Politecnico di Torino

III Facoltà di Ingegneria

Integrated systems architecture

Laboratory 1

Laurea Magistrale in Ingegneria Elettronica

Orientamento: Sistemi Elettronici

Group n. 9

Authors:

Favero Simone S270686
Micelli Federico S270456
Spanna Francesca S278040

Repository Github: github.com/federico-micelli/ISA_2020_Group_09

Index

Reference model development

1.1 Matlab pseudo-fixed-point

The first step in the filter design process is the definition of the design parameters, according to the given specifications:

$$\begin{gathered}\mathrm{p} = 1 \\ \mathrm{x} = 6 \\ \mathrm{y} = 7\end{gathered}$$

which lead to the implementation of the following filter:

Specification	**Value**
Filter type	FIR
Cut-off frequency, fc	2 kHz
Sampling frequency, fs	10 kHz
Filter order, N	8
Number of bits, Nb	8

In the Matlab script *my_fir_design.m*, a function able to return the values of the filter coefficients is employed. This function receives as parameters the order of the filter and the cut-off frequency normalized with respect to the Nyquist frequency. Those coefficients are subjected to a quantization operation over $\mathrm{N_b}$ bits and expressed both in integer and real form. Moreover, it is possible to esimate the effect of this quantization by comparing the designed transfer function to the quantized one.

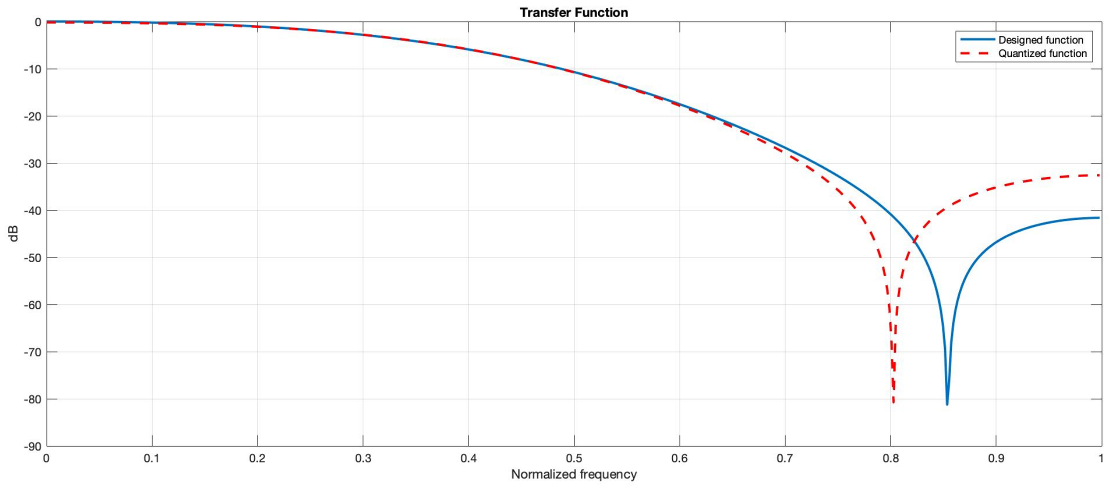

Figure 1.1: Transfer functions comparison

It can be noticed from the graph above that the quantized transfer function is characterized by a lower cut-off frequency.

In the Matlab script *my_fir_filter.m*, the filter behavior is tested with a combination of two sinewaves, characterized by different frequencies.

Signal	**Frequency [Hz]**
x1[n]	500
x2[n]	4500

It is expected from the implemented FIR to filter out the component related to x2[n], since it is not included in the bandwidth. The result of the filtering operation can be observed in time domain.

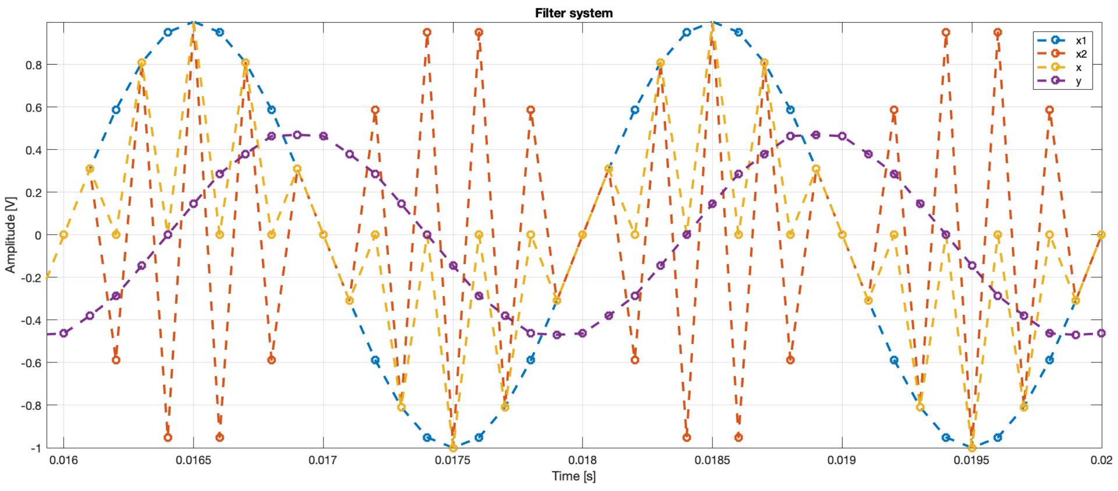

Figure 1.2: Filter time behavior

As visible from the graph above the signal x1[n], with a frequency of 500 Hz, is well represented. That is because, for the given sampling frequency, a sufficient number of samples is collected every period. This condition does not apply for the signal x2[n].
By simply observing the graph, the output of the filter is the expected one, since the contribution of the signal x2[n] is negligible.

In order to test the behavior of the filter using a C model implementation, coefficients, input and output samples are stored in proper text files as integer values.

1.2 Fixed-point c model

The aim of this section is to develop a C model of the filter to emulate the fixed point architecture. Since it is a finite impulse resposte filter, the generic equation to be implemented is the following one:

$$\sum_j x_{\text{i-j}} \cdot b_{\text{j}}$$

The coefficients are declared as a constant array of integers. A proper function is defined in order to evaluate the output y[n] at a specific time instant, knowing the input sample x[n]. This operation is repeated for all the samples contained in the file *samples.txt*.
It is important to notice that an internal buffer is needed to store the previous input values and shift them for each function call.

