Politecnico di Torino

III Facoltà di Ingegneria

Exercises and Homeworks for the course Integrated Systems Architecture

LAB 1: design and implementation of a digital filter

Master degree in Electronic Engineering

Group number: 37

Costantino Taranto - 274492
Pasquale Santoro - 278329
Alberto Aimaro - 253196

November 18, 2020

Repository Link: `https://github.com/isa037/lab1`

Contents

0.1 Introduction . . . 1
0.2 GitHub Repository . . . 1

1 Reference model development **2**
1.1 Filter design and coefficient quantization using Matlab . . . 2
1.2 Testing the filter . . . 3
1.2.1 First step: Matlab pseudo-fixed-point . . . 3
1.2.2 Second Step: fixed-point C model . . . 4

2 First architecture development **6**
2.1 Developement of the VHDL model of the filter and verification . . . 6
2.1.1 VHDL model . . . 6
2.1.2 TestBench design and VHDL model verification . . . 8
2.2 Logic synthesis . . . 9
2.2.1 Maximum clock frequency evaluation . . . 11
2.2.2 Area evaluation . . . 11
2.3 Synthesis with $f_{clk} = f_{max}/4$ . . . 11
2.3.1 Area evaluation . . . 11
2.3.2 Design Verification . . . 11
2.3.3 Power consumption estimation . . . 12
2.4 Place & Route . . . 13
2.4.1 Parasitics extraction . . . 17
2.4.2 Timing analysis . . . 17
2.4.3 Connectivity and Design rules verification . . . 18
2.4.4 Area evaluation . . . 19
2.4.5 Modelsim simulation and swithing activity recording . . . 19
2.4.6 Power consumption estimation . . . 19

3 Advanced architecture development **21**
3.1 Implementation of architecture techniques on the FIR . . . 21
3.1.1 Unfolding . . . 21
3.1.2 Pipelining . . . 23
3.2 Development of the VHDL model of the filter and verification . . . 23
3.2.1 VHDL model . . . 23
3.2.2 TestBench design and VHDL model verification . . . 25
3.3 Logic synthesis . . . 26
3.3.1 Max clock frequency evaluation . . . 26
3.3.2 Area evaluation . . . 26
3.4 Synthesis with $f_{clk} = f_{max}/4$ . . . 26

3.4.1 Area evaluation . . . 26
3.4.2 Design Verification . . . 26
3.4.3 Power consumption estimation . . . 26
3.5 Place & Route . . . 27
3.5.1 Parasitics extraction . . . 27
3.5.2 Timing Analysis . . . 28
3.5.3 Connectivity and Design rules verification . . . 28
3.5.4 Area evaluation . . . 29
3.5.5 Modelsim simulation and switching activity recording . . . 29
3.5.6 Power consumption estimation . . . 30

4 Comparison of the two architectures 31

5 Appendix 32
5.1 Results of the simulation compared to the C model output . . . 32
5.2 Results of the unfolded FIR simulation compared to the C model output . . . 36

0.1 Introduction

The aim of this lab is to implement a digital filter via hardware. From the specification our group (No. 37) has to design a **FIR filter** with the following parameters:

- N = 8 (Order of the filter)
- $n_b = 8$ (Number of bits for input and output data)

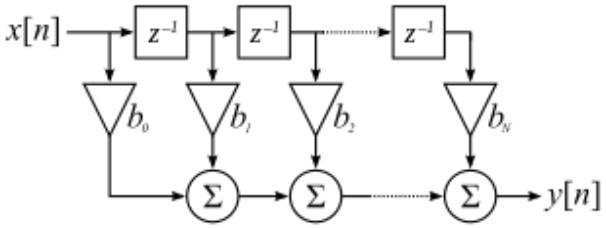

Figure 1: Generic digital FIR filter.

The FIR filter can be described by the following discrete equation: $y[n] = \sum_{i=0}^{N} x[n-i] * b[i]$

The step followed can be summarized as:

- Filter design and coefficient quantization, via software
- Development of a fixed-point C model
- Architecture development using VHDL
- Advanced architecture development applying unfolding and pipelining techinques
- Testing of the architectures using a mixed VHDL/verilog testbench
- Place & route of the filters and comparison between their behaviour and the model's one

0.2 GitHub Repository

The whole Laboratory has been carried out using the *GitHub* versioning tool. The repository containing all the files used and produced can be found at the following link:

`https://github.com/isa037/lab1`

CHAPTER 1

Reference model development

The cutoff frequency of the filter is required to be $f_c = 2kHz$, while the sampling frequency is $f_s = 10kHz$. Known the FIR order and the number of bits of numerical representation, the filter can be easily designed using Matlab.

1.1 Filter design and coefficient quantization using Matlab

Using the *myfir_design.m* script, which exploits the *fir1* function, with N=8 and n_b=8 the following floating point coefficients are computed:

$$b[n] = [-0.0061, -0.0136, 0.0512, 0.2657, 0.4057, 0.2657, 0.0512, -0.0136, -0.0061]$$

these coefficients had been quantized for 8 bits fractional point representation:

$b[n] = [-0.0078125, -0.015625, 0.046875, 0.265625, 0.3984375, 0.265625, 0.046875, -0.015625, -0.0078125]$

For ease of representation, they will be reported in an 8-bit integer form:

$$b[n] = [-1, -2, 6, 35, 51, 34, 6, -2, -1]$$

In figure 1.1 the comparison between the floating point and discrete point FIR response is shown.

