Politecnico di Torino

III Facoltà di Ingegneria

Exercises and Homeworks for the course Integrated Systems Architecture

Master degree in Electrical Engineering

Authors: Group 20

Alessandro Lovesio, Lorenzo Poloni, Simone Pont

November 14, 2020

Many thanks to Prof. Mariagrazia Graziano for providing us with this template.

Contents

Link to our group repository:
`https://github.com/AleLovesio/Laboratory_ISA_Group20.20`
Direct link to the folder of Lab 1 in the same repository:
`https://github.com/AleLovesio/Laboratory_ISA_Group20.20/tree/main/Lab1`

0.1 Introduction

The aim of this laboratory is to design a digital filter with a cut-off frequency of 2 kHz and a sampling frequency of 10 kHz. Starting from the guidelines written on the assignment, we defined the characteristics of the filter to implement. In particular we obtained:

- Infinite Impulse Response (IIR) filter
- Order of the filter: $N = 2$
- Number of bits: $n_b = 14$

First, we implemented and tested a reference model written in C language to find the best internal parallelism which lets us satisfy the THD requirement. After that, we developed a VHDL version of the architecture of the filter using, as components, adders and multipliers described behaviorally. We also modified the given testbench to adapt it at our filter architecture. In this way, we had the possibility to check the correct behavior of what we had implemented. Once the structure was tested, we synthesized it using a $45nm$ technology and we obtained the requested timing, area and power results. The following step was to perform the place and route of the filter with *Cadence SOC Innovus* which allowed us to obtain the final values needed for area, timing and power after the physical design. During all steps we always checked that our design worked properly and had no violations.
Finally we tried to improve our initial structure using look-ahead, retiming and pipelining techniques. This resulted in an optimized version of our filter with better performances. Also for this improved architecture, we followed the previous step of synthesis and place and route to extract realistic timing, area and power values.
In the following sections, we will analyze more in details the different steps we have just introduced and we will also compare the obtained results for both the initial and the optimized version of our filter. One last thing to clarify: all the data are represented as two's complement normalized fixed-point values, so the weight of the most significant bit (MSB) is -2^0, while the weight of the least significant bit (LSB) is 2^{-n_b+1}.

0.2 Reference Model Development

In this section we describe in details the functional model that we have developed for our filter. We used the given MATLAB script (”myiir_design”) to generate the coefficients which are rounded to obtain the integer version that will be further converted into the quantized form. The values obtained starting from the filter parameters are the following:

- $b_0 = 1692$
- $b_1 = 3384$
- $b_2 = 1692$
- $a_1 = 3028$
- $a_2 = -1604$

We, then, implemented the IIR filter using Matlab. By default, Matlab uses full-precision arithmetic to deal with fixed-point inputs and so the filter uses as many bits for the product, accumulator, and output as needed to prevent any overflow or rounding. The result of full-precision fixed-point filtering comes very close to floating-point. However, in our case the word length is set by the specifications therefore the actual filter won’t have infinite precision. Therefore, coefficients have to be quantized causing part of the initial accuracy to be lost. The filter taps, both integer and quantized, are generated by the ”myiir_design” MATLAB script and the filter’s transfer functions are then calculated and plotted. After this we implemented a C reference model in order to simulate the hardware behavior assuming that the filter works with a certain fixed number of bits. The chosen form to represent the IIR filter is the canonical direct form II that is reported in the following figure 1:

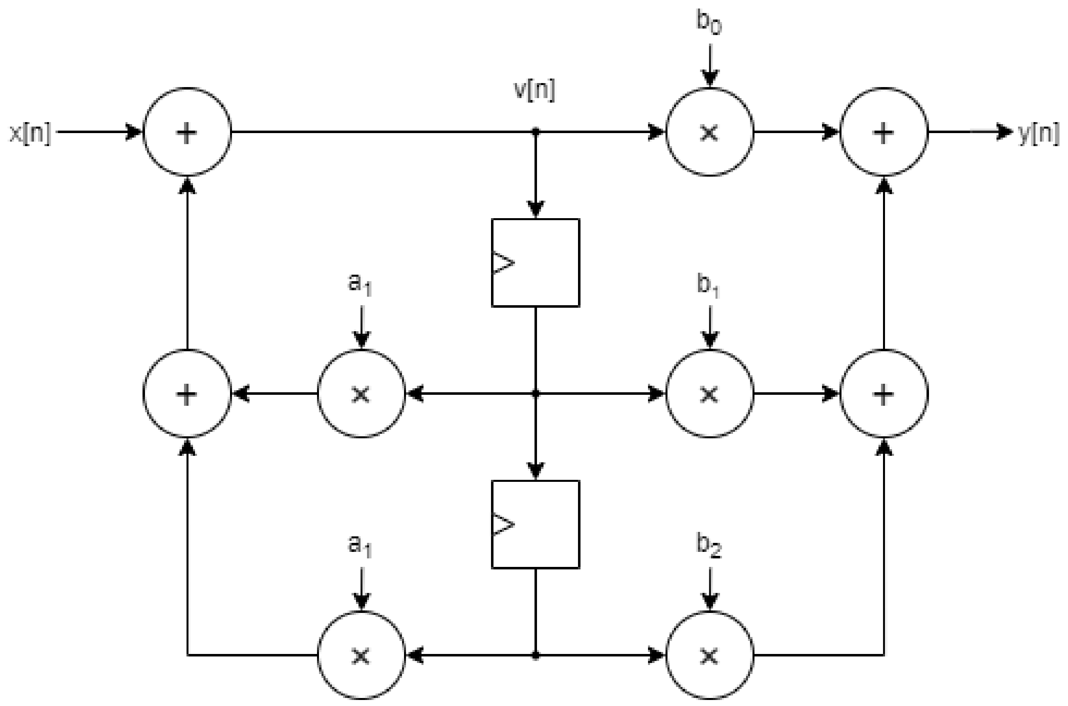

Figure 1: IIR filter in its canonical direct form II

We used a function to represent the filter behavior, whereas the main passes the correct inputs (generated by the MATLAB script) to the function and collects the outputs, writing them on a results file. The reference model implemented in the function is based on the following principles:

- The behavior of middle registers is modeled using a vector of values that are shifted by one position every time a cycle is finished;
- The feed-back and feed-forward loops formed by multipliers and adders are calculated simultaneously and independently by means of a for loop;
- The next value entering in the middle registers is obtained starting from the input and the feed-back;
- The final output is obtained starting from the previous results.

We have to precise that every time a multiplication takes place a correct number of left shifts is applied to the results to correctly model the hardware behavior where the internal parallelism is constant. We have to consider that in the final HW structure all the multiplications give as output a number represented on $2n_b$ bits (where n_b is the internal parallelism) but the following additions cannot use it as input because they work on n_b bits. So we have to truncate the results of the multiplications and maintain only the needed part that in our case corresponds to the MSBs because we are working with the fractional notation. In software, to simulate this behavior, we introduced the previous described shifts that let us realign the results before doing the additions.
Thanks to this reference model, we have obtained the outputs of the simulated real filter that are quite different from the ones observed from the MATLAB model working with no internal parallelism limitations and therefore with no truncation of intermediate results. Evaluating the Total Harmonic Distortion (THD) of the results, we have obtained a value of $-44.6322dB$ that is far from the target of $-30dB$. Hence, we decided to reduce the internal parallelism. In this way, the THD increases but the needed resources are fewer and so area and power are saved and, in case of real implementation, also money!
After some attempts, we found that the best choice is to use an internal parallelism of $n_b = 10$ that produces a THD equal to $-30.1814dB$. We have also calculated the filter's coefficients that we have to use working with 10 bits:

- $b_0 = 26$
- $b_1 = 52$
- $b_2 = 26$
- $a_1 = 48$
- $a_2 = -25$

Before starting to analyze the VHDL implementation of the filter, we have to say that the C reference model was lately modified to simulate also the behavior of the first advanced version of the filter. To do this a new function is created to model the optimized architecture and its behavior is similar to the one described before. The only difference is in the depth of the structure that becomes one level higher (due to the Look Ahead technique). More details about this structure are reported in the dedicated section.