The output is obtained by summing coefficient-input products using a for loop. For each add operation, a shift on the result is introduced before storing it into the accumulator, in order to emulate the finite internal parallelism of the hardware architecture. This truncation operation introduces an error in the evaluation of the output samples.

Since the multiplication is between 8 bits data, the maximum accurancy would be obtained with a parallelism of 16 bits. The initial choice for the shift operation consists of (N_{b}-1) bits, corresponding to an internal parallelism of 9 bits.

With an ideal filtering operation the output signal should be a pure 500 Hz sinusoid, but in reality an harmonic distortion contribution is still present. The latter can be evaluated with the THD parameter: a further Matlab script has been developed for this task. In this first case of study the obtained THD is:

$$\text{THD}_{\text{initial}} = \text{-38.8 dB}$$

Since it is requested a THD lower or equal than -30 dB, the result fits the constraint. However, this could not be the best solution also considering the minimization of the area. For this reason the variation of the THD with the internal parallelism has been studied, obtaining the following graph.

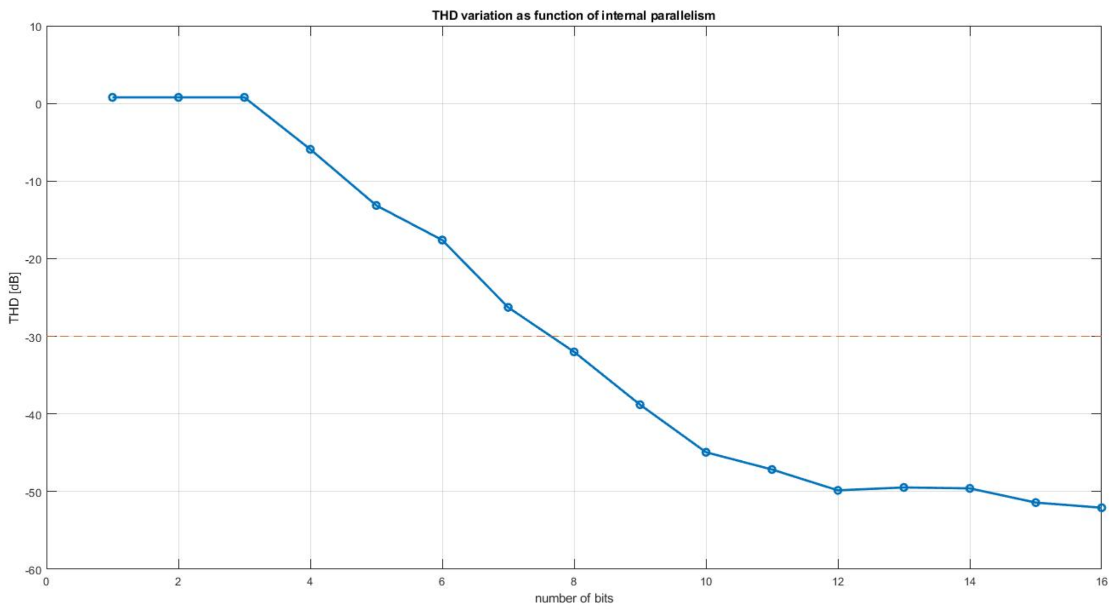

Figure 1.3: THD variation as function of internal parallelism

From this result it is possible to set the internal parallelism to 8 bits, reducing the overall area and maintaining a THD lower than the bound:

$$\text{THD} = \text{-32 dB}$$

VLSI implementation

2.1 Starting architecture development

The purpose of this section is to develop a behavioral hardware description of the designed filter. In order to do that, a hierarchical approach has been used. Therefore, the following components have been implemented through a VHDL description:

- Flip flop
- 8-bit register
- Adder
- Multiplier

Elements have been instantiated according to the data flow graph represented in the picture below, which shows a direct form implementation of an FIR filter.

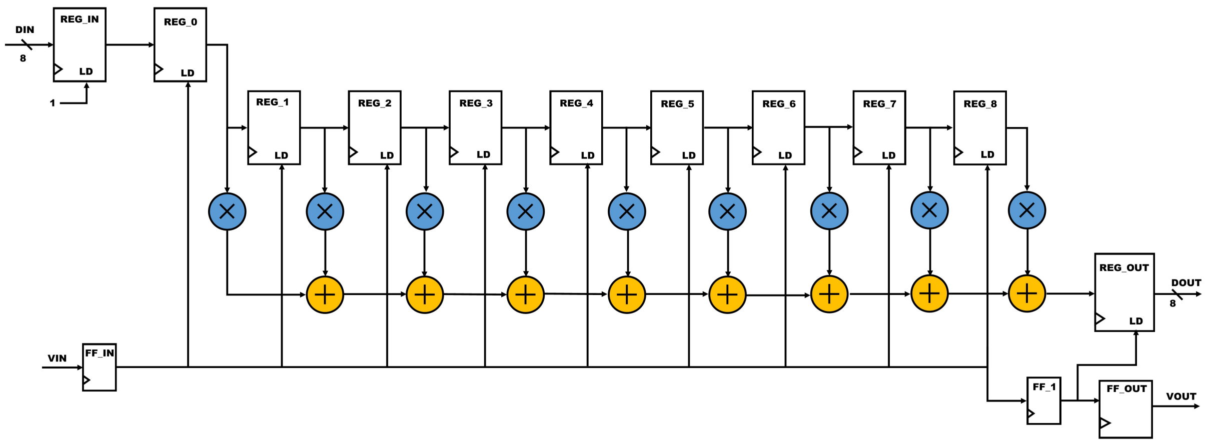

Figure 2.1: Data flow graph

As visibile in the designed data flow graph, the aim of REG_IN and FF_IN is to interface the FIR system with external input signals. Although they are fully optional for the functionality, their use is suggested in order to avoid external signals to control internal registers.
For the same reason, output data are provided with REG_OUT and FF_OUT.

