



Politecnico di Torino
III Facoltà di Ingegneria

Laboratory 2

Digital arithmetic

Master degree in Electrical Engineering

Authors: Group 21

Dilillo Nicola S284963
Moncalvo Stefano S290315
Carrano Lorenzo S281565

November 21, 2021

Contents

1	Reference model development	1
1.1	Introduction	1
1.2	C prototype	3
1.2.1	Evaluate the THD	3
2	VLSI implementation	5
2.1	Starting architecture development	5
2.2	Simulation	6
2.3	Logic synthesis	6
2.4	Place & Route	7
3	Advanced architecture development	9
3.1	Unfolding	9
3.2	Pipeline	9
3.3	Simulation	11
3.4	Logic synthesis	11
3.5	Place & Route	12

CHAPTER 1

Reference model development

1.1 Introduction

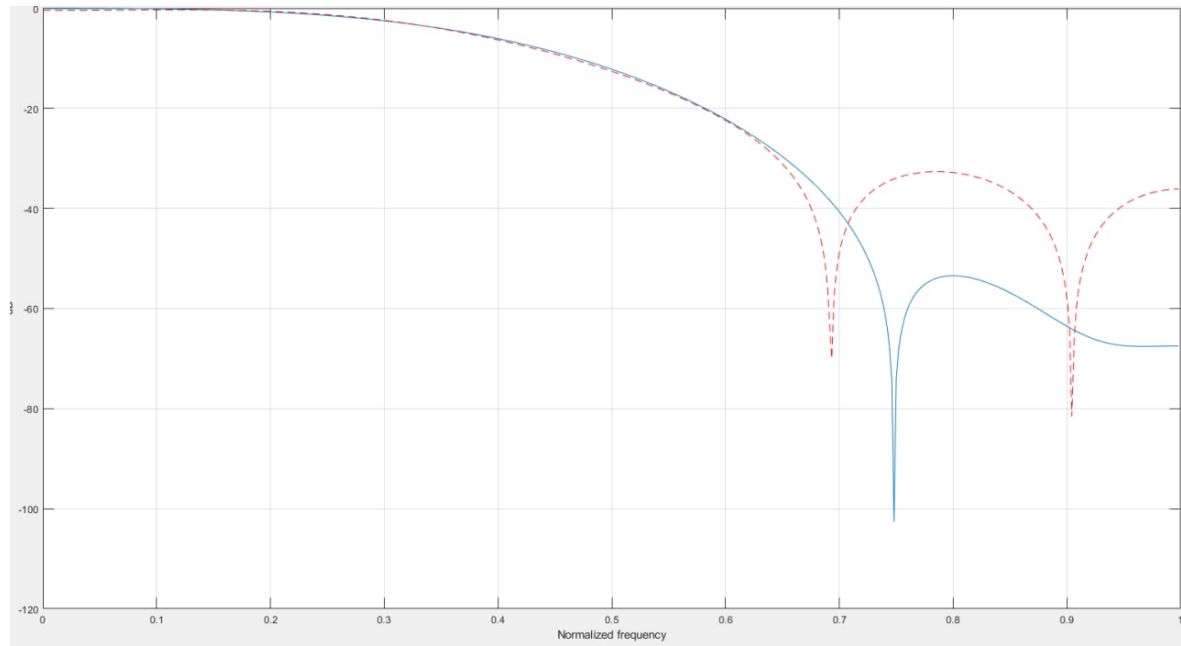
The goal of this laboratory is to design a Finite Impulse Filter filter (FIR) with a cut frequency of 2 kHz, and then applying some optimization techniques such as unfolding and pipelining to the basic structure. Filter was designed according two parameter: order and number of bits. The order employed for the following filter is 10 and the number of bits is 9. A prototype version of the filter has been developed in C language and Matlab in order to be able to compare their results with the ones coming from the simulation of the HDL design. In the following table the main filter specifications are summarized:

Filter Specification	Value
Filter Type	FIR filter
Cut-off frequency	2kHz
Sampling frequency	10kHz
Filter order	10
Number of bits	9