0.3 VLSI Implementation

0.3.1 Synthesis and Simulation

In the following paragraph, we will describe the first basic VHDL implementation of our filter and the passages needed to verify, synthesize and place and route it.
The VHDL description is based on the C code and, thus, must provide the same results. The structure used for the filter is the direct form II. As previously said, the two building blocks of the circuit are the multiplier and the adder, implemented in two separate VHDL files behaviorally. In the main file, the operators are instantiated in order to create a filter that follows the schematic in figure 1. In addition to the combinational part, registers have been added in the form of processes sensitive to the clock and to the reset signals. This means that the reset is asynchronous. Sampling is enabled by the signal VIN which is also propagated along the circuit and is eventually assigned to VOUT which validates the output. As loops are present, special care must be taken to ensure data integrity: when the VIN signal is not asserted and inputs are no longer valid, all registers are deactivated in order not to produce wrong results. All numbers (i.e. coefficients, input and output) are declared as std_logic_vector.
A pre-synthesis simulation has been run using the script "simulation.do". This file makes use of a test-bench to simulate the unit. In the test-bench, clock is generated, coefficients are set, input samples are read from file and fed to the filter and, finally, output is collected and written into another file. This last file contains the result of the simulation that are compared to the ones generated by the C model. All signals are delayed by a time equal to 1 ns. From the simulation, the correct behavior of the design has been verified. Moreover, to ensure that the filter works even in the case of VIN transitions, a new variable "clk_counter" has been declared. This variable is able, when a certain condition arises, to stop the reading from file and set the VIN signal to zero effectively disabling it. After the first simulation, the tool *Synopsys* is used to synthesize the design. To do this, a script called "iir_syn.src" is used. In the file, the circuit is analyzed, elaborated and compiled. However, before compiling, a few constraints are included to take into account the non-ideal effects present in an actual circuit. First of all, a clock of period equal to $18ns$ is created and a clock uncertainty (i.e. jitter) is added. Moreover, input and output delays with respect to the clock rising edge are specified and a load is connected to the output of the filter. The value of clock period (T_{CK}) is chosen according to the specifications: in fact, we have to work with a clock frequency $f_{CK} > \frac{f_{max}}{4}$. From timing analysis, it can be seen that the critical path delay is equal to $T_d = 4.23ns$. With this last figure, we are able to simulate the filter at a clock period equal to $T_{CK} > 4 \cdot t_d = 16,92ns$. We decided to use as clock period an even value to avoid problems in case of working with half T_{CK}, this because *Modelsim* is set with a resolution equal to $1ns$ and it could do some unwanted approximations.
The script then generates the SDF file, the verilog netlist and reports the post-synthesis timing and area. The design is simulated again using *Modelsim* to ensure that the synthesis was successful. Yet another script called "forward.do" is run for this step. In this case, the functional model of the cells in the library used for synthesis (the nangate45) is specified in the vsim command. A VCD file containing the actual activities is generated. From the timing analysis it can be seen that the slack for the filter is $13.73ns$. A realistic power consumption is obtained by converting the VCD file into a SAIF file and then feeding it again to *Synopsys* in order to generate a power report file. Everything is done by the script "backward.src". RTL names are preserved in the netlist.

0.3.2 Place And Route

In this last part of the filter design, the objective is to come up with the definitive layout of the IC that will be sent to the foundry after, of course, having verified it again. Since the real world is far from ideal, results different from the initial simulations are expected in terms of area, timing and power consumption. 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. The Standard Cell Library is again the 45 nm one, which was also used for synthesis. After that, we have to structure the floorplan by setting, among other things, the core aspect ratio. In this case, it was set to 1.0 (i.e. 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 VSS (GND) to the whole die. These lines run along the top layers which are dedicated to power supply. The next step is perform the special routing so that the standard cells can be reached by VDD and VSS wires. Throughout the circuit, higher layers are connected to the lower ones by means of vias.

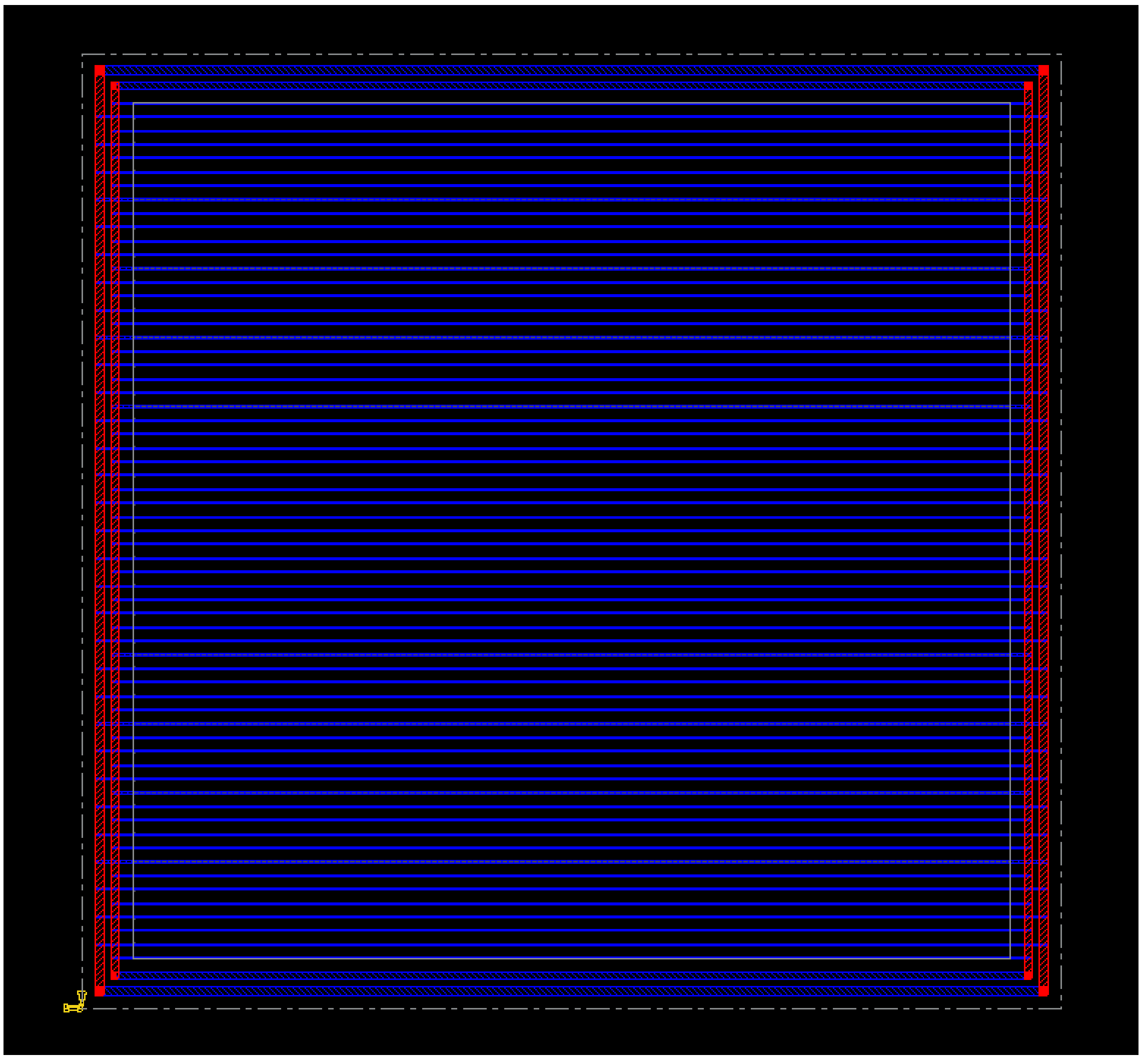

Figure 2: Floorplanning

Now the placement can take place. This process is a delicate one and can take 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 during this procedure. 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.

After this step, filler cells are placed. These cells are very much 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. Mask generation is also easier when continuity is guaranteed. 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 wirelength, the congestion, the coupling etc. are optimized. The final routing must: guarantee the compliance with timing constraints, keep noise low by applying same-layer and adjacent-layer shieldings, keep buses together. Upon all of these operations, design rules must not be violated.

A second optimization is performed following routing: setup and hold times are verified again and possibly adjusted. After this second improvement, 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. No negative slack times have been reported in the generated files. In the following figure is shown the end result of the physical design for this first filter.