It is also important to notice the presence of REG_0: it has been instantiated in order to reduce the overall power consumption. Indeed, in the case of VIN = 0, the absense of REG_0 could lead to a propagation of unwanted commutations along the combinational blocks.

As a consequence an additional flip flop, FF_1, has to be inserted to take into account the increment of latency of one clock cycle.

In order to control registers' loading operations and drive VOUT, a delayed version of VIN is used. This is implemented by means of a cascade of flip-flops.

2.2 Simulation

Simulation is intended to verify the correct behavior of the designed architecture through a mixed VHDL-Verilog testbench. Also in this case a hierarchical description has been chosen. In particular the following entities are defined:

- **Clk_rst_gen**: it generates a clock signal with a given period Ts; moreover a reset signal is asserted during the first clock cycles, in order to simulate the initial reset condition.
- **FIR**: it is the device under test, developed in the previous section.
- **Read_data**: this block is able to read from external files both the values of the coefficients and the input samples, on *coefficients.txt* and *samples.txt* respectively. In particular, the former ones are assigned at the beginning of the simulation, while the latter ones are provided with a customizable periodicity given by *data_clock* signal. When an input sample is available, a proper signal reports the validity of the input. When the end of input samples file is reached, a signal whose purpose is to stop the clock generation is asserted: this allows to end the simulation correctly.
- **Write_results**: every output sample computed and made available by the device under test is written on a proper text file, *results_vhdl.txt*.
- **Error_check**: it controls the correctness of the values obtained by the hardware architecture. In particular, the file *results_c.txt* is read and compared line by line to the output of the filter structure. An output report file, *error_check.txt*, contains the result of the comparison. When an error is detected, it is highlighted giving the expected value and the computed one.

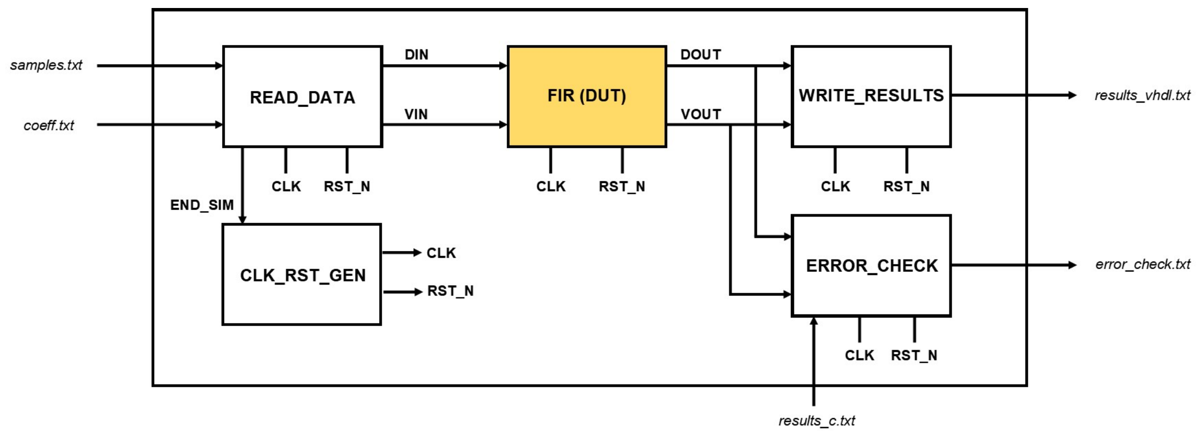

Figure 2.2: Testbench block scheme

The simulation has been launched with a periodicity of the input samples equal to two clock cycles.
The error check files does not contain any error report, since the VHDL results are equal to the C implementation ones. The first ten results for both files are reported in the following table.

Row	***results_c.txt***	***results_vhdl.txt***
0	0	0
1	-1	-1
2	-1	-1
3	-1	-1
4	4	4
5	8	8
6	17	17
7	21	21
8	26	26
9	26	26

Moreover, the timing of the system can be observed in the following waveform.

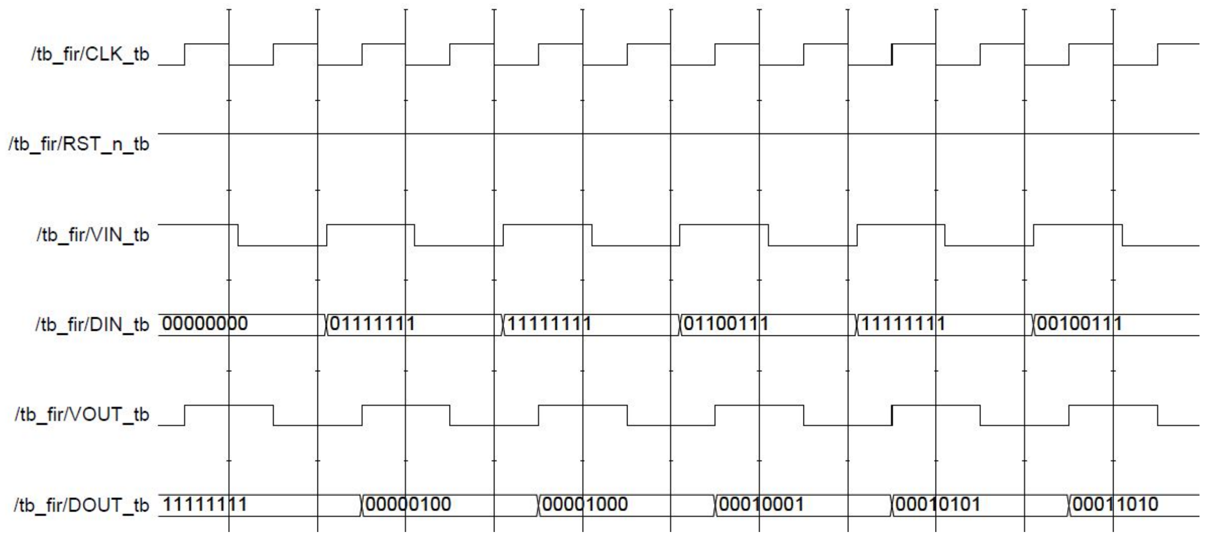

Figure 2.3: Waveforms timing diagram

It can be noticed that the filter is able to work correctly if the input samples are provided with a different periodicity with respect to the clock cycle, since its behaviour is only dependent on the VIN signal.