First step is the generation of coefficients. To do this Matlab function `fir1` has been used. The coefficients are shown in the following table:

Number	Integers	Normalized
0	-1	-0.0078125
1	-2	-0.015625
2	-4	-0.03125
3	8	0.0625
4	35	0.2734375
5	50	0.390625
6	35	0.2734375
7	8	0.0625
8	-4	-0.03125
9	-2	-0.015625
10	-1	-0.0078125

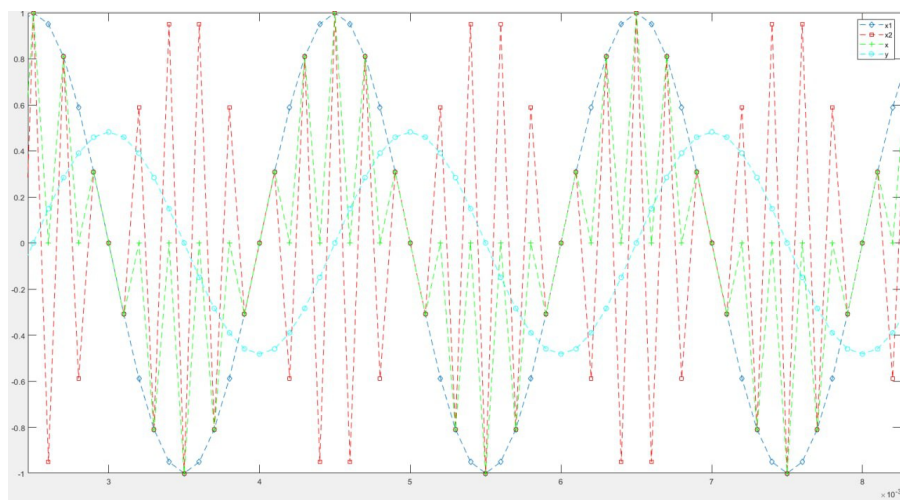
Those coefficients are subjected to a quantization operation over Nb bits and expressed both in integer and real form. It is possible to estimate the effect of this quantization by comparing the designed transfer function to the quantized one, as shown in the following figure:



At this point, another Matlab script is executed in order to perform different simulations with prototype filter with a cut-off frequency of 2 kHz and a sampling frequency of 10 kHz, taking as input signal the average value between two sinusoidal waves of frequency of 500 HZ and 4.5 kHz respectively. After this execution two files have been generated:

1. *sample.txt*, which contains the sample values that have fed the input of the FIR;
2. *result.txt*, which contains the output values that has been elaborated from the FIR.

The following figure shows the two sinusoidal waveforms and the effective filter's input:



1.2 C prototype

The C-language script simulates a FIR filter implementing the following relation:

$$[!h]y_i = \sum_{n=0}^{10} x_{i-n} \cdot b_n$$

Inside the script, a function is defined in order to evaluate the output $y[n]$ at a specific time instant, knowing the input sample $x[n]$.

FIR constants are hardcoded inside the script, declared as a constant array of integers, while an internal buffer is needed to store the previous input values and shift them for each function call, and the output is obtained by summing coefficient-input products in a loop.

In order to emulate the finite internal parallelism of the hardware architecture, a shift operation is performed on the result before storing it into the accumulator. Of course, this truncation operation introduces an error in the evaluation of the output samples. The maximum accuracy would be obtained with a parallelism that is equal to the double of the bits number used for the architecture.

Thanks to this script is possible to compare the performance of the fixed-point version with respect to the Matlab computation results.