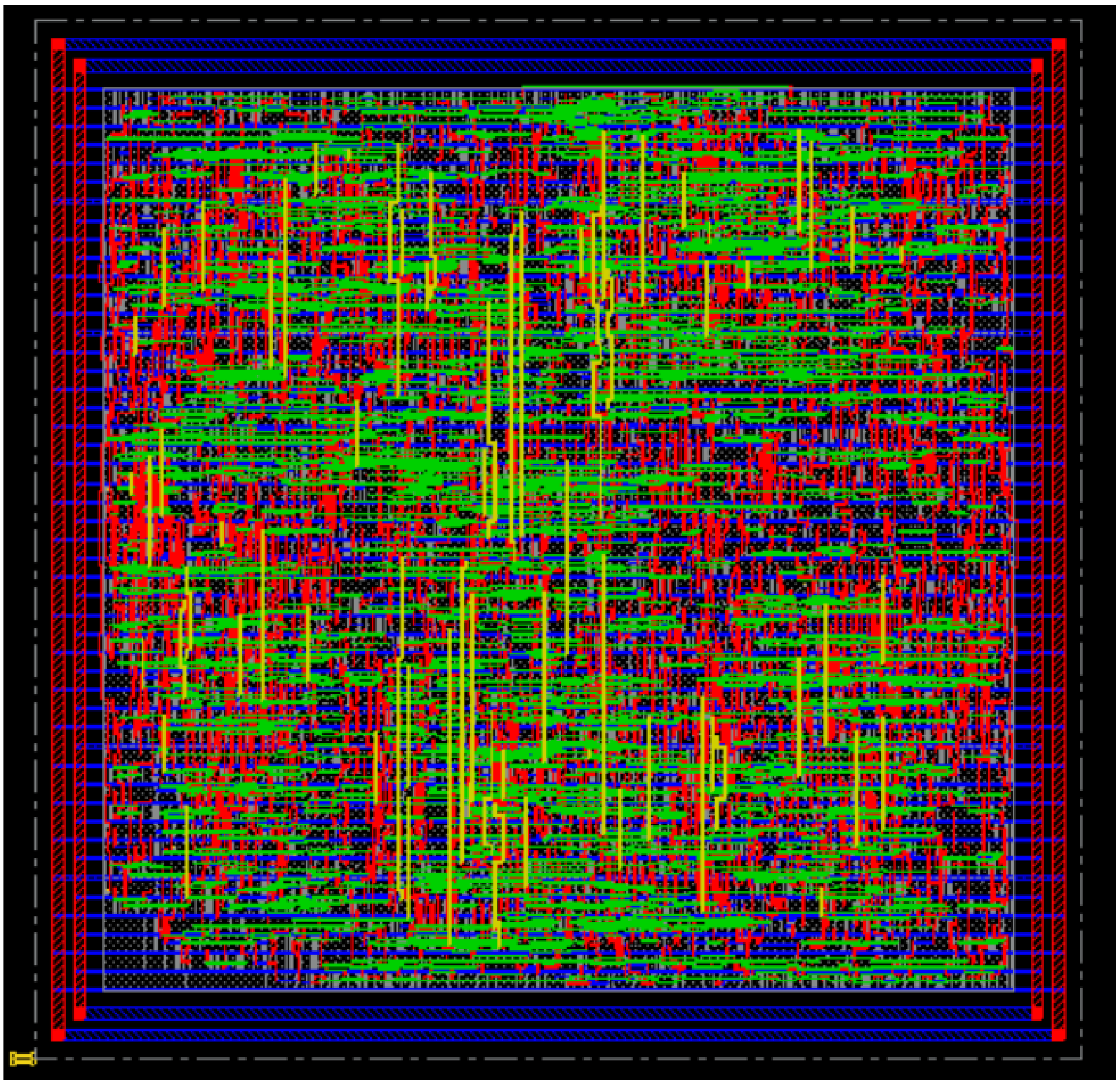

Figure 3: Physical Design of the IIR filter (first version)

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. From *Innovus*, a new verilog netlist and a new SDF file are generated. *Modelsim*, as usual, performs the simulation, annotates the switching activities and generates a VCD file which is read by *Innovus* in order to get the final power report. From the *Innovus log file* the following data have been extracted.

Parameter	Data
Leaf Cells	4052
Nets	1770
Standard Cells	1309
Gates	3393
Total StdCell Length	1.94 mm
Total StdCell Area	2713 μm^2
Total Allocated Area	4558 μm^2
Design Density	59.5 %
Pin Density	30.8 %

Table 1: Some important parameters

Filler Cells Drive Strength	Number of Cells
FILLCELL_X1	1840
FILLCELL_X2	0
FILLCELL_X4	701
FILLCELL_X8	141
FILLCELL_X16	48
FILLCELL_X32	13
Total	2743

Table 2: Number of filler cells needed for each design

Setup mode	all	reg2reg	default
WNS (ns)	14.532	14.532	17.071
TNS (ns)	0.000	0.000	0.000
Violating Paths	0	0	0
All Paths	206	96	191

Hold mode	all	reg2reg	default
WNS (ns)	0.074	0.074	0.000
TNS (ns)	0.000	0.000	0.000
Violating Paths	0	0	0
All Paths	96	96	0

Table 3: WNS: Worst Negative Slack, TNS: Total Negative Slack

	Wire Length
Layer M1	821 μm
Layer M2	6679 μm
Layer M3	5649 μm
Layer M4	613 μm
Layer M5	0 μm
Layer M6	0 μm
Layer M7	0 μm
Layer M8	0 μm
Clock Nets	398 μm
Total	13376 μm

	Number Of Vias
M1	5182
M2	3241
M3	149
Total	8572

Table 4: Clock Nets length was estimated during eGR (early global routing)

0.4 Advanced Architecture Development

The next task was about optimizing the architecture in order to speed up the processing of the data. In order to make the architecture more advanced, the look ahead technique was applied to the IIR so that additional optimization techniques could have been employed. In particular pipelining and the retiming optimizations were performed in order to reduce the critical path.
Recalling figure 4, we can find that it has following characteristics (T_m and T_a being the combinatorial delay of the multiplier and of the adder respectively):

- $T_{CP} = 2T_m + 3T_a$
- $T_{CK} \geq 2T_m + 3T_a$
- Minimum number of registers in the loops: 1
- $T_\infty = \frac{T_m+2T_a}{1} = T_m + 2T_a$

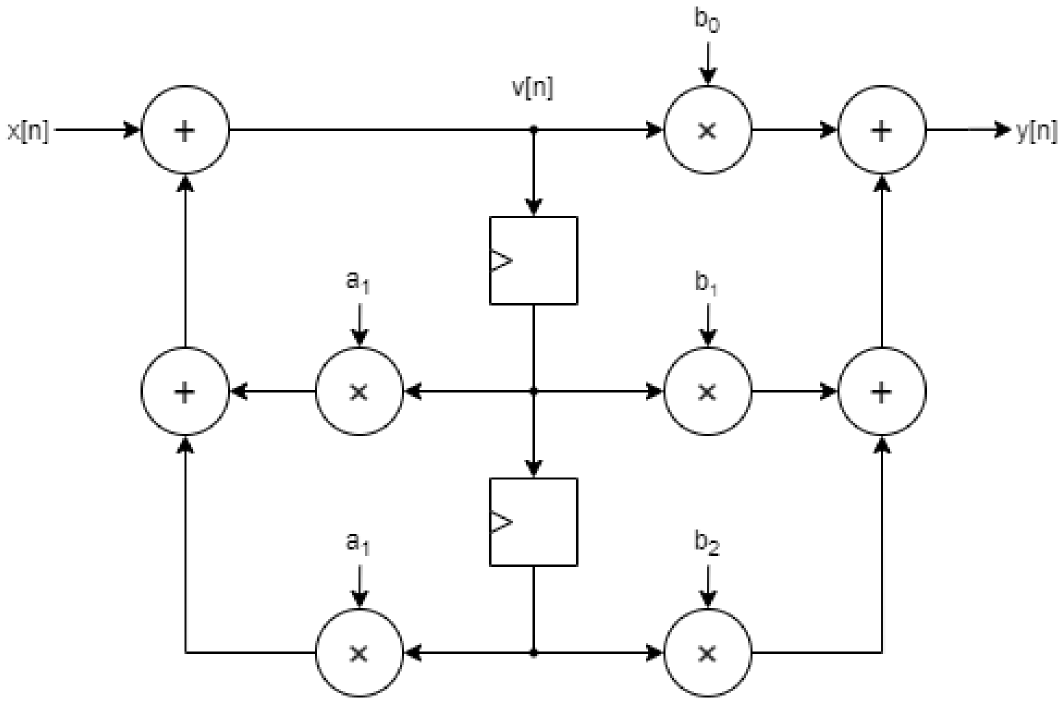

Figure 4: IIR filter in its canonical direct form II

Some pipelining could be added to the feed forward part of the circuit, and the resulting larger combinational path would be reduced to $T_{CP} = T_m + T_a$. As T_{CK} would then be equal to T_{inf}, no universal technique would improve the performance, so Look Ahead is employed. Starting from the theoretical equation we can find the 1-look-ahead form of the given Infinite Impulse filter with a few substitutions and a few transformations. By calling the internal signals $v[n]$, we can write the formal expression of $y[n]$ as function of $v[n]$ in the following form:

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

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

By taking advantage of the z-transform we can rewrite $v[n]$ as $v(z)$ and $y[n]$ as $y(z)$:

$$v(z) = x(z) + a_1 v(z) z^{-1} + a_2 v(z) z^{-2}$$
$$y(z) = b_0 v(z) + b_1 v(z) z^{-1} + b_2 v(z) z^{-2}$$

By grouping $v(z)$ we can find find that it is equal to:

$$v(z) = \frac{x(z)}{1 - a_1 z^{-1} + a_2 z^{-2}}$$

So substituting into $y(z)$ we find that:

$$y(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} + a_2 z^{-2}} x(z)$$

and by redistributing we have demostrated the equivalence of canonical form I and II:

$$y(z) = b_0 x(z) + b_1 x(z) z^{-1} + b_2 x(z) z^{-2} + a_1 y(z) z^{-1} + a_2 y(z) z^{-2}$$

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

To perform the 1-look-ahead transformation we find $y[n-1]$:

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

and we substitute this equation into the $y[n]$ equation:

$$\begin{aligned} y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + a_1 b_0 x[n-1] + a_1 b_1 x[n-2] + a_1 b_2 x[n-3] \\ + a_1^2 y[n-2] + a_1 a_2 y[n-3] + a_2 y[n-2] \end{aligned}$$

$$\begin{aligned} y[n] = b_0 x[n] + (b_1 + a_1 b_0) x[n-1] + (b_2 + a_1 b_1) x[n-2] + a_1 b_2 x[n-3] \\ + (a_1^2 + a_2) y[n-2] + a_1 a_2 y[n-3] \end{aligned}$$

For simplicity, we define the constants:

- $f_0 = b_0$
- $f_1 = b_1 + a_1 b_0$
- $f_2 = b_2 + a_1 b_1$
- $f_3 = a_1 b_2$
- $e_2 = a_1^2 + a_2$
- $e_3 = a_1 a_2$

So applying the substitution find a simpler version of $y[n]$:

$$y[n] = f_0 x[n] + f_1 x[n-1] + f_2 x[n-2] + f_3 x[n-3] + e_2 y[n-2] + e_3 y[n-3]$$