2.3 Logic synthesis

The aim of this section is to synthesize the logic design on a given cell library and perform analysis on timing, area, and power consumption.

First of all, it is requested to estimate the maximum clock frequency achievable by the circuit. In order to do that, VHDL files must be analyzed and the top level entity must be elaborated choosing the desired architectural description.

The next step consists of defining a clock signal with a specified period. In particular, if the clock period is set equal to 0 ns, this constraint can be exploited to find the maximum working frequency.
In this way the synthesizer optimizes as much as possibile the design: as a consequence the slack reported by the timing analysis corresponds to the minimum clock period.

After that, it is possibile to change the clock period to its minumum value, found in the previous analysis, and get an area estimation.

Minimum clk period	**Maximum clk frequency**	**Area**
2.88 ns	347.2 MHz	3198 um^2

For the next steps, the working frequency has been fixed to the following value:

$$f_{\text{clk}} = \frac{f_{\text{max}}}{4} = 86.8 \text{ MHz}$$

It is necessary to close and re-open the design compiler in order to avoid the new estimations to be affected by the previous optimizations. As a consequence, a new area is estimated:

$$\text{Area} = 3045 \text{ um}^2$$

It can be noticed how a more relaxed constraint on the timing leads to a smaller circuit size.

The next task consists in obtaining an accurate power consumption estimation of the designed filter, using the back annotation method.
Initially, a verilog netlist is generated using Synopsys, associating to it a .sdf file reporting circuit delays.

Then it is possible to run the developed testbench, using ModelSim, taking into account the new hardware description provided by the netlist. In particular, this simulation allows to get a realistic estimation of the switching activity for each node. This information is stored in a .vcd file, which has to be converted to .saif format since Synopsys must be able to read it.
Moreover, it is possible to check che correctness of the actual implementation, verifying that the netlist results are equal to the ones obtained with the VHDL description.

Finally, the last step consists of performing a power report based on these switching acitivities. The obtained results are reported in the following table.

Power [uW]			
Leakage	Switching	Internal	Total
63	99.68	144.55	307.23

2.4 Place & route

This section is intended to perform the place and route procedure on the designed circuit using the software Innovus.

The complete procedure is composed by several steps. First of all, it is necessary to import the netlist design provided by Synopsys. Using Innovus, it is possible to define the desired floorplanning and, consequently, the structure for the power rings. After that, the power planning is completed and a feedback of the implemented operations is visible on the graphical interface.

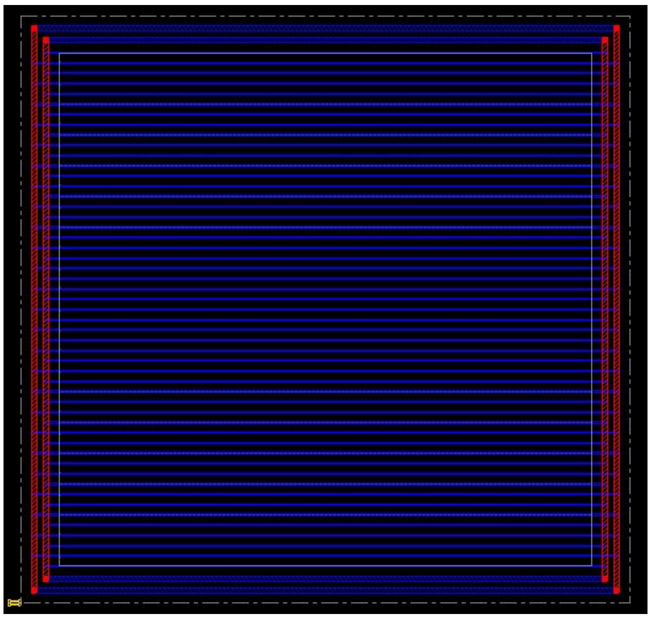

Figure 2.4: Floorplanning and power distribution

Then the standard cells are positioned through the placement operation; in addition, in order to ensure continuity of n-well and p-well regions, it is possible to introduce some filler cells.

Routing is the next operation to be performed; it consists of placing interconnections between cells in a proper way. The final result is visible in the following picture.

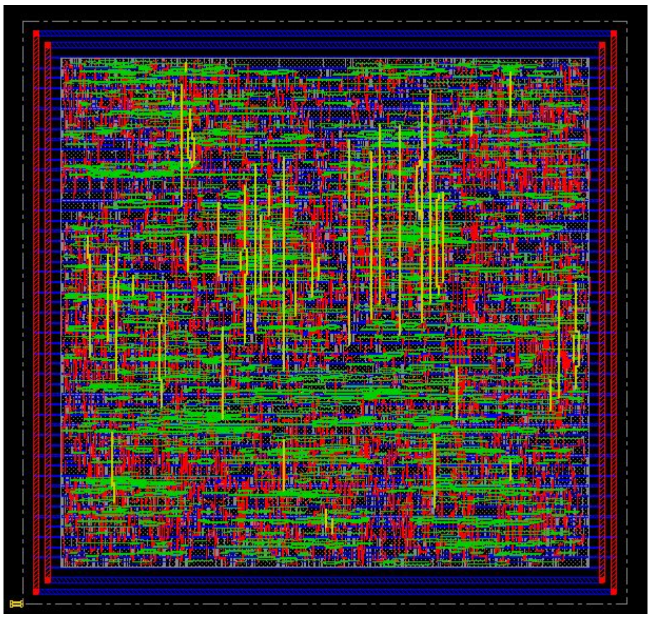

Figure 2.5: Final result of place & route

Moreover, after the placement and routing phases, it is possible to perform additional optimizations, in order to achieve the required time constraints.

Before proceding with the next tasks, it is necessary to verify both the connectivity and geometry, in order to check the presence of possible violantions.
Another important control consists of a timing analysis for both setup and hold conditions, verifying that all reported slacks are positive.
Then, also in this case, the resulting netlist and the related .sdf delays file can be saved.