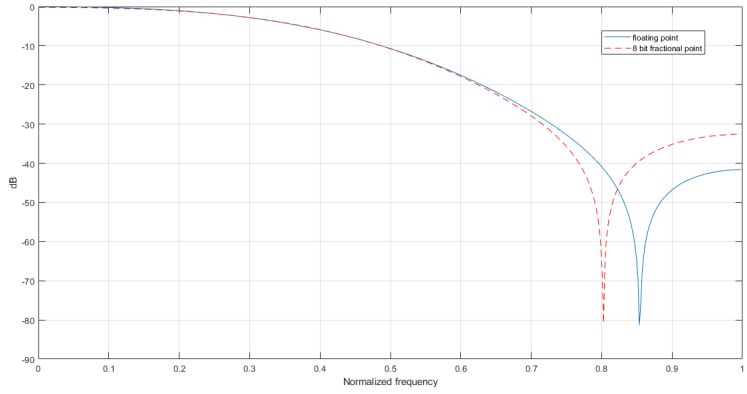

Figure 1.1: FIR frequency responses

The frequency response of the discrete filter is different, since the coefficients are different. However, the behaviour around the cutoff frequency (which has a normalized value of 0.4) is quite similar.

1.2 Testing the filter

1.2.1 First step: Matlab pseudo-fixed-point

After its design, a test of the filter is performed. It is done by creating a signal (named x) made of two sinusoidal waves (one "in band" and the other "out of the band", named *x1* and *x2* respectively) and feeding the FIR with it. The input signal together with its two components is showed in figure 1.2 , while the output is showed in figure 1.3.

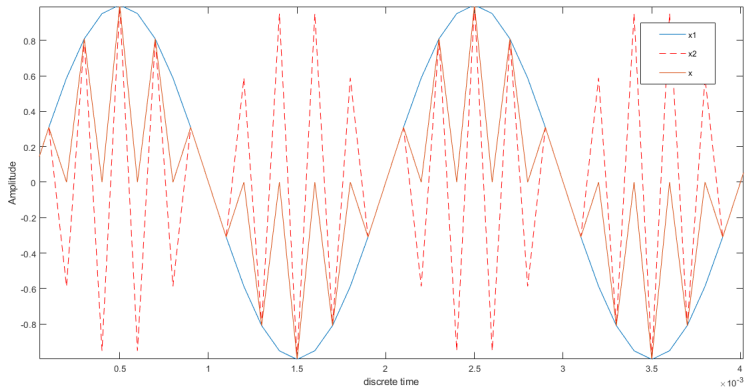

Figure 1.2: Input signal generation: *x=x1 + x2*

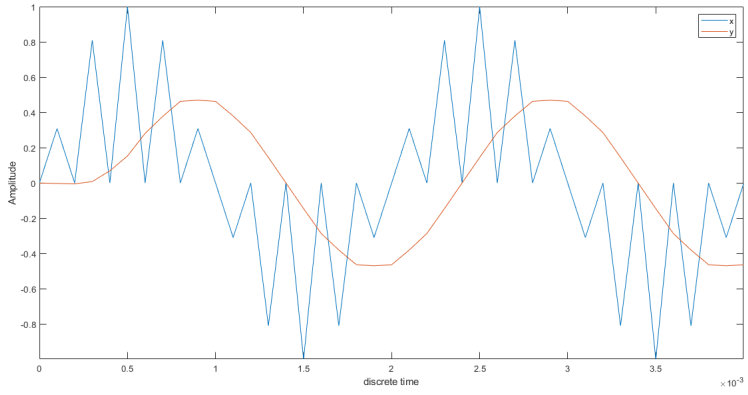

Figure 1.3: Fir output when the input is x

Notice that the output y of the filter is quite similar compared to $x1$. This is because it's the only component of x which falls in the band of the filter, while $x2$ gets "cut". The input and output samples are saved in two different files (*samples.txt* and *resultsm.txt*), represented as 8 bit integer values.

1.2.2 Second Step: fixed-point C model

The next step is two write in C language a fixed point implementation of the filtering operation, that is:

$$y_i = \sum_{i=0}^{8} x_{i-j} * b_j \tag{1.1}$$

where x_i, y_i and b_i are the input samples, output samples and filter coefficients respectively. The form used is the **direct form**. The formerly mentioned code is written in file *myfilter.c* . After performing the multiplications, the 7 LSBs are removed with a shift operation in order to consider the scale factor on the coefficients (which are fractional point numbers in contrast to samples which are integers).
The C code, like the Matlab script, saves the output results in a file, here named *resultsc.txt*.
Importing the output of the C fixed-point program in Matlab, it is possible to evaluate the Total Harmonic Distortion (THD) of the result. This is done in the script *thd_evaluation.m*. The **THD** computed value is of **-31.86dB**, which meets the -30dB requirement.
In figure 1.4 the output of matlab *thd* function is shown.