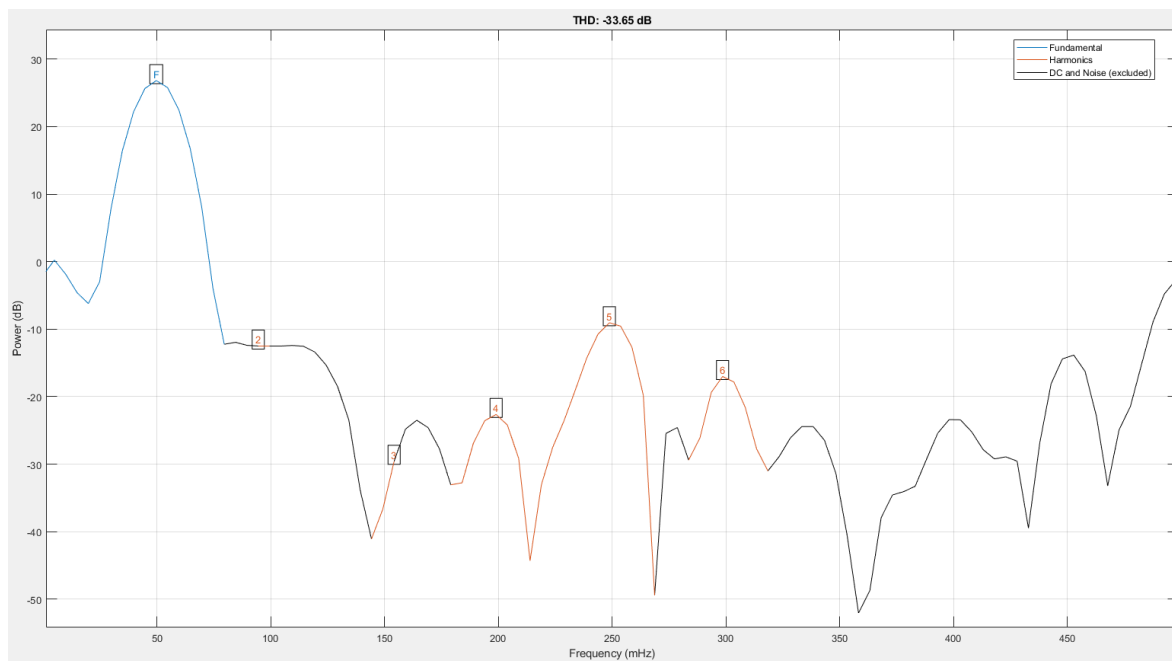
1.2.1 Evaluate the THD

The purpose of this step is to evaluate the Total Harmonic Distortion (THD), trying to obtain a maximum value of -30dB.

If THD exceeds the maximum tolerated value, it is necessary to increase the bit numbers in order to reduce its amount, while if there is a gap between the maximum tolerated value and the obtained one, it is possible to reduce bit numbers and thus the complexity of the FIR implementation.

With 9 bits used for data, obtained THD is -39.07 dB. Reducing the number of bits to 8 the obtained value of THD is -33.65 dB, thus still acceptable. Applying a further bit-number reduction, thus with a 7-bit parallelism, the value of THD exceeds the maximum allowed amount, reaching the value of -27.01 dB.

At the end of this analysis, it has been decided to use 8 bits for the final implementation of the FIR, in order to reduce the area while still accomplish the requested THD amount, shown in the following figure:



CHAPTER 2

VLSI implementation

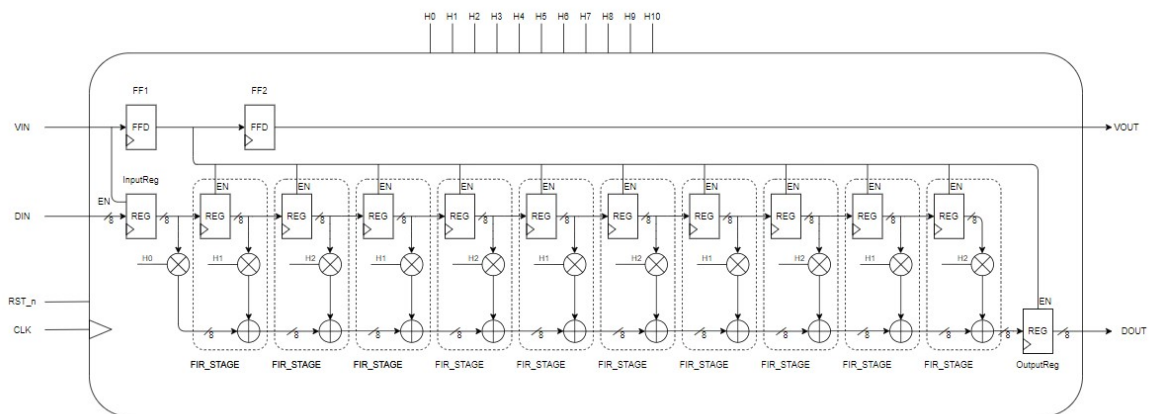
2.1 Starting architecture development

The purpose of this section is to develop in VHDL the architecture of the previously designed filter. The architecture of the filter is composed by four elements:

- Adders
- Multipliers
- Flipflops
- Registers

The 8-bit input is received and then propagated through a chain of 10 registers; the output of each register is multiplied by the corresponding coefficient, and the results are summed together to form the filter's output. All registers use VIN as an enable signal, in order to avoid unwanted propagation of data.

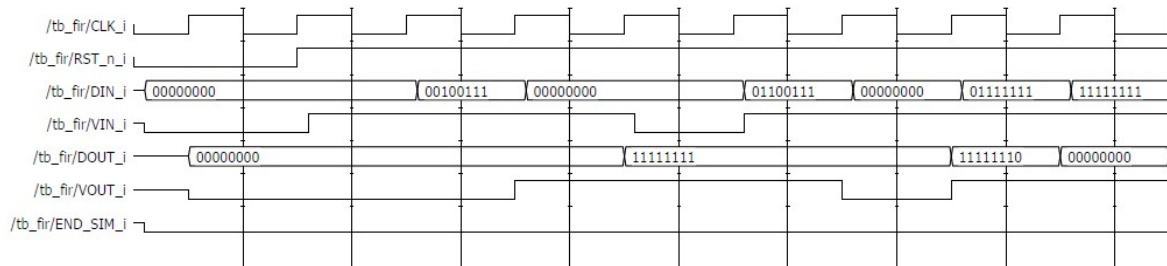
The VIN signal, delayed of two clock cycles, is also used to drive VOUT. Every input and output signal is loaded or produced by registers or flipflops, to reduce the risk of interference from external signals.



2.2 Simulation

The design was simulated using a testbench written in both Verilog and VHDL. The testbench is composed of four distinct entities:

- **clk_gen:** generates a clock signal of the specified frequency, and a reset signal.
- **data_maker:** reads the samples.txt file and provides an input every clock cycle and its validity using the VIN signal.
- **data_sink:** recives the outputs of the filter every clock cycle and writes them in the output.txt file if VOUT is equal to 1.
- **tb_fir:** is the testbench top entity written in Verilog.



The image above shows an extract of the Modelsim simulation; the waveforms show the moment where VIN goes to zero, and after two clock cycles VOUT is set to zero in order to avoid errors in the output signal.

At the end of the simulation the values stored in Output.txt were compared with the ones produced by the C prototype. The two files are equal, which means that the filter is behaving correctly.

2.3 Logic synthesis

After the simulation the design must be synthesized.

To estimate the maximum working clock frequency of the filter, the clock period in the design compiler is set to 0 ns. In this way the compiler optimizes the circuit as much as possible, and the negative slack of the timing report corresponds to the maximum clock frequency.

After running the synthesis at the computed frequency the area is evaluated.

Max Clock Frequency	Min Clock Period	Area
303 MHz	3.3 ns	3765.23 μm^2

It is requested to set the frequency to 25% of the maximum value.

$$\frac{1}{4}f_{MAX} = 75.75\text{MHz}$$

After the synthesis a new area estimation is produced.

$$\text{Area} = 3682 \mu\text{m}^2$$

The constraints on the clock have a significant role in the estimation of the area: by allowing the frequency to be lower, the size of the circuit will be smaller.

The Design Compiler produces the Verilog netlist of the synthesized circuit and a .sdf file containing the circuit's delays. Those files are used by Modelsim to simulate the netlist with the correct timing parameters and obtain the switching activity of the nodes, which is saved in a .vcd file.

The results of this simulation have been checked to assure that they are coherent with the ones of the original circuit.