As before, a new estimation of the switching activities is performed by ModelSim, along with the verification of the correctness of the output results. These activities can be exploited by Innovus to provide an estimation of the power consumption; before proceding with it, it is important to extract the parasitics from the design. The following results have been obtained:

Power [uW]			
Leakage	Switching	Internal	Total
62	243.50	350.30	655.80

An increment of the dissipated power, in particular on the switching and internal contributions, can be noticed. A possible explaination is related to the additional power consumption modeled due to the presence of interconnections between cells, that is not taken into account in the previous synthesis operations.

Finally, an area estimation has been obtained.

Gates	Cells	Area [um^2]
3793	1523	3027

A further consideration is related to the total cell area. Indeed, after the place and route phase, its value is slightly reduced with respect to the one obtained after the synthesis performed by Synopsys. This decrement is expected, since Innovus has applied several optimization to the final design.

Advanced architecture development

The aim of this chapter is to apply some universal techniques in order to improve the performances of the designed filter. In particular, L-unfolding and pipeline techniques are exploited to increase the throughput of the machine.

3.1 Unfolding optimization

The unfolding method is applied considering an unfolding degree equal to 3.

In order to obtain the final architecture, the equations approach has been implemented and it has provided the following results:

$$\begin{aligned} y[3k] = {} & B0x[3k] + B1x[3(k-1)+2] + B2x[3(k-1)+1] + B3x[3(k-1)] + B4x[3(k-2)+2] + \\ & + B5[3(k-2)+1] + B6x[3(k-2)] + B7x[3(k-3)+2] + B8x[3(k-3)+1] \end{aligned}$$

$$\begin{aligned} y[3k+1] = {} & B0x[3k+1] + B1x[3k] + B2x[3(k-1)+2] + B3x[3(k-1)+1] + B4x[3(k-1)] + \\ & + B5x[3(k-2)+2] + B6x[3(k-2)+1] + B7x[3(k-2)] + B8x[3(k-3)+2] \end{aligned}$$

$$\begin{aligned} y[3k+2] = {} & B0x[3k+2] + B1x[3k+1] + B2x[3k] + B3x[3(k-1)+2] + B4x[3(k-1)+1] + \\ & + B5x[3(k-1)] + B6x[3(k-2)+2] + B7x[3(k-2)+1] + B8x[3(k-2)] \end{aligned}$$

As visible in the equations above, each output can be evaluated by proper addition and multiplication operations, implemented in a substructure like the following one. Therefore, the entire architecture includes three of them.

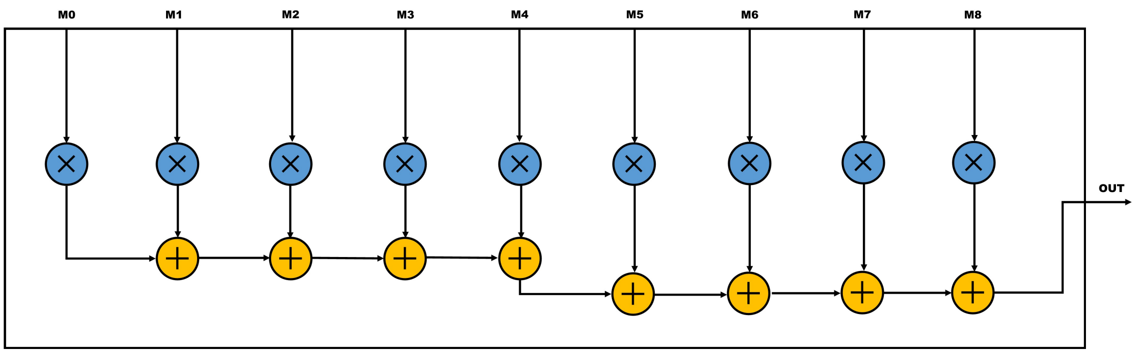

Figure 3.1: Unfolding substrucure to be repeated

Results can be computed by connecting properly delayed inputs with the correct multiplier in these substructures. As reported in the shown equations, the input of type *3k* requires 2 delay registers, while inputs *3k+1* and *3k+2* need 3 delay registers each.

The final architecture will be shown after the application of pipeline technique.

3.2 Pipeline optimization

The aim of this optimization is to reduce the critical path of the original circuit, so that the clock frequency can be increased.

After the application of the unfolding method, it is possible to notice that the combinational logic is only present inside the three substructures. Since these ones are equal to each other, the study of pipeline optimization can be focused on their general scheme, represented in figure 3.1.

Referring to the scheme previously presented, the critical path can be estimated as follows:

$$\mathrm{T_{cp}} = \mathrm{T_m} + 8 \cdot \mathrm{T_a}$$

where $\mathrm{T_m}$ and $\mathrm{T_a}$ are the delays associated respectively to the multiplier and to the adder.

A feedforward cut-set can be identified between multipliers and adders; since multipliers are, in general, expensive in terms of delay introduced, a first pipe level has been inserted.

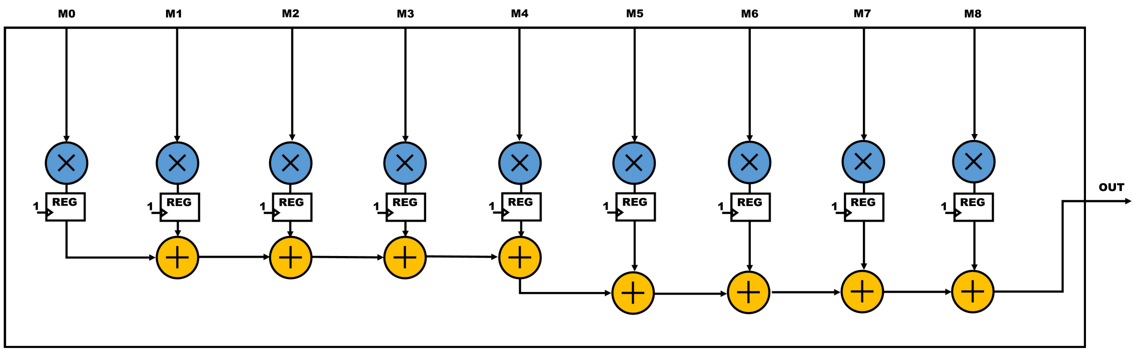

Figure 3.2: First level of pipe

After the insertion of this first level of pipe, the critical path has been modified as:

$$\mathrm{T_{cp}} = \max(\mathrm{T_m},\ 8{\cdot}\mathrm{T_a})$$

In order to check if a further improvement is needed, a timing report has been performed through Synopsys design compiler. The result showed that the critical path is the one associated to the adders cascade.