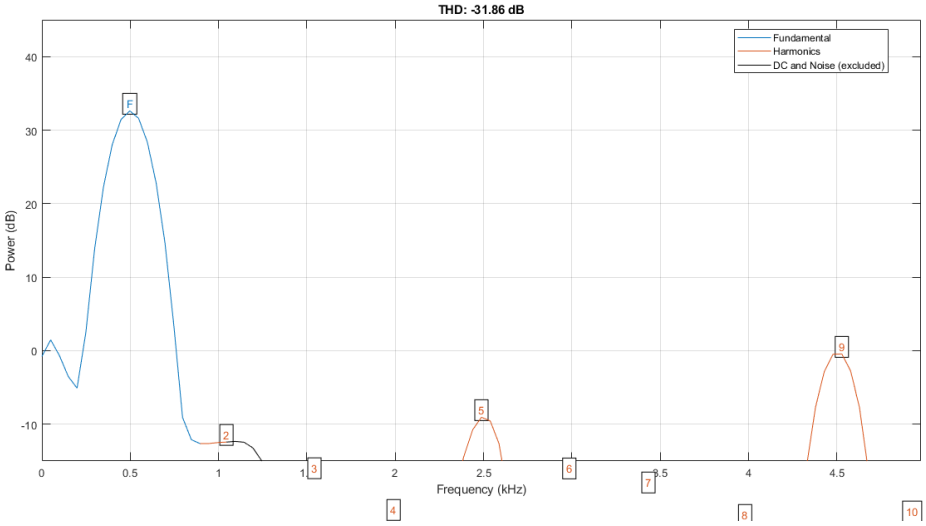

Figure 1.4: Matlab THD computation

This result is quite close to the constraint, but it can be considered a sufficient result. To improve this parameter, it would be useful to change the internal parallelism to achieve a better accuracy.
In figure 1.5 is reported a comparison between the output of the Matlab floating point FIR and the C fixed point FIR.

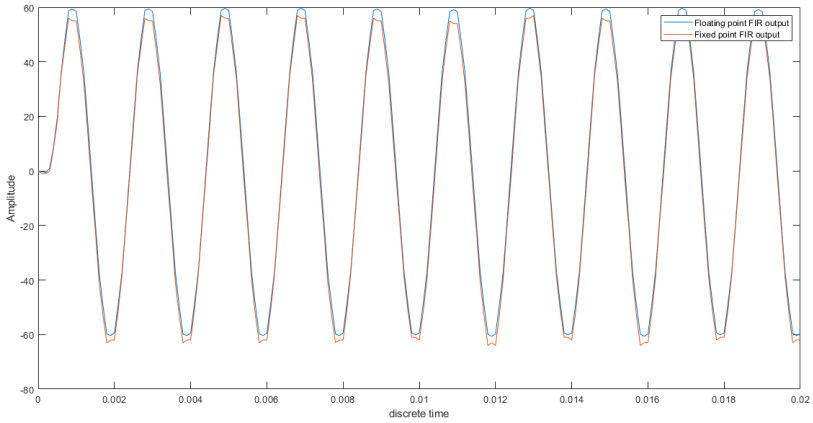

Figure 1.5: Comparison between Matlab and C FIR outputs

Some distortion can be observed in the fixed point FIR output due to the lower accuracy with respect to the floating point architecture. However, since the THD constraint is met, this result is considered acceptable.

The C script implemented in the code *myfilter.c* is the **model** of the system. The architecture described afterwards has to be perfectly coherent with it in terms of parallelism and produced results.

CHAPTER 2

First architecture development

2.1 Developement of the VHDL model of the filter and verification

2.1.1 VHDL model

The **contents** related to the following implementation are located **in the branch 'no_pipe'.**