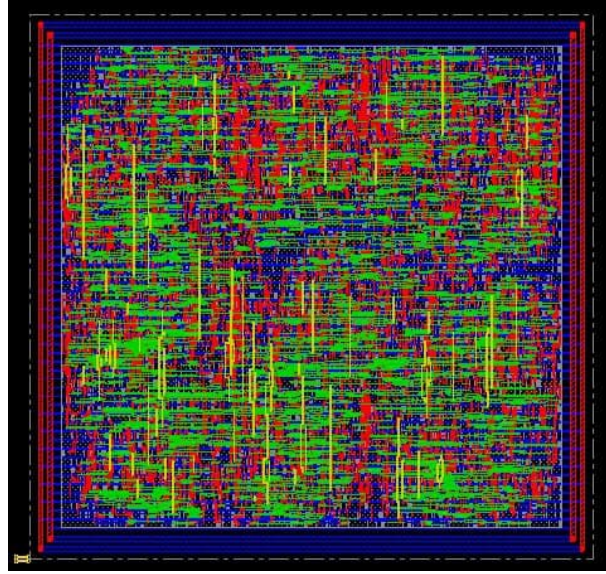
This file is converted to .saif and used by Design Compiler to generate a power report.

Power [μ W]			
Internal	Switching	Leakage	Total
249.6327	209.0005	76.498	535.1316

2.4 Place & Route

The last section requires to perform the place and route on the synthesized circuit to obtain the switching activity and power report. To do that using Innovus several steps are necessary:

- **Structuring the floorplan**, where Innovus allocates the area for the cells;
- **Inserting power rings**, two rings for power (VDD) and ground (VSS) are inserted around the floorplan;
- **Standard cell power routing**, horizontal wires for power and ground are prepared for the cells;
- **Placement**, the cells are placed in the floorplan but are still to be connected between them;
- **Post Clock-Tree-Synthesis optimization**, the design is optimized to achieve the required timing constraints;
- **Place filler**, filler cells are placed to ensure continuity in n+ and p+ wells in each row;
- **Routing**, the cells are connected among each other;
- **Post routing optimization**, the design is optimized again to achieve the timing constraints. After this step, the design is saved as a .enc file;
- **Parasitics extraction**, Innovus extracts the parasitic values of resistencies and capacitances;
- **Timing analysis**, the performance of the circuit is evaluated, if the slack is negative the constraints are violated;
- **Design analysis and verification**, Innovus checks for the presence of floating wires and violations on the constraints on the geometric features of the circuit. Finally the area and gate count, the netlist and a file with delay annotations are saved.



Since the slack values for setup and hold are positive and the verification on connectivity and geometry returned no errors, the final step is to simulate the produced netlist with Modelsim, to check if the circuit behaves correctly and to calculate the switching activity. The following results have been obtained:

Power [μW]			
Internal	Switching	Leakage	Total
207.5	175.1	73.98	516.5

Gates	Cells	Area
4488	1874	$3582 \mu\text{m}^2$

After the place and route phase, both the area and the power consumption estimations are reduced with respect to the ones obtained after the synthesis. This is possible due to the numerous steps of optimization that Innovus performs on the circuit's netlist.

CHAPTER 3

Advanced architecture development

In this section the purpose is to improve the FIR performance.

Initially the unfolding technique has been applied to improve the throughput, then pipeline technique has been implemented to reduce the critical path and improve the maximum clock frequency.

3.1 Unfolding

Unfolding of order 3 has been applied to FIR filter ($N = 3$) and the equations derived to build the new system are the following:

$$\begin{aligned} y[3n] = & a_0 \cdot x[3n] + a_1 \cdot x[3(n-1) + 2] + a_2 \cdot x[3(n-1) + 1] + \\ & a_3 \cdot x[3(n-1)] + a_4 \cdot x[3(n-2) + 2] + a_5 \cdot x[3(n-2) + 1] + a_6 \cdot x[3(n-2)] + \\ & a_7 \cdot x[3(n-3) + 2] + a_8 \cdot x[3(n-3) + 1] + a_9 \cdot x[3(n-3)] + a_{10} \cdot x[3(n-4) + 2] \end{aligned} \quad (3.1)$$