As a consequence, a second level of pipe has been introduced between 4th and 5th adder. Following the feedforward cut-set rule, additional registers must be inserted at the input of the following accumulation stages, as shown in the following figure.

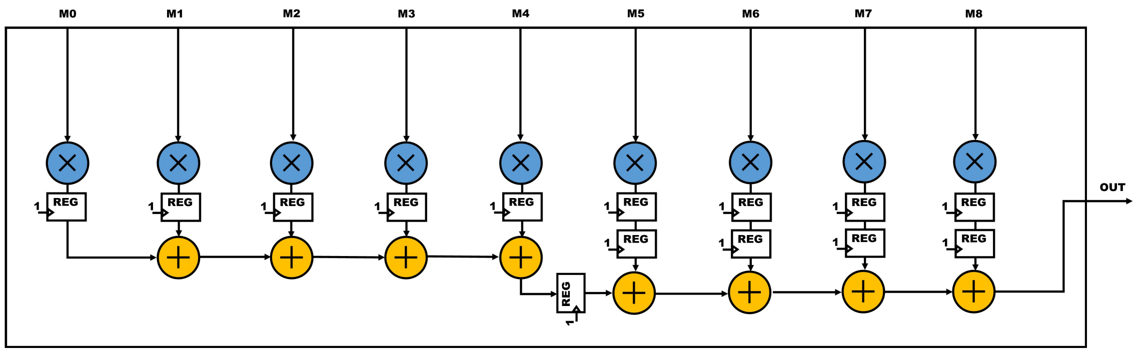

Figure 3.3: Second level of pipe

A further timing analysis has reported that the critical path is now associated to the multiplier operator. Therefore, since it is requested to implement a coarse grain pipeline, additional pipeline registers would not improve performances.

Moreover, in addition to the increased throughtput, an important consequence due to the pipeline introduction is the increment of the latency needed to process a sample. In this case, since two pipeline stages have been inserted, latency will increse by two clock cycles.

3.3 Final architecture

The optimized architecture includes the following elements:

- **I/O interfacing registers**, already present in the original structure.
- **Flip flop set** that allows the correct propagation of the signal VIN to VOUT, taking into account the additional latency introduced by the pipeline.
- **Delay registers** for input samples, considering, in addition to the ones achievable by the equations of the L-unfolding method, one register for each input type to guarantee the correct timing.
- **Three computational substructures** to implement multiply and accumulate operations.

NOTE: due to the complexity of the datapath representation, connections have been indicated by means of proper net labels.