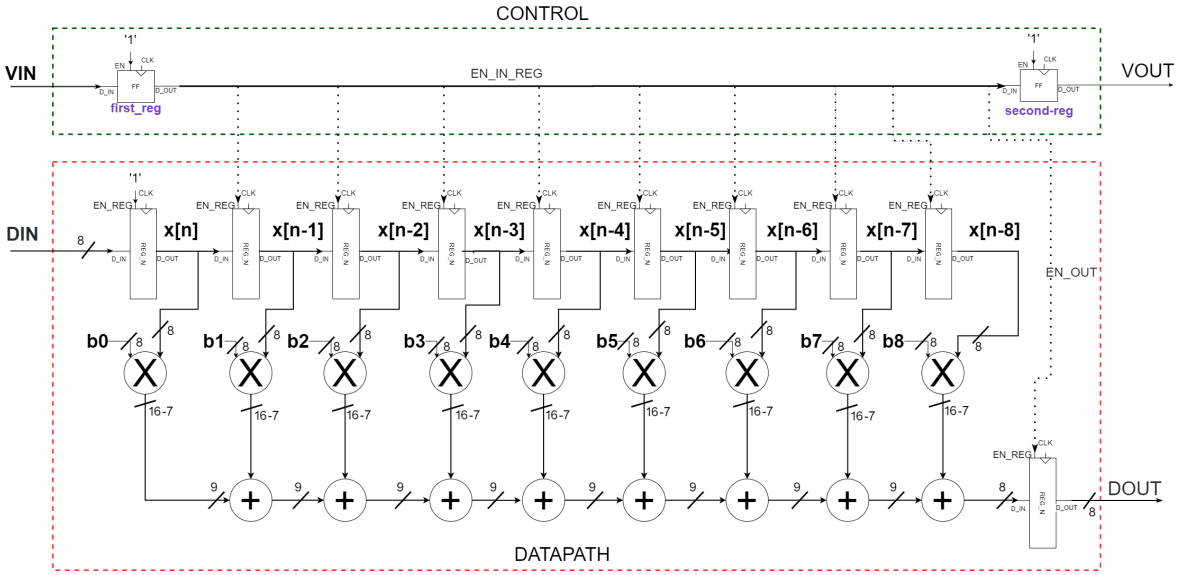

Figure 2.1: *DATAPATH of the implemented solution*

The design of the filter has been divided into 2 main parts: the control part and the datapath one. **The control part** manage the VIN input signal in order to assert properly the Enable signals of the internal registers and provide the VOUT signal when the data output is ready. Since it is a very simple structure it has been implemented using 2 flip-flop which properly delay the VIN input signal to generate the required internal signals.
The datapath part is composed by a shift register with 1 input and 9 output in order to collect the

$x[n], x[n-1]$, ... $, x[n-8]$ samples required to compute the output result:

$$y[n] = \sum_{i=0}^{8} b_i \, x[n-i] \tag{2.1}$$

Then, all the x values are multiplied for the correspondent coefficient and finally summed up to compute the output value. The computed value is saved in the output register and it will be available for 1 clock cycle, together with the VOUT signal generated by the control section.

The parallelism of the operation is implemented taking into account the C model.
Input data and coefficients are on 8 bits and their multiplication provide a result of 16 bits. From the C model we pointed out that discarding the least 7 significant bits the THD requirement is met, so only the 9 most significant bits are taken out the multiplier.
The outputs of the multiplier come into the cascade of adders shown in figure 2.1, and at the output of the last one the MSB in discarded, since the required ouput parallelism is of 8 bits. This reduces the output range possible values, so the inputs provided to the filter must satisfy the input range such that the outputs does not exceed it. Notice that is not true in the C model since the y value is an *int* variable that can assume greater values while granting a correct result. So we can conclude that the C model and the VHDL implementation produce identical results until the input range constraint is satisfied.

With the depicted structure the filter has no optimizations and the whole operations are carried on in 1 clock cycle. There is no latency and the output is available 1 clock cycle after the input data arrival.
In the timing diagram below depict the working behavior:

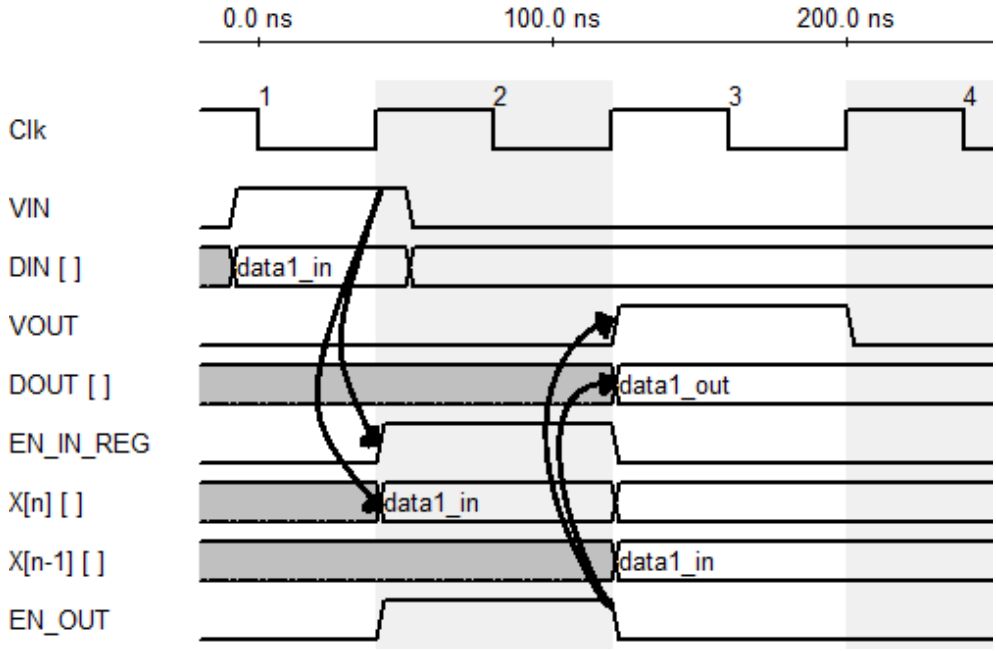

Figure 2.2: *Timing diagram of single data elaboration*

2.1.2 TestBench design and VHDL model verification

The testbench consist in two main blocks:

- **tb_data_maker**: it takes the input samples from the input file used also in the ideal C model and send them to the DUT managing properly the timing of VIN.
- **tb_output_checker**: it takes the output of the DUT whenever the VOUT signal is asserted and compare the value with the output file of the ideal model.