This formula is the 1-look-ahead version of the IIR in a sort of direct form I. By exploiting once again the z-transform we can transform it into a sort of direct form II. So, using the z-transform we get:

$$y(z) = f_0 x(z) + f_1 x(z) z^{-1} + f_2 x(z) z^{-2} + f_3 x(z) z^{-3} + e_2 y(z) z^{-2} + e_3 y(z) z^{-3}$$

by grouping $y(z)$ we get:

$$y(z) = \frac{f_0 + f_1 z^{-1} + f_2 z^{-2} + f_3 z^{-3}}{1 - e_2 z^{-2} + e_3 z^{-3}} x(z)$$

We can now define $v'(z)$ as:

$$v'(z) = \frac{x(z)}{1 - e_2 z^{-2} + e_3 z^{-3}}$$

and we can then redistribute the factors:

$$v'(z) = x(z) + e_2 v'(z) z^{-2} + e_3 v'(z) z^{-3}$$

and also find $y(z)$ as a function of $v'(z)$:

$$y(z) = f_0 v'(z) + f_1 v'(z) z^{-1} + f_2 v'(z) z^{-2} + f_3 v'(z) z^{-3}$$

Finally, we can get back into the discrete time domain to find $y[n]$ as function of $x[n]$ and $v'[n]$ itself:

$$v'[n] = x[n] + e_2 v'[n-2] + e_3 v'[n-3]$$

$$y[n] = f_0 v[n-1] + f_1 v[n-1] + f_2 v[n-2] + f_3 v[n-3]$$

This correspond to a sort of direct form II of the 1-look-ahead given IIR filter and it is represented in figure 5:

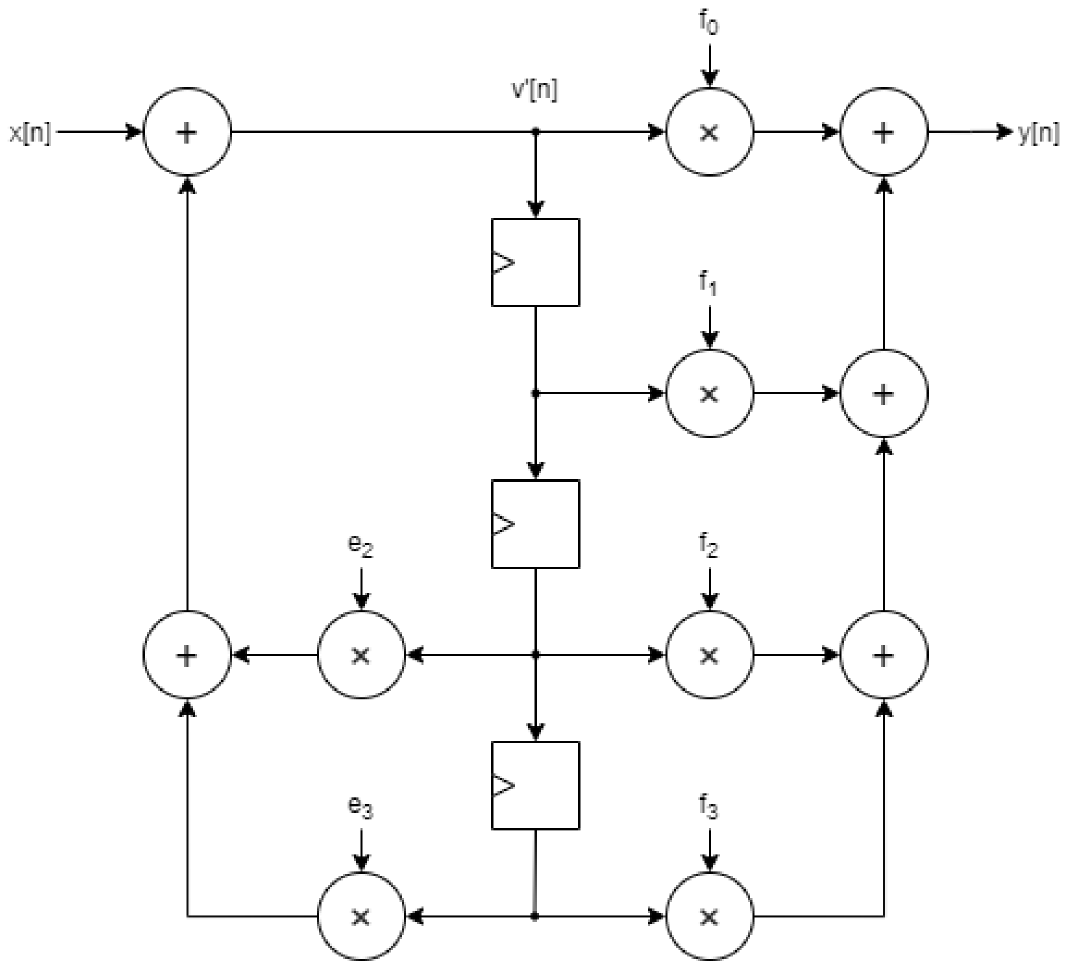

Figure 5: IIR filter with 1-Look-Ahead transform in "direct form II"

We proceeded to test if our calculations were correct by modifying the C program in order to simulate the new Look Ahead architecture. First we calculated the new coefficients by using the formulas reported before and the original coefficients, but performing the simulation we discovered that the Total Harmonic Distortion was higher than our target of -30dB. In order to keep the THD lower than the target we had to increase the internal parallelism n_b by one bit to 11. The new coefficients had to be calculated first by shifting left the original coefficients and then by performing the substitutions reported before. The newly calculated coefficients are:

- $f_0 = 52$
- $f_1 = 113$
- $f_2 = 71$
- $f_3 = 9$
- $e_2 = -32$
- $e_3 = -10$

These changes allowed us the get a THD of $-33.9501dB$, which is well within our target. This new architecture shown in figure 5 has the following characteristics:

- $T_{CP} = 2T_m + 3T_a$
- $T_{CK} \geq 2T_m + 3T_a$
- Minimum number of registers in the loops: 2
- $T_\infty = \frac{T_m+2T_a}{2}$

Like before the look ahead transformation, it is still possible to add pipeline stages to the feed forward side of the filter. The identified cutsets are marked in figure 6:

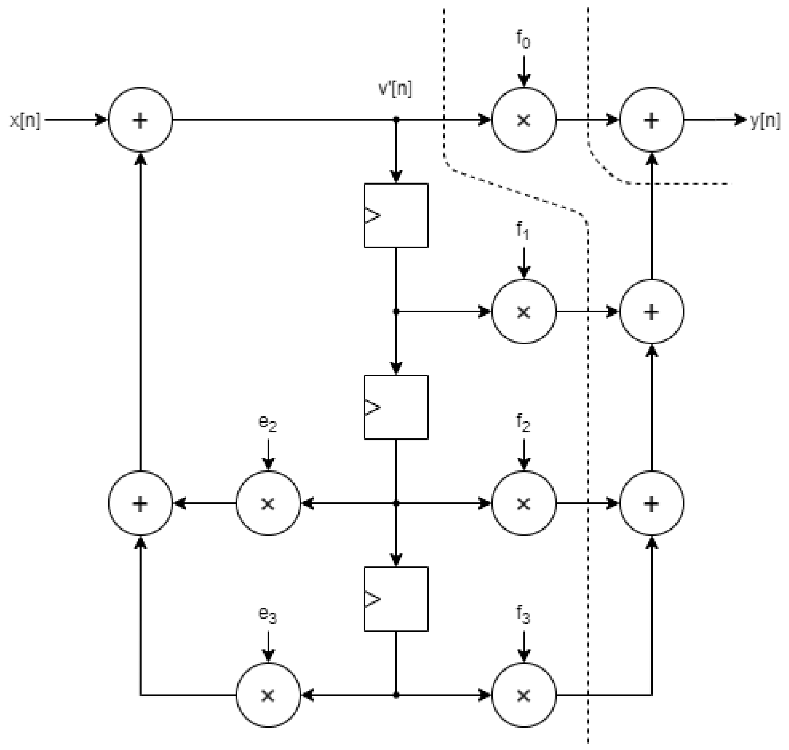

Figure 6: Possible cutsets for pipelining the IIR filter with 1-Look-Ahead transform in "direct form II"

Adding the registers, the new pipelined architecture is following in figure 7:

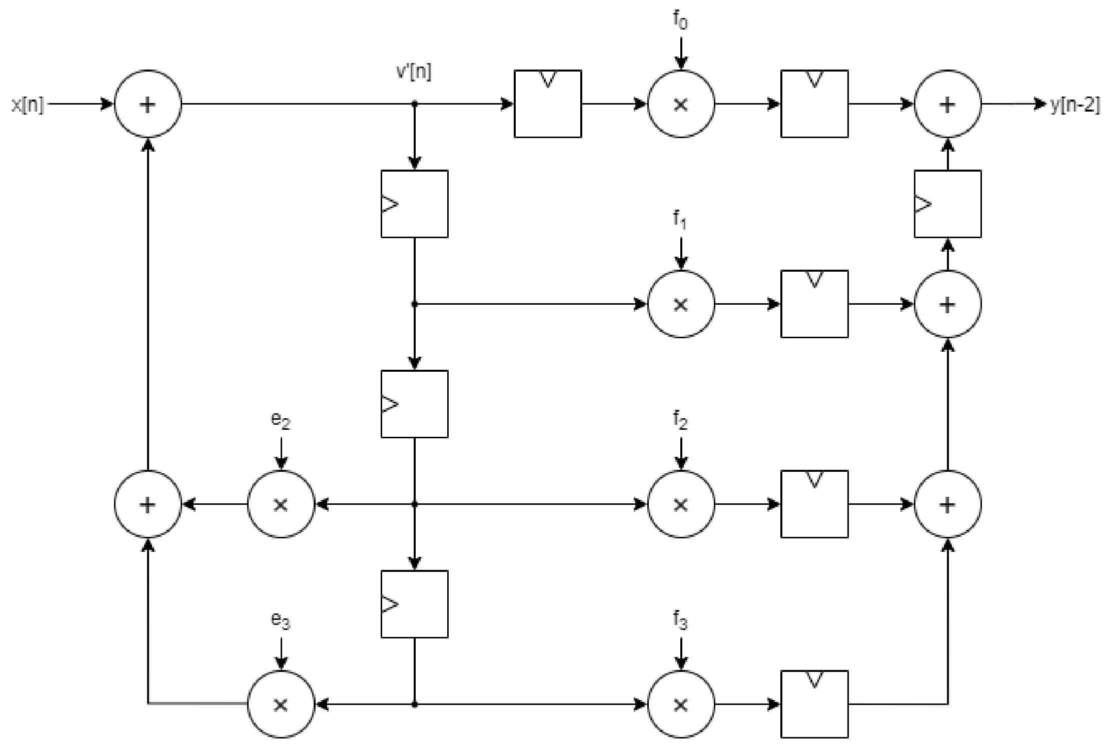

Figure 7: Pipelined IIR filter with 1-Look-Ahead transform in "direct form II"

The pipelined architecture now has the following characteristics:

- $T_{CP} = T_m + 2T_a$
- $T_{CK} \geq T_m + 2T_a$
- Minimum number of registers in the loops: 2
- $T_\infty = \frac{T_m+2T_a}{2}$

As T_∞ is smaller than the minimum T_{CK}, more universal optimizations are possible. As a matter of fact we can retime the Feed-back portion of the filter, as shown in the following figures. The red registers, highlighted by the red arrows, show the new registers that substitute the register where the arrows originate from.

Figures 8 and 9 show the retiming of the bottom v register, this is important as it cuts a critical path from $T_m + 2T_a$ to $max\{T_m, 2T_a\}$:

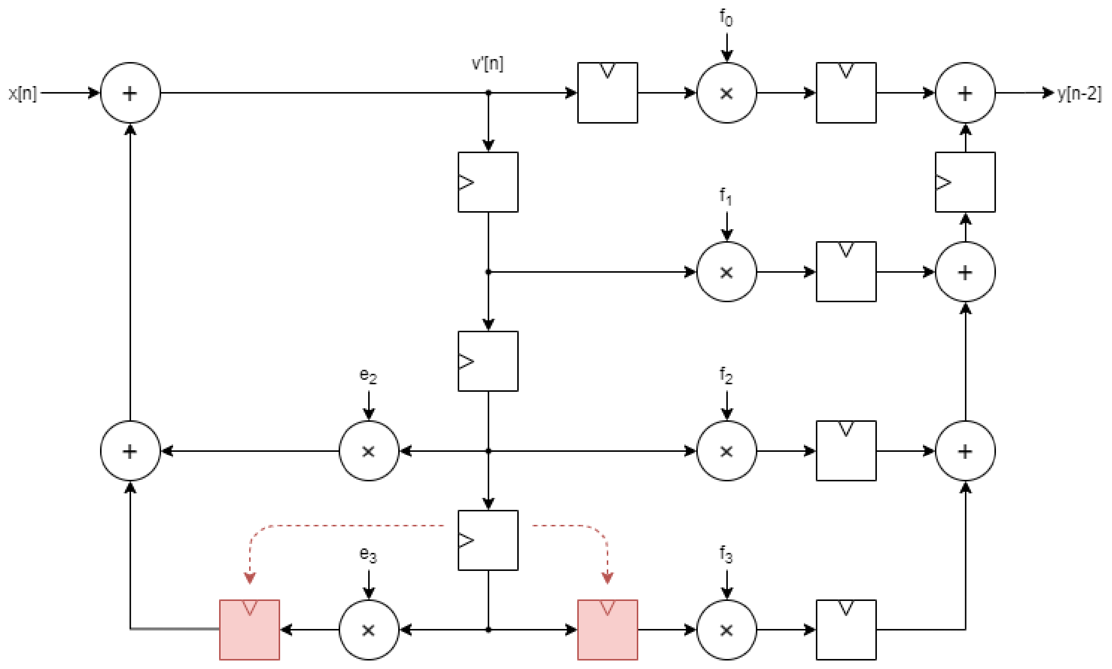

Figure 8: First step of the retiming of the Pipelined IIR filter with 1-Look-Ahead transform in "direct form II"

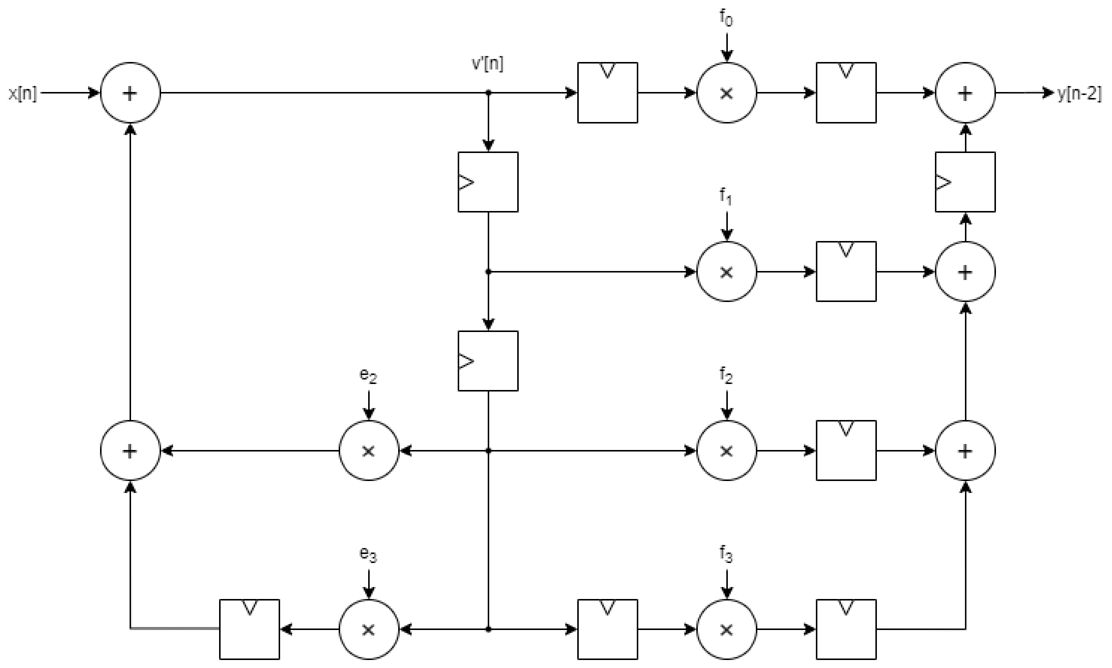

Figure 9: Second step of the retiming of the Pipelined IIR filter with 1-Look-Ahead transform in "direct form II"

Figures 10 and 11 show the retiming of the middle v register, this is important as it cuts ance again another critical path from $T_m + 2T_a$ to $maxT_m, 2T_a$:

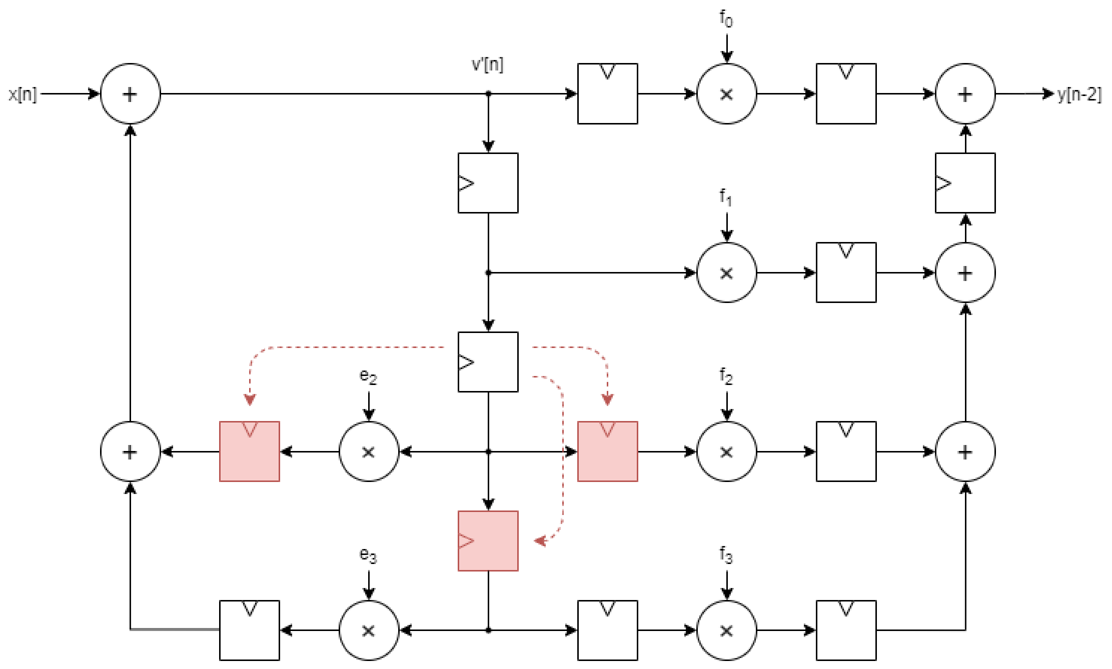

Figure 10: Third step of the retiming of the Pipelined IIR filter with 1-Look-Ahead transform in "direct form II"