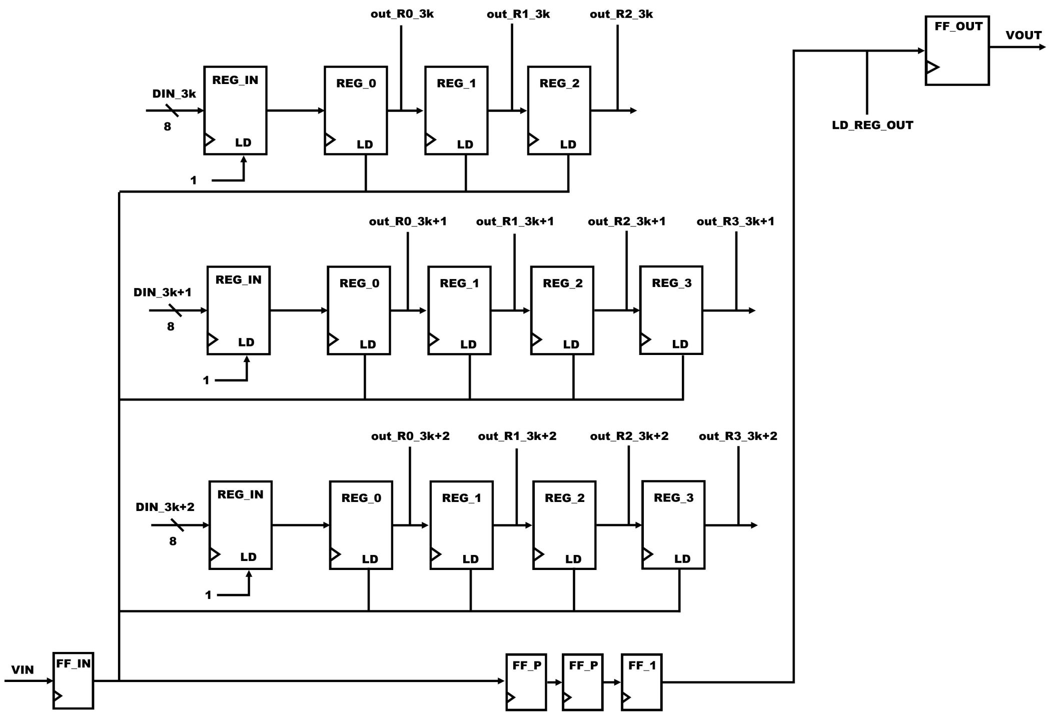

Figure 3.4: Final architecture - 1

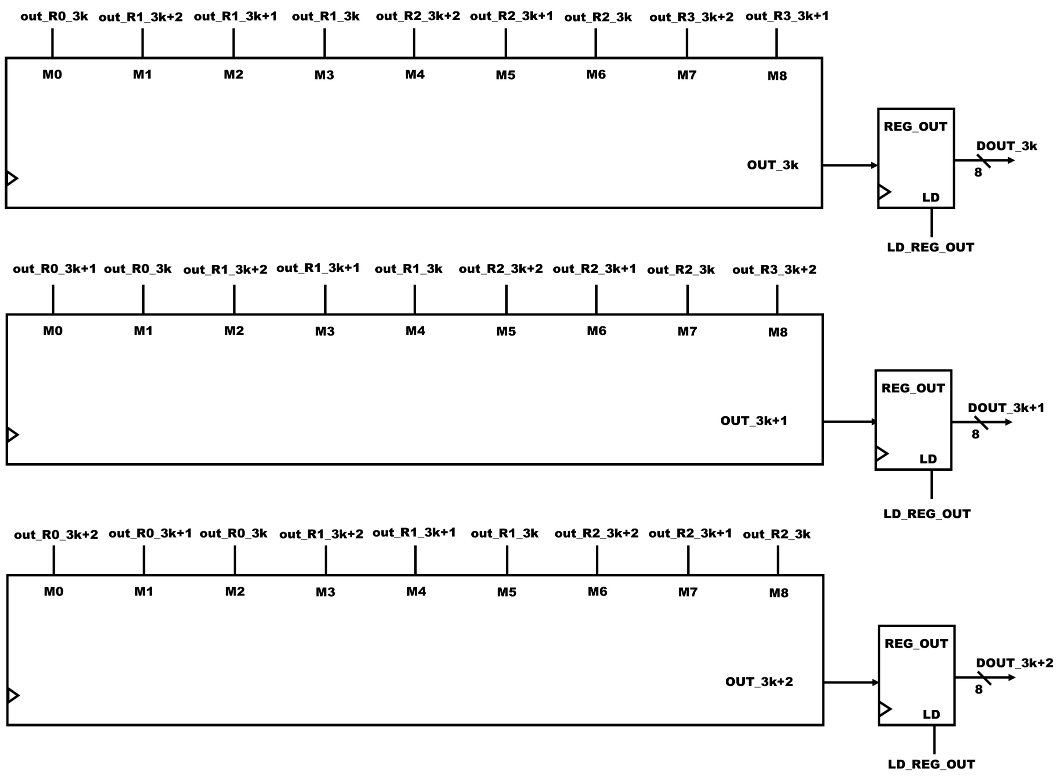

Figure 3.5: Final architecture - 2

3.4 Simulation

The testbench developed in chapter 2 has been adapted to the optimized architecture, which relies on the unfolding method. In particular, *Read_data.vhd* provides three samples to the filter, while *Write_results.vhd* and *Error_check.vhd* collect and write three output data at a time, checking the correctness of the results.

Moreover, the additional latency introduced by the pipeline has been taken into account in the management of the simulation's end.

Also in this case, no errors were found in the results comparison, as noticeable in the file *error_check.txt*. As before, the first ten results are reported in the following table.

Row	**C implementation results**	**Optimized VHDL results**
0	0	0
1	-1	-1
2	-1	-1
3	-1	-1
4	4	4
5	8	8
6	17	17
7	21	21
8	26	26
9	26	26

In order to appreciate the improvements related to the unfolding optimization, the simulation has been run with the same clock period for both original and optimized implementations. Since both architectures work with the same clock frequency, the benefits of pipeline have not been noticeable in the throughput.

The end of the simulations is shown in the following pictures.