The connection are performed in the verilog testbench /VHDL/tb/TB_FILTER.v.
The resulting block scheme is:

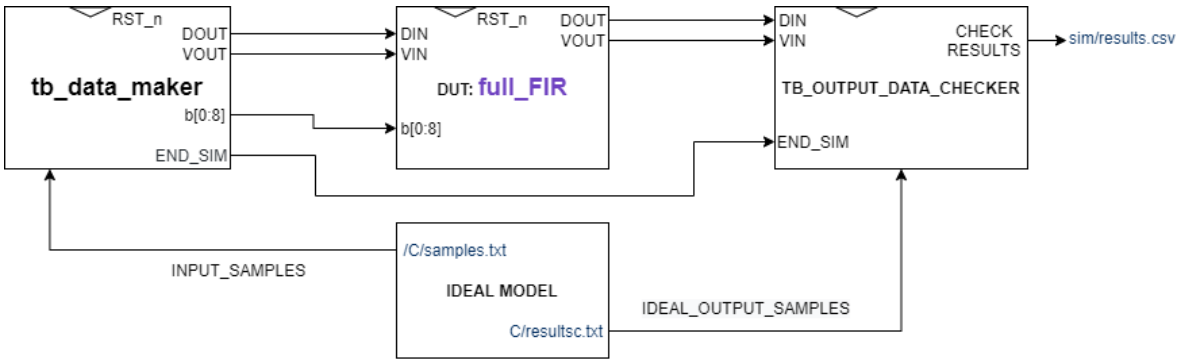

Figure 2.3: *BLOCK SCHEME of the Testbench*

The output of the simulation is a .csv file that report the comparison result of every output sample of the DUT.
The .csv file output format is: INPUT, OUTPUT, EXPECTED OUTPUT, TEST RESULT; so for each output sample we have a clear idea of which input generate it and what is the expected result. The TEST RESULT parameter is the result of the comparison between the obtained OUTPUT and the IDEAL OUTPUT of the C model.
Finally the output checker writes "SIMULATION ENDED SUCCESSFULLY" at the and of the .csv file if no output errors are detected.
Subsequently, the VHDL model (the *full_FIR* entity) is verified using the *Modelsim* tool. The following commands have been executed in order to compile files and run the simulation:

```
vcom -93 -work ./work ../src/coeffs.vhd
vcom -93 -work ./work ../src/pkg.vhd
vcom -93 -work ./work ../src/control.vhd
vcom -93 -work ./work ../src/full_FIR.vhd
vcom -93 -work ./work ../src/my_fir.vhd

vcom -93 -work ./work ../tb/clk_gen.vhd
vcom -93 -work ./work ../tb/tb_output_data_checker.vhd
vcom -93 -work ./work ../tb/tb_data_maker.vhd
vcom -93 -work ./work ../tb/pkg.vhd


vlog -work ./work ../tb/TB_FILTER.v

vsim work.tb_fir
```

The simulation brings out **no errors** since the results given are equal to the ones obtained with the C model. The comparison of the outputs, printed by Modelsim in the *results.csv* file, are shown in table 5.1.
The results of the VHDL are correct even when VIN moves from '1' to '0' and then back to '1' thanks to the "control part" of the architecture. This is shown in the simulation postscript below.

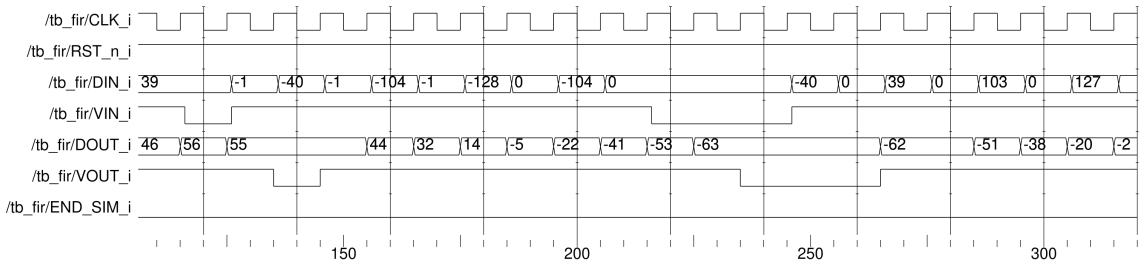

Figure 2.4: *Simulation of the VHDL model - wave diagram*

2.2 Logic synthesis

Synopsis software is used to synthesize the circuit starting from the VHDL design. The following steps are performed:

- loading of all the .vhd files in the tool and lunch the elaborate command to start Synopsys elaboration of our files;
- apply clock and delays constraints;
- start Synopsys compilation and collect result files.