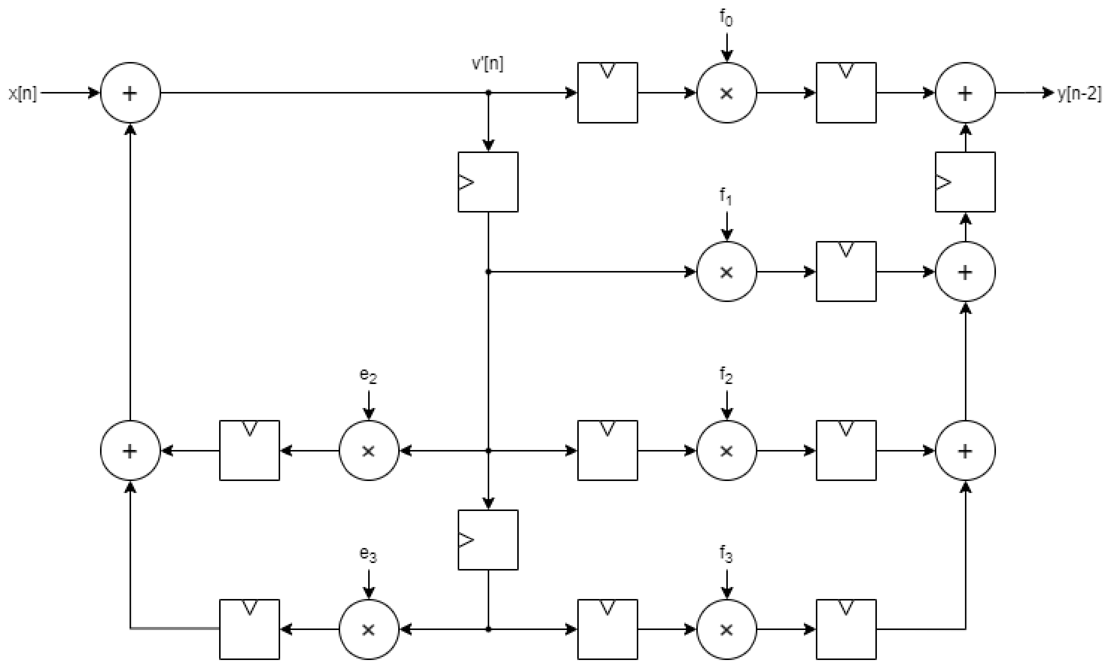

Figure 11: Fourth step of the retiming of the Pipelined IIR filter with 1-Look-Ahead transform in "direct form II"

Figures 12 and 13 finally show a "dummy" retiming of the two highlighted registers in order to use one less register as it is not needed.

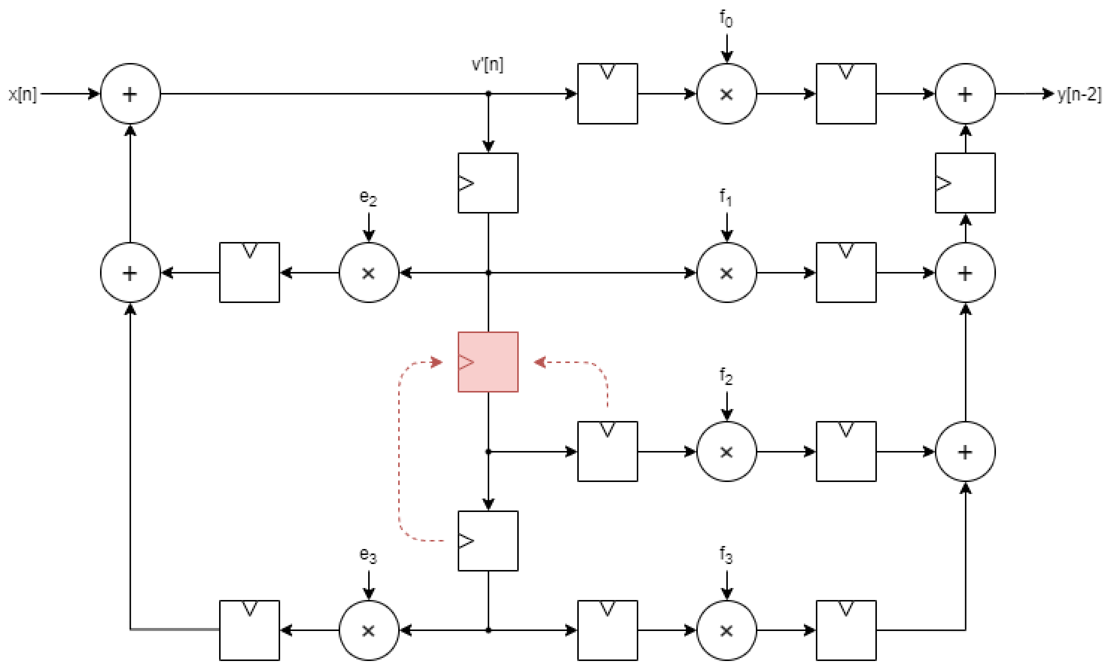

Figure 12: Fifth step of the retiming of the Pipelined IIR filter with 1-Look-Ahead transform in "direct form II"

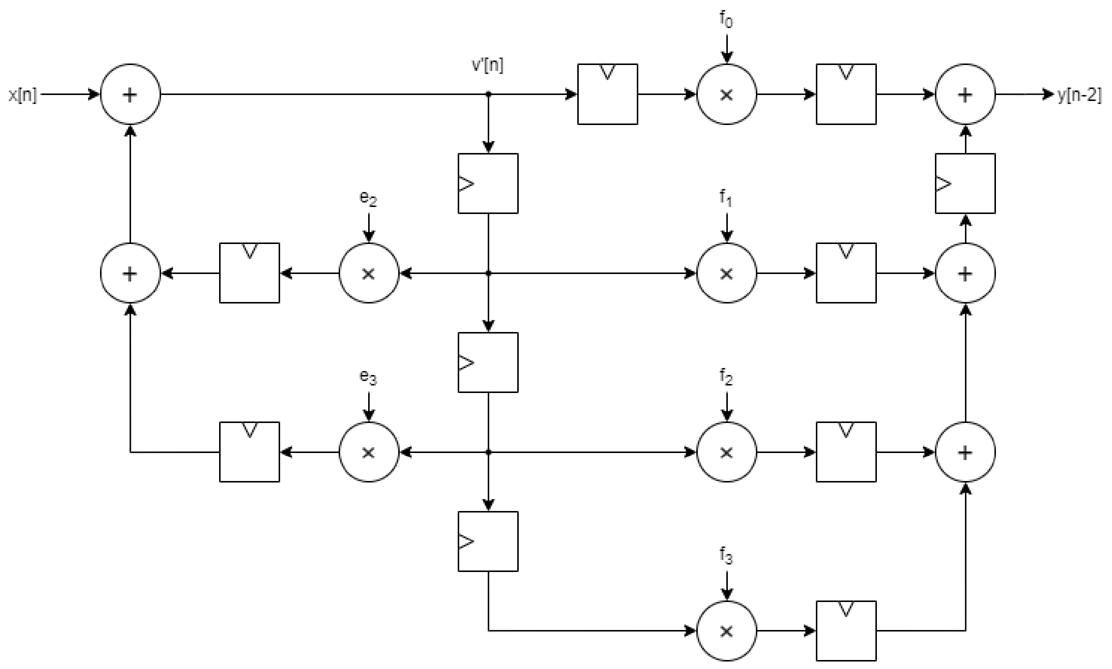

Figure 13: Pipelined and retimed IIR filter with 1-Look-Ahead transform in "direct form II"

The newest and final architecture has the following characteristics:

- $T_{CP} = max\{T_m, 2T_a\}$
- $T_{CK} \geq max\{T_m, 2T_a\}$
- Minimum number of registers in the loops: 2
- $T_{\infty} = \frac{T_m + 2T_a}{2}$

As we can't perform a fine-grain pipelining or retiming, this is the best performing architecture for this kind of filter, making the obvious assumption that $T_m > T_a$.
Following the design phase, two Look-ahead architectures were implemented, the first was the one represented in figure 5, which was the basic architecture obtained after the look ahead technique was implemented. This version was only used for simulations in order to verify that the transformation was implemented correctly. Then the optimized pipelined and retimed architecture was implemented in VHDL. For the prior version and for the non look-ahead version the VOUT flag, the flag that indicates that a new data is present at the output, was very simple to be provided as it was just the VIN signal delayed by two clock cycles by employing two flip flops. The sampled VIN signal was also used as an enable for the internal and output registers. For the most advanced version it is more complex as there are two pipeline stages which require different timings and require more delays for the VOUT signal. The implemented enable circuitry and the VOUT signal logic are shown in figure 14. The new flip flops used for implementing the correct timings are shown at the bottom, while the data registers are marked with the enable signal they ar connected to.