$$\begin{aligned} y[3n+1] = & a_0 \cdot x[3n+1] + a_1 \cdot x[3n] + a_2 \cdot x[3(n-1) + 2] + \\ & a_3 \cdot x[3(n-1) + 1] + a_4 \cdot x[3(n-1)] + a_5 \cdot x[3(n-2) + 2] + a_6 \cdot x[3(n-2) + 1] + \\ & a_7 \cdot x[3(n-2)] + a_8 \cdot x[3(n-3) + 2] + a_9 \cdot x[3(n-3) + 1] + a_{10} \cdot x[3(n-3)] \end{aligned} \quad (3.2)$$

$$\begin{aligned} y[3n+2] = & a_0 \cdot x[3n+2] + a_1 \cdot x[3n+1] + a_2 \cdot x[3n] + a_3 \cdot x[3(n-1) + 2] + \\ & a_4 \cdot x[3(n-1) + 1] + a_5 \cdot x[3(n-1)] + a_6 \cdot x[3(n-2) + 2] + a_7 \cdot x[3(n-2) + 1] + \\ & a_8 \cdot x[3(n-2)] + a_9 \cdot x[3(n-3) + 2] + a_{10} \cdot x[3(n-3) + 1] \end{aligned} \quad (3.3)$$

Using this method of optimization the two more input and output ports have been added because now 3 inputs are processed and produce, at the same time, 3 outputs.

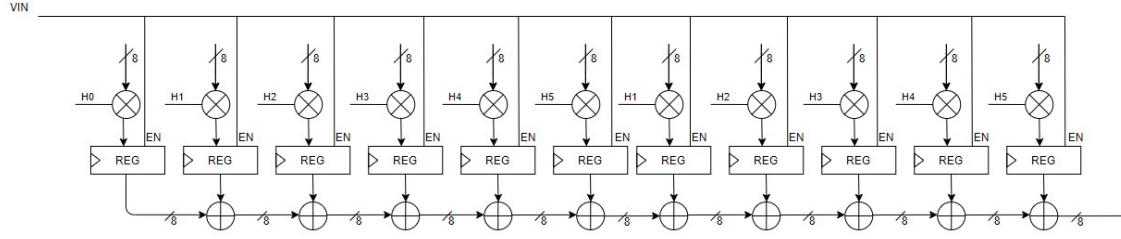
The overall throughput has been triplicated with respect to the original architecture.

3.2 Pipeline

A further optimization has been applied. This method allows to reduce the size of critical path.