To perform the listed steps the following command has been executed.

```
cd /home/isa37/Documents/lab1/VHDL/syn/
rm -r work
mkdir work
mkdir analysis_results
#*******Reading VHDL source files**************
analyze -f vhdl -lib WORK ../src/pkg.vhd
analyze -f vhdl -lib WORK ../src/control.vhd
analyze -f vhdl -lib WORK ../src/reg8.vhd
analyze -f vhdl -lib WORK ../src/full_FIR.vhd
analyze -f vhdl -lib WORK ../src/my_fir.vhd
#Set one parameter to preserve rtl names in the netlist.
set power_preserve_rtl_hier_names true
#Launch elaborate command to load the components
#elaborate <top entity name> -arch <architecture name> -lib WORK > ./elaborate.txt
elaborate full_FIR -arch struct -lib WORK > ./elaborate.txt
#uniquify #optional command to addres to only 1 specific architecture
link

#*******  Applying constraints   ***************
#create 100 Mhz clock
create_clock -name MY_CLK -period 0 CLK
```

```
set_dont_touch_network MY_CLK

#jitter simulation
set_clock_uncertainty 0.07 [get_clocks MY_CLK]

#input/output delay
set_input_delay 0.5 -max -clock MY_CLK [remove_from_collection [all_inputs] CLK]
set_output_delay 0.5 -max -clock MY_CLK [all_outputs]

#set output load (buffer x4 used)
set OLOAD [load_of NangateOpenCellLibrary/BUF_X4/A]
set_load $OLOAD [all_outputs]

#*********    Start the syntesis    *************
compile >  ./analysis_results/compilation_results.txt

#*********    Save the results      *************
report_timing  >  ./analysis_results/timing_results.txt
report_area    >  ./analysis_results/area_results.txt

#Finally, we can save the data required to complete the design and to perform
switchingactivity-based power estimation.
#First, we ungroup the cells to flatten the hierarchy as follows:
ungroup -all -flatten
#Then, we have to export the netlist in verilog. So that we impose verilog rules
for the names of the internal signals. This is obtained with
change_names -hierarchy -rules verilog
#We also save a file describing the delay of the netlist:
write_sdf ../netlist/myfir.sdf
#We can now save the netlist in verilog:
write -f verilog -hierarchy -output ../netlist/myfir.v
#and the constraints to the input and output ports in a standard format:
write_sdc ../netlist/myfir.sdc
```

2.2.1 Maximum clock frequency evaluation

In order to evaluate maximum clock frequency it has been performed a synthesis evaluation with Clock set to 0:

```
create\_clock -name MY\_CLK -period 0 CLK
```

This simulation enable us to evaluate the minimum clock period from the negative slack pointed out in the simulation results: -2.38 ns.In order to validate maximum clock frequency new synthesis has been performed with a clock slightly higher than 2.38 ns.The lower clock period in which the delay constraints are met is 2.5 ns, so **the maximum working frequency** of our circuit is 400 MHz.

2.2.2 Area evaluation

The **area** evaluated is 3012.72 μm^2.

2.3 Synthesis with $f_{clk} = f_{max}/4$

2.3.1 Area evaluation

To perform this required step it has been executed the procedure depicted in paragraph 2.2. In this case the clock constraints are different since it is required to set the clock period to $4 * T_{min}$,so it has been set to 10 ns:

```
create\_clock -name MY\_CLK -period 10 CLK
```

The result of the synthesis can be summarized into 2 elements:

- timing evaluation: SLACK = -7.38 ns
- evaluated area: 2975.74 μm^2

The occupied area is slightly the same of the synthesis with the minimum period clock, also the critical path period (2.62 ns) so the synthesizer does not further optimized the circuit design.

2.3.2 Design Verification

Synopsis tool creates also the verilog description of the synthesized circuit that use only the components taken from our library, this is made with the command:

```
write -f verilog -hierarchy -output ../netlist/myfir.v
```

The verilog file is our synthesized circuit and its functionality can be evaluated with the test bench we use for the VHDL design. To perform that evaluation we can use the script:

```
cd /home/isa37/git/lab1/VHDL/sim/
source /software/scripts/init_msim6.2g
rm -r work
vlib work
vcom -93 -work ./work ../tb/clk_gen.vhd
vcom -93 -work ./work ../tb/tb_output_data_checker.vhd
vcom -93 -work ./work ../tb/tb_data_maker.vhd

#Assuming Testbench file is tb_fir.v
#and testbench module is tb_fir
```

```
#Compile the verilog type:
vlog -work ./work ../netlist/myfir.v
vlog -work ./work ../tb/TB_FILTER.v

#link to Modelsim the compiled library of the cells
#vsim -L /software/dk/nangate45/verilog/msim6.2g work.tb_fir
#link the delay file
vsim -L /software/dk/nangate45/verilog/msim6.2g -sdftyp /tb_fir/UUT=../netlist/myfir.sdf work.tb_fir
```

This script will load to modelsim simulator the previous testbench using the synthesized circuit as DUT. Starting the simulation we point out the correct functionality of the circuit since **no errors** are printed in the *results.csv* file. The result of the simulation is the same as the one seen in table 5.1 of the Appendix.

2.3.3 Power consumption estimation

The power estimation is carried on using a value-change-dump (.vcd) file that record the switching activity of our simulation. This file is produced by modelsim software starting from the testbench of the syntetzed circuit, executing the following command before running the simulation:

```
#open the VCD file:
vcd file ../vcd/myfir_syn.vcd
#specify that we want to monitor all the signals inside our unit-under-test (the FIR filter):
vcd add /tb_fir/UUT/*
```

Finally the .vcd file can be analysed by Synopsys Design Compiler in order to print out the power consumption. First step is to convert the fil from .vcd to .saif format, using the following command from synopsis tool:

```
vcd2saif -input ../vcd/myfir_syn.vcd -output ../saif/myfir_syn.saif
```

Then start Synopsys and run the commands:

```
    #read the netlist
read_verilog -netlist ../netlist/myfir.v
#read the saif file generated by modelsim simulation
read_saif -input ../saif/myfir_syn.saif -instance tb_fir/UUT -unit ns -scale 1
#show the clock signal
create_clock -name MY_CLK CLK
report_power > power_report.txt
```

Executing this steps the Design Compiler is able to report the power consumption. For our synthesized circuit it is estimated to:

Total Dynamic Power	542.6726 μW
Cell Leakage Power	61.2101 μW

From the table printed out by the tool we can evaluate some information:

Report 2.1: Power estimation of the synthesized circuit

Power **Group** Attrs	Internal Power	Switching Power	Leakage Power	Total Power	(%)
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)
register	85.1749	38.9908	7.2374e+03	131.4032	(21.76%)
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)
combinational	211.5650	206.9420	5.3973e+04	472.4795	(78.24%)
Total	296.7399 uW	245.9328 uW	6.1210e+04 nW	603.8827 uW	

The compiled circuit is entirely composed by registers and combinational network, that reflect our VHDL design. It can be seen that most of power consumption come from the combinational network that compose the adders and the multipliers.