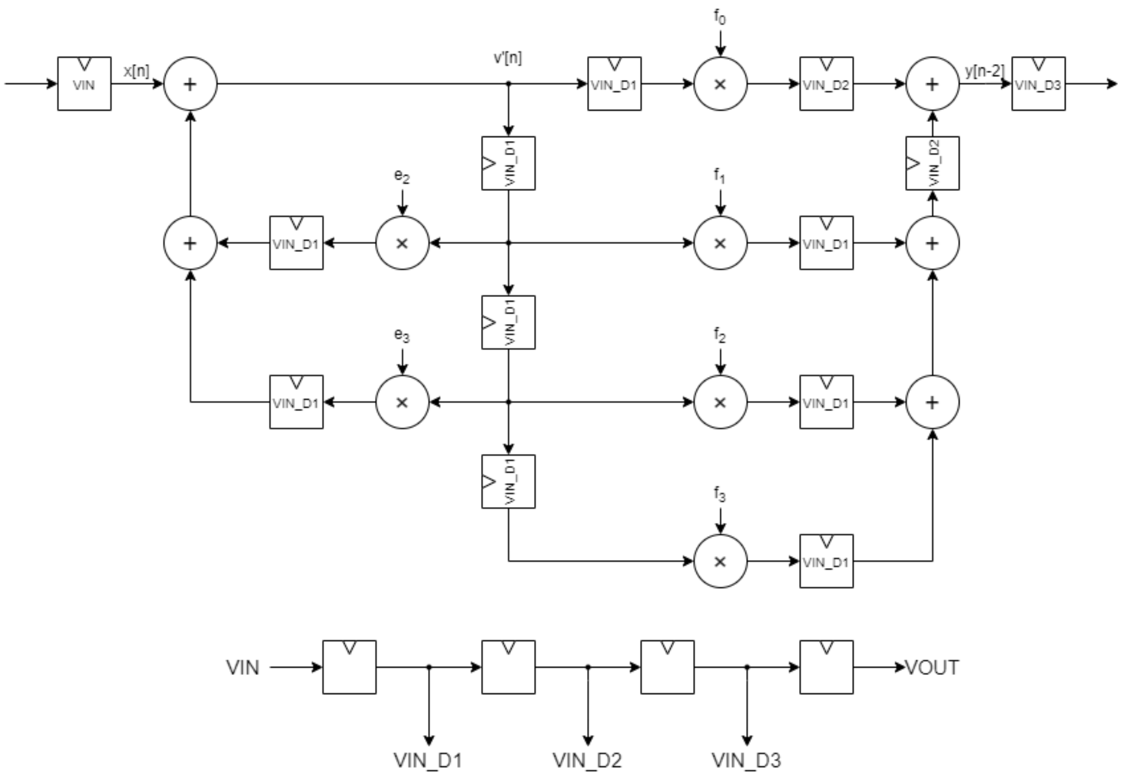

Figure 14: Enable circuitry of the pipelined and retimed IIR filter with 1-Look-Ahead transform in "direct form II"

The process following this step is the same followed for the baseline architecture: the design was first verified with *Modelsim* taking advantage of the test bench which was slightly modified in order

to provide the new coefficients, that need an higher precision in order to satisfy THD requirement. As this test was passed, the design was first synthesized with the *Synopsys Design Vision* tool and then placed and routed with the *Cadence Innovus* tool.
In the following figure the physical design for the IIR filter, LookAhead version, is reported.

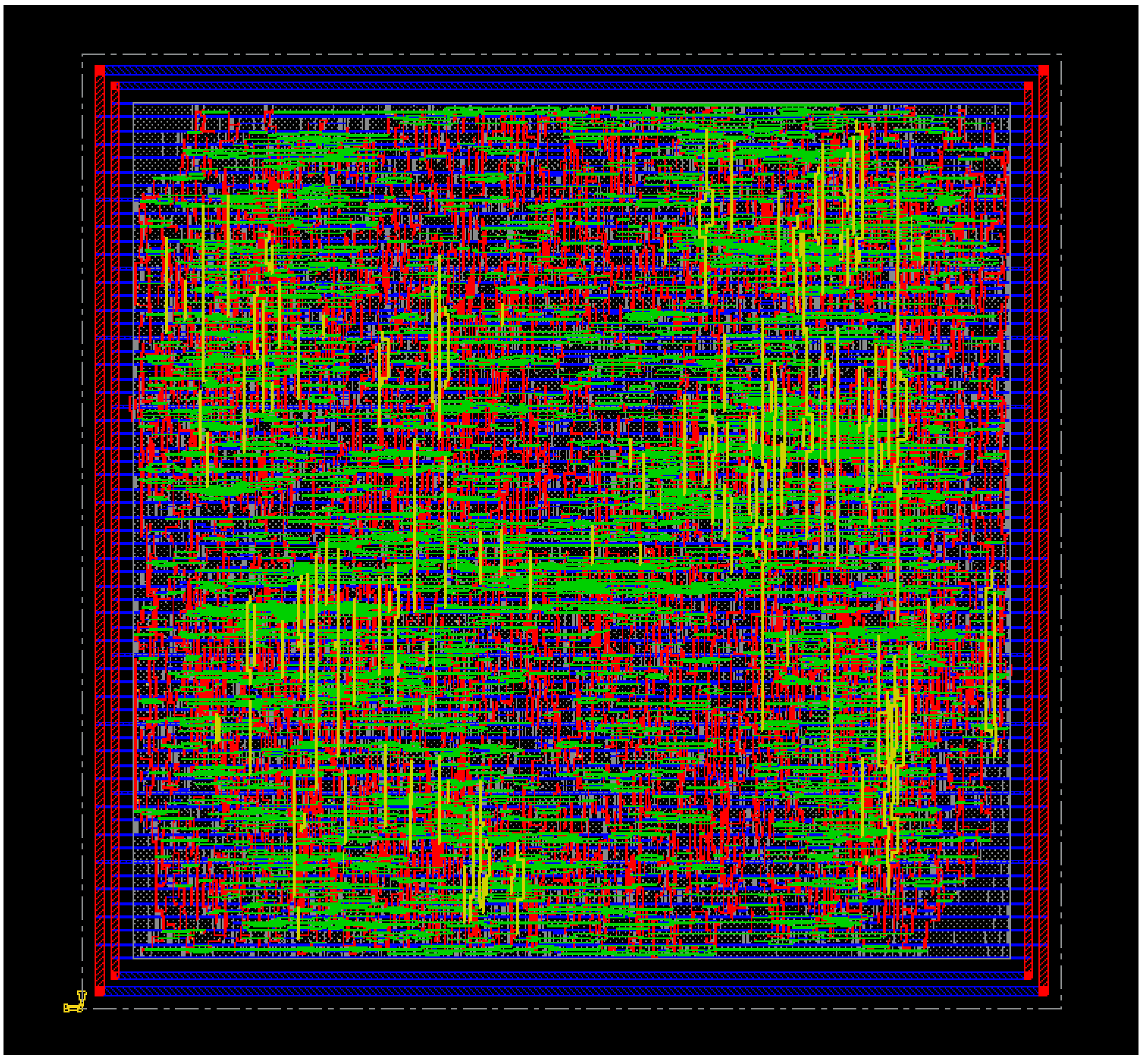

Figure 15: Physical Design of the IIR filter (LA version)

From the *Innovus log file* the following data have been extracted.

Parameter	Data
Leaf Cells	6597
Nets	2886
Standard Cells	2147
Gates	5723
Total StdCell Length	3.26 mm
Total StdCell Area	4558 μm^2
Total Allocated Area	7735 μm^2
Design Density	59.05 %
Pin Density	29.5 %

Table 5: Some important parameters

Filler Cells Drive Strength	Number of Cells
FILLCELL_X1	2925
FILLCELL_X2	0
FILLCELL_X4	1064
FILLCELL_X8	363
FILLCELL_X16	82
FILLCELL_X32	16
Total	4450

Table 6: Number of filler cells needed for each design

Setup mode	all	reg2reg	default
WNS (ns)	0.074	0.074	0.000
TNS (ns)	0.000	0.000	0.000
Violating Paths	0	0	0
All Paths	203	203	0

Hold mode	all	reg2reg	default
WNS (ns)	9.742	9.742	11.070
TNS (ns)	0.000	0.000	0.000
Violating Paths	0	0	0
All Paths	423	203	296

Table 7: WNS: Worst Negative Slack, TNS: Total Negative Slack

	Wire Length
Layer M1	1391 μm
Layer M2	10972 μm
Layer M3	9322 μm
Layer M4	1159 μm
Layer M5	0 μm
Layer M6	0 μm
Layer M7	0 μm
Layer M8	0 μm
Clock Nets	783 μm
Total	21427 μm

	Number Of Vias
M1	8395
M2	5260
M3	313
Total	13968

Table 8: Clock Nets length was estimated during eGR (early global routing)