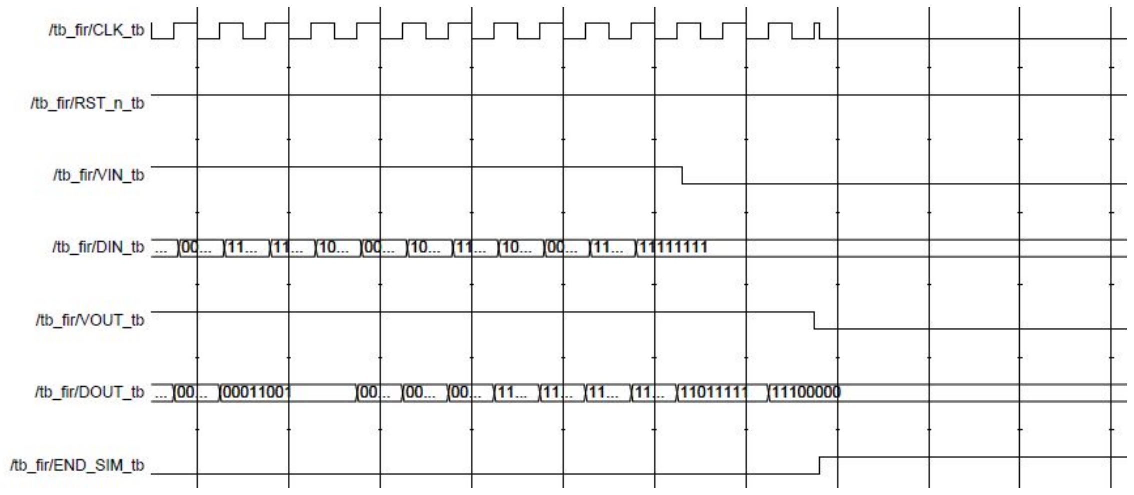

Figure 3.6: Simulation - original architecture

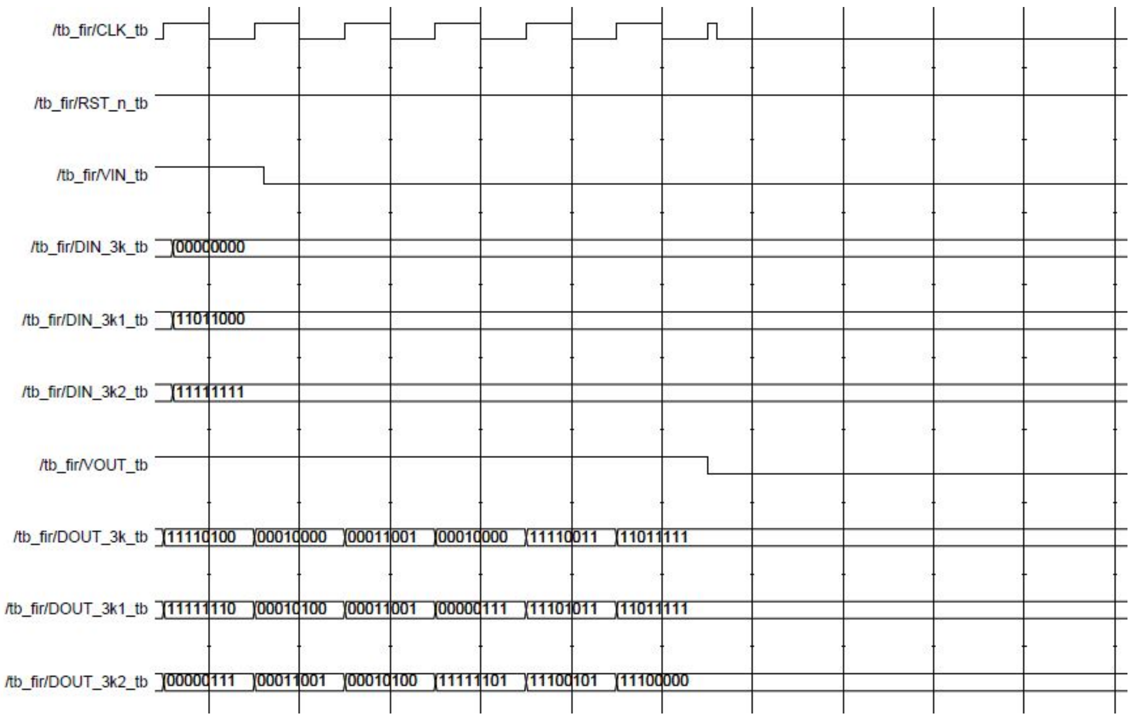

Figure 3.7: Simulation - optimized architecture

As expected the second simulation shows that the architecture is able to manage three samples at the same time. The end of simulation time instants have been measured in both cases:

Original architecture	**Optimized architecture**
2056 ns	736 ns

A significant improvement to the performances has been noticed.

3.5 Logic synthesis

In order to fully appreciate the overall performance improvement, it is necessary to estimate the new critical path consequently of the pipeline insertion.

The new VHDL design is mapped on the given cell library, defining a clock period equal to 0 ns, in order to find the critical path delay and the maximum frequency associated to it. After that, another compilation has been performed, declaring a clock period equal to the slack previously found. The results are reported in the following table.

Minimum clk period	**Maximum clk frequency**	**Area**
1.41 ns	709.2 MHz	11977 um^2

In the case of maximum working frequency, it is possible to notice a significant area increment in the optimized design with respect to the one reported in section 2.3: this increment is mostly related to the application of the unfolding method.

On the other hand, as expected, the minimum clock period has been reduced by a factor equal to 2.04.
Considering the total increment of performances, due to both pipeline and unfolding techniques, throughput has been improved by a factor $2.04 \cdot 3 = 6.12$. As a drawback, the area has been increased by a factor 3.75. Therefore the cost of the architecture is increased, as well as the power consumption.

As in the section 2.3, a more detailed analysis has been performed imposing the following working frequency:

$$f_{\text{clk}} = \frac{f_{\text{max}}}{4} = 177.3 \text{ MHz}$$

As a consequence a new area has been estimated:

$$\text{Area} = 10598 \text{ um}^2$$

In order to obtain an estimation of the power consumption, a netlist generated by Synopsys has been used in a Modelsim simulation to annotate the switching activities of each node. Moreover, the correctness of the output results has been checked.
The simulation results are used by Synopsys to generate the following power report.

Power [uW]			
Leakage	Switching	Internal	Total
220.44	278.99	575.77	1075.20

The power consumption is increased of a factor 3.5 with respect to the power dissipation estimated in the original architecture. As already mentioned, the higher number of cells takes part in the increment of the power consumpion, in particular considering the leakage contribution. A further cause is the new working frequency, which has been set to a higher value.

3.6 Place & route

Following the specifications, the place and route precedure has been performed using the netlist generated by Synopsys, imposing a clock frequency equal to 177.3 MHz. The following steps have been completed as in section 2.4.

- Design importing
- Floorplanning
- Power rings insertion
- Placement
- First optimization phase
- Filler cells insertion
- Routing
- Second optimization phase

The result of these operations is represented in the following picture.

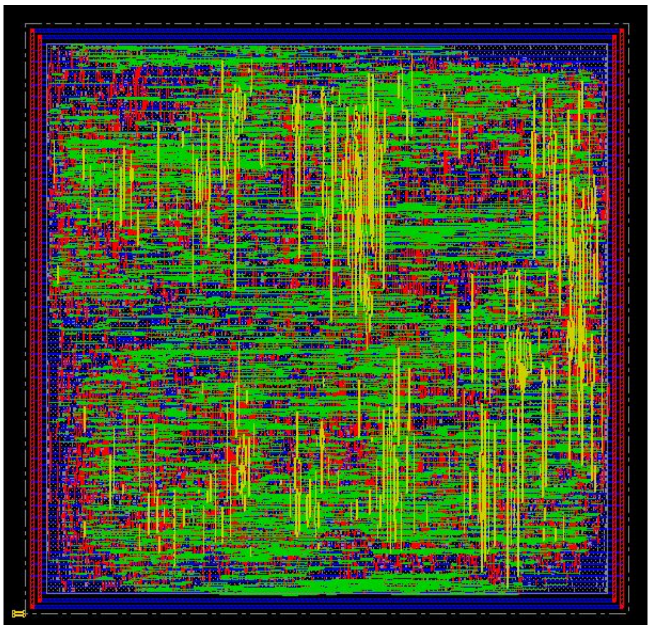

Figure 3.8: Place & route representation

Comparing the figures 2.4 and 3.8, the latter presents a larger number of elements in the visual representation of the integrated circuit.

At this point, the parasitics have been extracted and a timing analysis has been performed: the aim is to verify that all the reported slacks assume positive values, both for the setup and hold conditions.
Also in this case, the connectivity and geometry have been checked to avoid possible violations.

A Modelsim simulation on the new netlist has been exploited to estimate a realistic activity map of the nodes through the back-annotation method.
Consequently, power consumption and area have been estimated by Innovus.

Power [uW]			
Leakage	Switching	Internal	Total
198.3	356.3	753.6	1308.2

Gates	Cells	Area [um^2]
12372	4468	9873

As noticed in section 2.4, also in this case the total dissipated power is increased with respect to the Synopsys estimation. That is because of the interconnections that are taken into account. The optimizations provided by Innovus were able to slightly reduce the area occupation.