2.4 Place & Route

The Cadence's *Innovus* software is used to perform place and route operation of the system together with some files produced in the previous step. At first, a setup information file is imported into *Innovus*: it contains information about the paths based on the netlist synthesized by the design compiler (the *.v* file) and the constraints to the input and output ports (the *.sdc* file). This information are followed by commands that set the Standard Cell Library together with some fixed delays and the VDD and GND net name.

```
## CUSTOMIZE

set IN_DIR "../netlist"
set TopLevelDesign "full_FIR"
set in_verilog_filename "${IN_DIR}/myfir.v"
set in_sdc_filename "${IN_DIR}/myfir.sdc"

## DO NOT TOUCH

set LIB_DIR /software/dk/nangate45/liberty
set MyTimingLib ${LIB_DIR}/NangateOpenCellLibrary_typical_ecsm_nowlm.lib

set LEF_DIR /software/dk/nangate45/lef
set LEF_list [list ${LEF_DIR}/NangateOpenCellLibrary.lef]

set init_design_netlisttype "verilog"
set init_design_settop 1
set init_top_cell $TopLevelDesign
set init_verilog $in_verilog_filename

set init_lef_file "${LEF_list}"
```

```
set aspect_ratio 1.0
set target_row_utilization 0.6

set CustomDelayLimit 1000
set CustomNetDelay 1000.0ps
set CustomNetLoad 0.5pf
set CustomInputTranDelay 0.1ps

set MycapTable $LEF_DIR/captables/NCSU_FreePDK_45nm.capTbl

set init_gnd_net {VSS}
set init_pwr_net {VDD}
```

After the design import a series of step is followed, all of them reported below.

Floorplan structuring

In this step, the area of the cell ensemble is assigned: Core aspect ratio is set as 1.0, the utilization as 0.6, while the Core Margins are forced as *Core to die boundary, 5μm* from the four sides of the core.

Power Rings Insertion

The metal rings for power supply (VDD) and ground (VSS) are inserted.

Standard cell power routing

This operation allows to place horizontal wires needed to connect the VDD and VSS wires to the standard cells. The result of this step can be seen in figure 2.5.

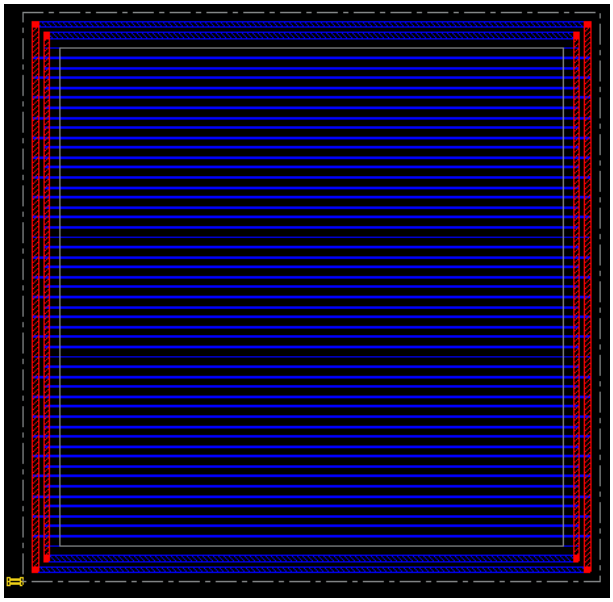

Figure 2.5: Vdd and Vss inserted rings

Placement

In this step the cells are placed in the floor planning.

Post Clock-Tree-Synthesis (CTS) optimization

It is based on trying to optimize the design in order to achieve the required timing constraints.

Filler Placement

This step is done to complete the placement with filler cells. It is important from a technological point of view because having placement with filler cell to ensure continuity in n+ and p+ wells implantation.

Routing

The connections among the cells are realized using the available metal layers. After starting the routing process, in the Innovus shell a few messages compare, some of them reported below:

```
    #Start Post Route Wire Spread.
#Done with 526 horizontal wires in 1 hboxes and 500 vertical wires in 1 hboxes.
#Complete Post Route Wire Spread.
#
#Total wire length = 14045 um.
#Total half perimeter of net bounding box = 14410 um.
#Total wire length on LAYER metal1 = 965 um.
#Total wire length on LAYER metal2 = 7232 um.
#Total wire length on LAYER metal3 = 5236 um.
#Total wire length on LAYER metal4 = 612 um.
#Total wire length on LAYER metal5 = 0 um.
#Total wire length on LAYER metal6 = 0 um.
#Total wire length on LAYER metal7 = 0 um.
#Total wire length on LAYER metal8 = 0 um.
#Total wire length on LAYER metal9 = 0 um.
#Total wire length on LAYER metal10 = 0 um.
#Total number of vias = 9223
#Up-Via Summary (total 9223):
#
#-----------------------
# metal1           5799
# metal2           3277
# metal3            147
#-----------------------
#                  9223
```

From this messages some interesting information can be derived. First of all, the **number of wires** used for **each layer** is: 526 horizontal wires and 500 vertical wires. These two numbers are equal also due to the square Core aspect ratio defined in the Floorplanning. The **total length routed in each layer** is reported in table 2.1.