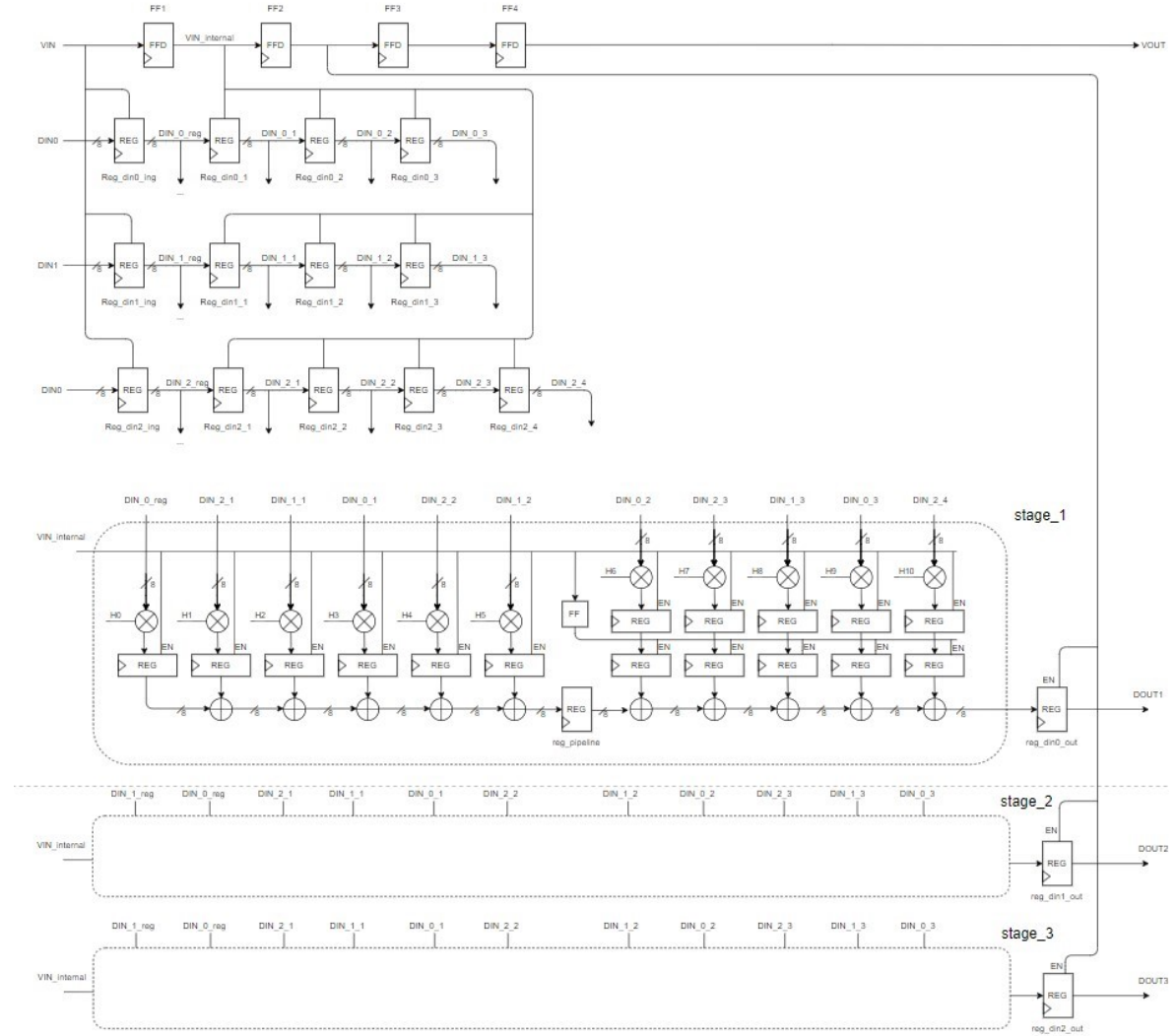
From the schematic of the unfolded FIR is possible to see that to reduce the critical path a chain of registers is needed to separate the multipliers from the adders.

After these registers are added, the new critical path becomes the long chain of adders at the bottom of the scheme.



A register is added in the middle of the adder chain; by doing so, it is necessary to delay the stages of the filter that are positioned behind the new register. Also VIN has to be carefully delayed according to which register it is enabling.

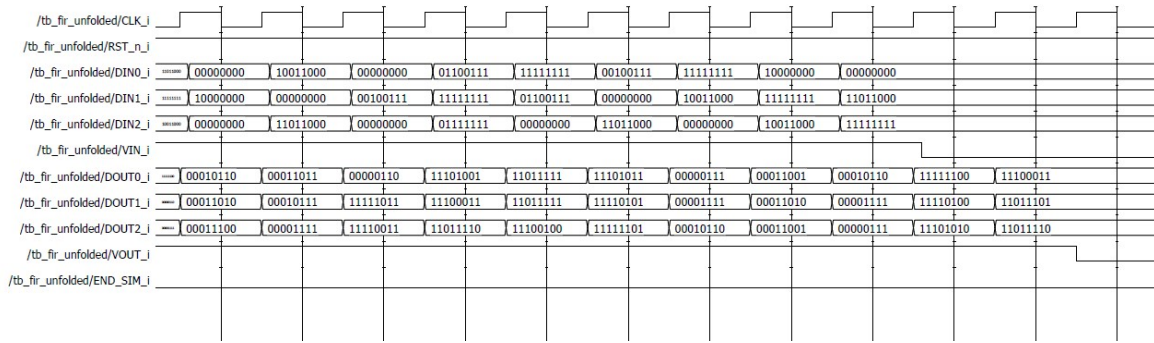
After the optimization, the new critical path corresponds to a single multiplier, so it is not possible to improve it more without adding pipelining to the arithmetic blocks.