0.5 Obtained Results

In this last part, we will discuss the obtained results of delay, area and power for both the basic and the advanced architectures of the filter. We will analyze the differences between the two structures and also the changes in the reports obtained before and after the place and route. To guarantee a correct comparison, all the reports have been acquired using the same procedure, working with a clock frequency equal to $4 * f_{max}$ and starting from the same input values during the simulations. We have also to underline that before extracting the needed results we checked the correct behavior of what we have implemented.
We can now start comparing the values of delay, area and power obtained for the basic version of the filter after the synthesis and after the place and route to find the changes due to the physical design:

Basic Structure	After Synthesis	After Place And Route
Critical Path's Delay	$4.23ns$	$3.467ns$
Area Occupation	$2732.4\mu m^2$	$2707.9\mu m^2$
Total Power Consumption	$250.075\mu W$	$771.06\mu W$

From the table we can notice that the critical path's delay and the total occupied area decreased by a little after the place and route. This could seem unusual because the addition of interconnections' parameters should increase both the delays and the area. We have, however, to consider that during the place and route process we have applied some optimizations to our design, which in turn gives us a final result that is a little better than the previous one. If we instead consider the obtained power consumption, we can see that after the place and route, it has increased by a lot and becomes about three times the one after the synthesis. This is due, primarily, to the fact that in the place and route process, as interconnections are added, the dynamic power contribution increases dramatically. This is because of the presence of capacitances on the lines which have a big role in dynamic power. We have to consider, also, that the the optimizations mentioned before, while letting us improve the efficiency of our circuit, they have a little or a null effect on the reduction of power, which instead might sometimes be subject to an increase. In the following graphs we can see more in details the sources of the total power consumption for both cases. We can notice how the biggest differences are in the dynamic contributions (internal and switching power):

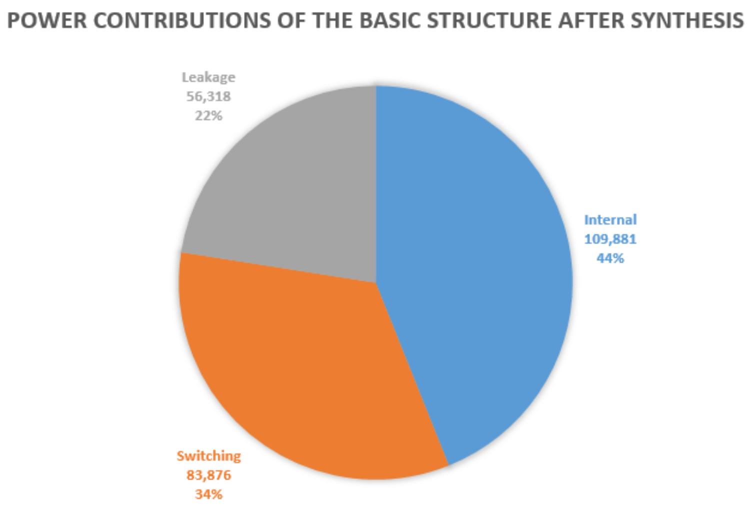

Figure 16: Power consumption contributions for the basic structure after synthesis

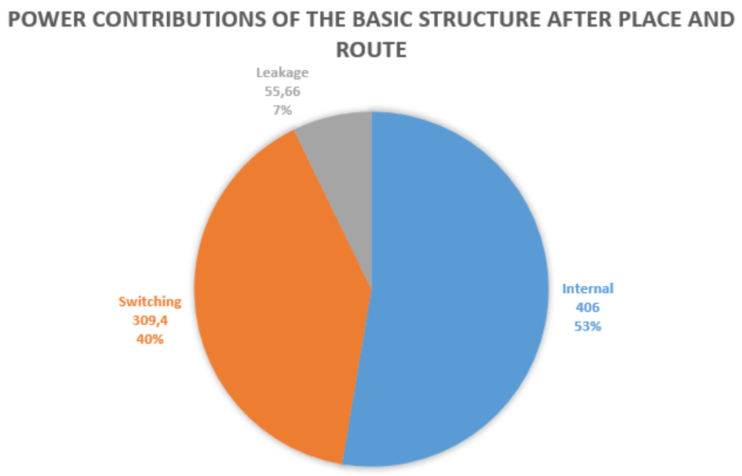

Figure 17: Power consumption contributions for the basic structure after place and route

The same comparison can be done for the advanced architecture of the filter where the Look Ahead, retiming and pipelining techniques are applied:

Advanced Structure	After Synthesis	After Place And Route
Critical Path's Delay	$2.62ns$	$2.258ns$
Area Occupation	$4639.3\mu m^2$	$4567.0\mu m^2$
Total Power Consumption	$633.841\mu W$	$1356.69\mu W$

The obtained results have a similar behavior of what we have already seen for the basic structure. In particular, we notice a decrease in both critical path delay and area and an important increase in total power consumption. We can also see that if we compare these results with the ones obtained for the basic version we have a significant reduction in the delay and an increase in both area occupation and power consumption. We will analyze this better in the following.
As we have already done before, we can now see the power distribution for the two considered cases, the behavior is again similar to what happens in the basic structure:

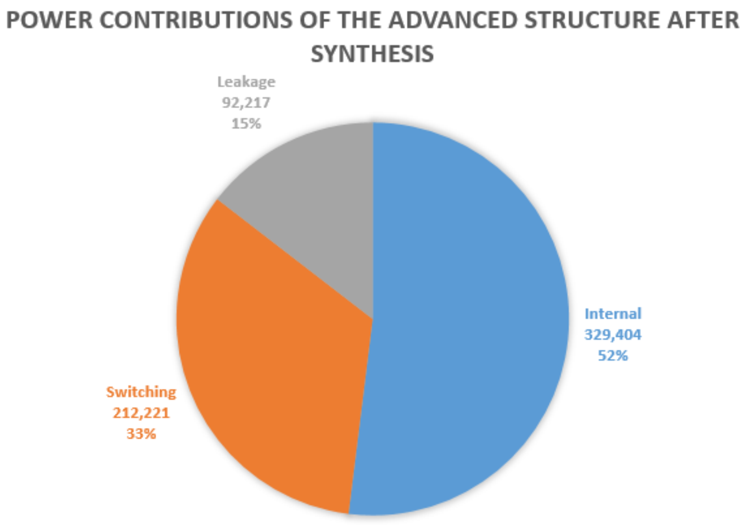

Figure 18: Power consumption contributions for the advanced structure after synthesis

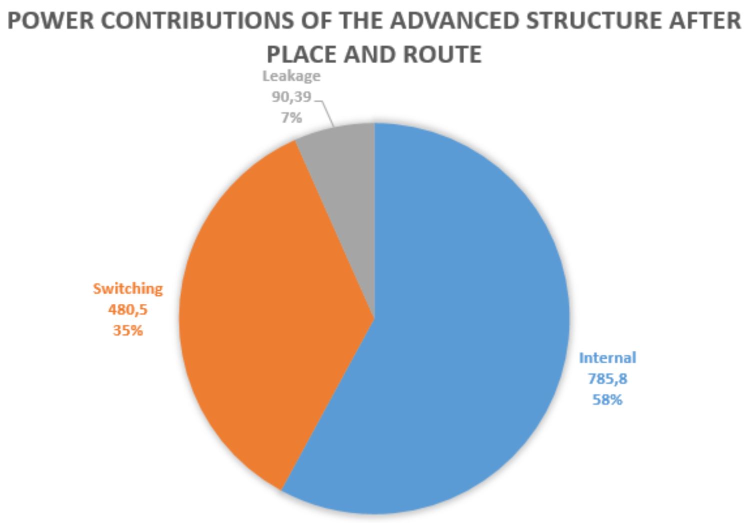

Figure 19: Power consumption contributions for the advanced structure after place and route

0.5.1 Basic and Advanced Structure Comparison

We can now compare the two different versions to see whether the advanced one has the desired improvements. We start by considering the results obtained after the synthesis but before the place and route process:

After Synthesis	Basic Structure	Advanced Structure
Critical Path's Delay	$4.23ns$	$2.62ns$
Area Occupation	$2732.4\mu m^2$	$4639.3\mu m^2$
Total Power Consumption	$250.075\mu W$	$633.841\mu W$

From this table we can notice that, as we have already introduced before, the advanced version of our filter has a reduction in critical path delay but an increase in both the area occupation and the power consumption. Using the advanced structure we can obtain a good reduction in the delays and so in the minimum clock period, but this brings a major increase in both the area and the power which are more than doubled. This increase in area and consumption is obviously due to the presence of a higher number of operators and registers in the advanced version. These added elements are, however, necessary to obtain the wanted improvements. We can see that a trade-off between efficiency and costs is present. In particular, to obtain a more efficient filter the costs (in terms of both area and power) have to increase, while, if we want a cheaper circuit, its performances will be limited.
We can finally analyze also the results obtained in the two different structures after the place and route process:

After Place And Route	Basic Structure	Advanced Structure
Critical Path's Delay	$3.467ns$	$2.258ns$
Area Occupation	$2707.9\mu m^2$	$4567.0\mu m^2$
Total Power Consumption	$771.06\mu W$	$1356.69\mu W$

Also in this case the results are in line with the ones we have discussed before. We must notice that, after the place and route process, the differences between the basic and the advanced structures are always important but not as much as before. We now have, for example, that in the advanced structure both the area and the power consumption are less than the double of the ones in the basic structure, while before they more than doubled. This can be due to the optimizations applied during the place and route process.