Combinational	0.7226	0.6279	0.0363	1.387
Clock (Combinational)	0.0000	0.0000	0.0000	0.0000
Clock (Sequential)	0.0000	0.0000	0.0000	0.0000
Total	0.7684	0.6386	0.03965	1.447

Table 2.2: post-place-and-route switching-activity-based power report for $\frac{f_M}{4}$ IIR filter

CHAPTER 3

Look-Ahead IIR Filter

The aim of this part of the laboratory is to develop, simulate, synthesize, place and route the previous 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 by the laboratory asignment.

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 register 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 keep the previous values. 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.

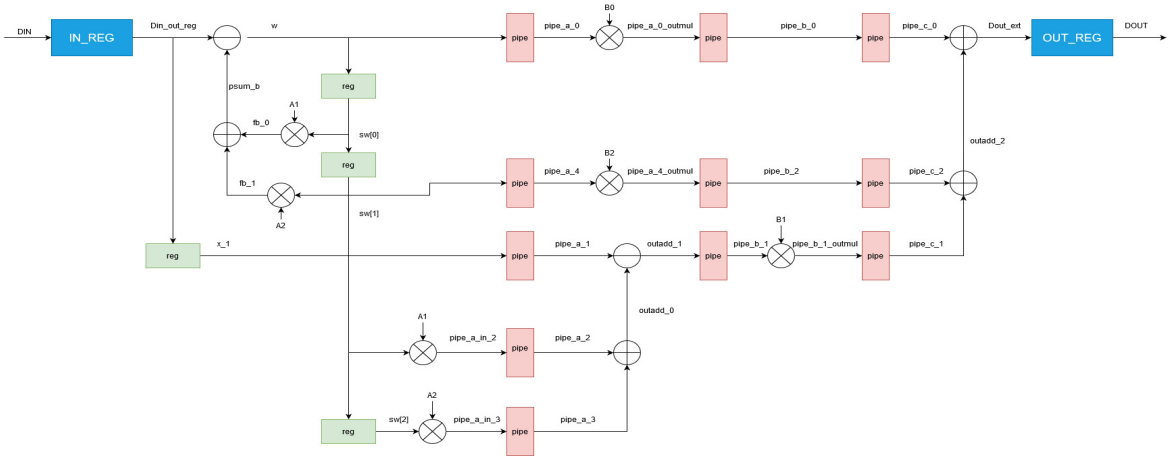

Figure 3.1: J-Look-ahead IIR Filter architecture

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. Figure 3.2 shows this behaviour. This simulation has been carried out using the testbench file called "$tb_i ir_a d.vhd$".

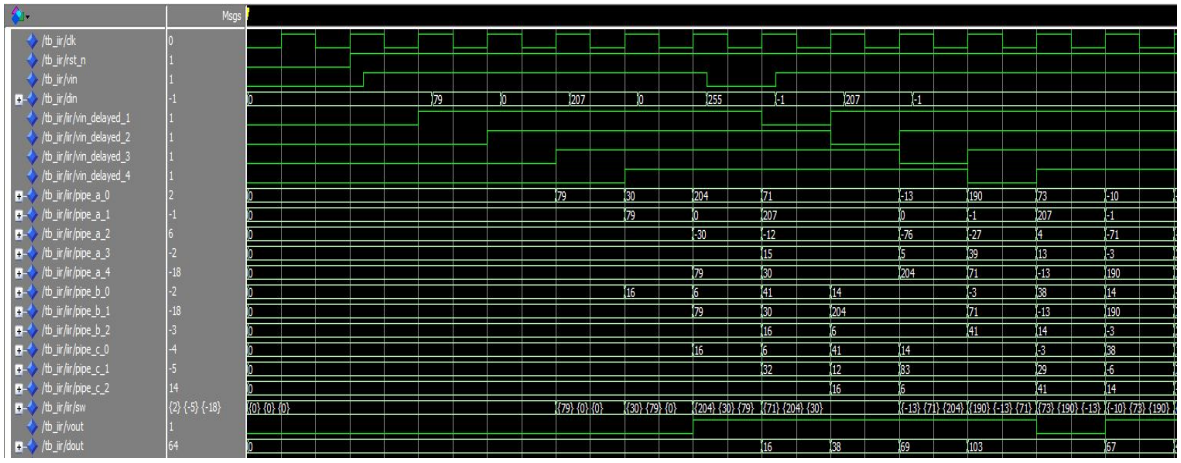

Figure 3.2: J-Look-ahead IIR Filter Simulation

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, so f_M is equal to 452.48MHz. 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, "$area\_report fM.txt$", 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 constraint equale to four times the one used before. Thus, the clock period has been set to 8.84ns($f = 113.12$MHz) and the synthesis has been run. Also in this case, an area report has been generated ("$area\_report fM\_4.txt$") 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 $IIR\_FILTER\_ADVANCED.v$. 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.

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. From Table 3.1, it can be observed how the ratio between the contribution of the combinational group and the register one is much higher than in the basic IIR filter implementation.

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

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. An area report named "$IIR\_FILTER\_ADVANCED.GateCount$" has been generated and it shows the following paramenters:

- Gates = 4493;
- Cells = 1743;
- Gate Area = $0.7980\mu m^2$;
- Area = $3585.4\mu m^2$;

As expected, the total area is larger than the one obtained with the basic filter implementation. 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 re-opening 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 = 0.72218366mW;
- Total Switching Power = 0.46988709mW;
- Total Leakage Power = 0.07051136mW;
- Total Power = 1.26258211mW;

Power Groups	Internal Power[mW]	Switching Power[mW]	Leakage Power[mW]	Total Power[mW]
Sequential	0.1698	0.04578	0.01387	0.2294
Macro	0.0000	0.0000	0.0000	0.0000
IO	0.0000	0.0000	0.0000	0.0000
Combinational	0.5524	0.4241	0.05664	1.033
Clock (Combinational)	0.0000	0.0000	0.0000	0.0000
Clock (Sequential)	0.0000	0.0000	0.0000	0.0000
Total	0.7222	0.4699	0.07051	1.263

Table 3.2: post-place-and-route switching-activity-based power report for $\frac{f_M}{4}$ IIR filter

As expected, the total power is much higher than the correspondent value retrieved with Design Compiler. A comparison with the values obtained for the basic implementation can be done. It can be observed that the total power for the J-look-ahead circuit is lower. Even though the values of the sequential group are higher than the correspondent ones in the basic IIR filter, the same thing does not happen with the combinational group. These results are lower than the ones retrieved before, leading to a lower total power.