3.3 Simulation

To simulate the advanced implementation of the filter the testbench had to be modified.

The data_sink and data_maker were updated in order to be able to transmit and receive 3 inputs every clock cycle. The first noticeable effect of the optimizations can be noted in the length of the Modelsim simulation, which changes from approximately 2100 ns to 700 ns, a third as expected due to the increased throughput.



The produced output file has been compared with the results of the C prototype as before, confirming the same behaviour.

3.4 Logic synthesis

The steps performed for the logic synthesis of the base circuit have been repeated to quantify the performance gain. Running Design Compiler with a timing constraint for the clock period of 0 ns the timing analysis returns the maximum achievable clock frequency, as well as an area estimation.

Max Clock Frequency	Min Clock Period	Area
556 MHz	1.8 ns	14740.4 μm^2

This values show that the new architecture is able to work at nearly double the frequency with respect to the previous implementation, at the expense of a much higher cost in terms of area.

The synthesis is repeated using a clock frequency equal to 25% of the maximum value previously found.

Clock Frequency	Clock Period	Area
139 MHz	7.2 ns	13399.5 μm^2

The netlist produced by Design Compiler has been used by Modelsim to check the behaviour of the circuit and to annotate the switching activity of the nodes in a .vcd file. After converting this file to .saif a power estimation was generated by Synopsys.

Power [μW]			
Internal	Switching	Leakage	Total
1780	1170.1	277.68	3227.8

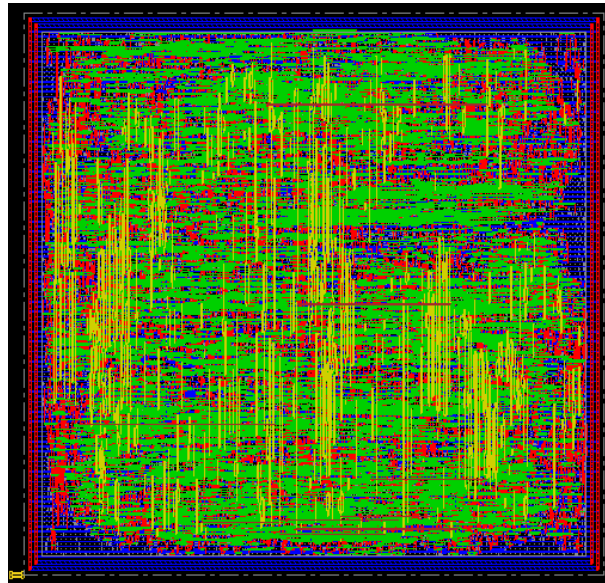
The higher number of cells results also in higher power consumption as shown in the previous

table. The higher working frequency is also a cause for the increase in consumption.

3.5 Place & Route

The place and route is performed using Innovus on the netlist produced by Synopsys, working at 139 MHz. The synthesis includes the following steps as shown in the previous chapter:

- Structuring the floorplan
- Inserting power rings
- Standard cell power routing
- Placement
- Post Clock-Tree-Synthesis optimization
- Place filler
- Routing
- Post routing optimization
- Parasitics extraction
- Timing analysis
- Design analysis and verification



After verifying that there were no violations in geometry and connections, as well as timing, the netlist with the corresponding parasitics values was simulated with Modelsim to measure the switching activity of the nodes. The behaviour was checked as well, confirming the results of the previous simulations.

After restoring the design at the routed level in Innovus, the parasitic values were extracted again and the report power was produced.

Power [μW]			
Internal	Switching	Leakage	Total
1437	796.6	258.1	2492

Gates	Cells	Area
15844	6370	12644 μm^2

Also in this case Innovus was able to reduce both area and power consumption with respect to the Synopsys evaluation, thanks to the various optimization algorithms at its disposal.