Table 2.1: Total wire length on each layer

Layer	Length
metal 1	965 μm
metal 2	7232 μm
metal 3	5236 μm
metal 4	612 μm
metal 5 to 10	0 μm

The upper levels (5 to 10) lack of wires due to the relatively small number of interconnections, which do not need to occupy them.
Finally, we can identify the **number of vias used**, reported in the table 2.2.

Table 2.2: Number of vias used

Layer	Vias
metal 1	5799
metal 2	3277
metal 3	147
metal 4 to 10	0 μm

There are no vias used for metal 4 to 10 because there are no layers used above metal 4, so there's no need of connections between metal 4 and the upper layers.

Post routing optimization

In this last step the design is optimized again to achieve the required timing constraints. The final product of Place & Route is shown in figure 2.6, where all the cells and the interconnection can be observed.

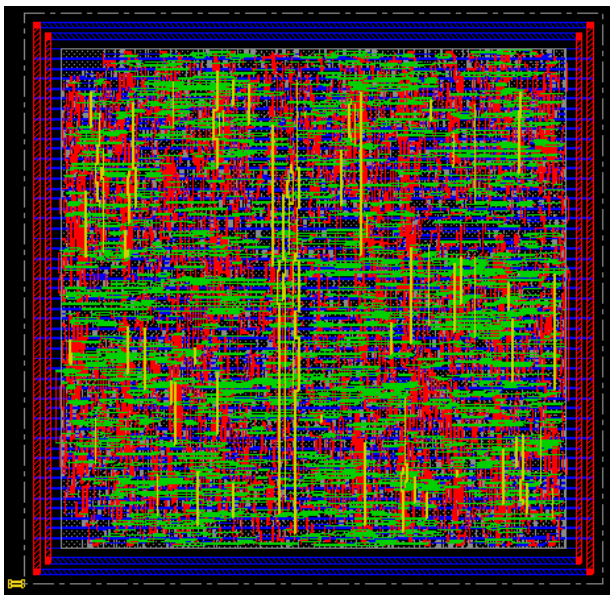

Figure 2.6: FIR layout in its first implementation

After the Place & Route phase some additional steps are performed in order to study the produced layout in terms of delay, area and, later, power.

2.4.1 Parasitics extraction

Here resistance and capacitance parasitic values are extracted for each wire. This information will be used by Innovus to analyze in more accurate way the time constraints. An offprint of the Innovus messages is showed below.

```
Initializing multi-corner capacitance tables ...
Initializing multi-corner resistance tables ...
Checking LVS Completed (CPU Time= 0:00:00.0  MEM= 1171.0M)
Extracted 10.0114% (CPU Time= 0:00:00.1  MEM= 1234.8M)
Extracted 20.0094% (CPU Time= 0:00:00.1  MEM= 1234.8M)
Extracted 30.0074% (CPU Time= 0:00:00.1  MEM= 1234.8M)
Extracted 40.0121% (CPU Time= 0:00:00.1  MEM= 1234.8M)
Extracted 50.01% (CPU Time= 0:00:00.1  MEM= 1234.8M)
Extracted 60.008% (CPU Time= 0:00:00.1  MEM= 1234.8M)
Extracted 70.0127% (CPU Time= 0:00:00.2  MEM= 1234.8M)
Extracted 80.0107% (CPU Time= 0:00:00.2  MEM= 1234.8M)
Extracted 90.0087% (CPU Time= 0:00:00.2  MEM= 1234.8M)
Extracted 100% (CPU Time= 0:00:00.3  MEM= 1234.8M)
Number of Extracted Resistors     : 24248
Number of Extracted Ground Cap.   : 26013
Number of Extracted Coupling Cap. : 33016
```

2.4.2 Timing analysis

It is based on verify the respect of the timing constraints. The clock period has been set by the design compiler in the previous part as $f_M/4$ (where f_M is the maximum operating clock frequency achievable by the system). After running the Timing Analysis and generate the reports, it has been verified that **the setup and hold requirements are met**: every slack computed **is positive**. A brief extract of setup and hold timing reports is shown below.

Report 2.2: Extract of Timing Report - setup

# Format: clock	timeReq	slackR/slackF	setupR/setupF	instName/pinName	# cycle(s)
MY_CLK(R)->MY_CLK(R)	9.894	7.797/*	0.036/*	FIR_DOUT_tmp_reg_7_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	7.869/*	0.036/*	FIR_DOUT_tmp_reg_6_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	7.940/*	0.036/*	FIR_DOUT_tmp_reg_5_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	8.011/*	0.036/*	FIR_DOUT_tmp_reg_4_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	8.083/*	0.036/*	FIR_DOUT_tmp_reg_3_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	8.155/*	0.036/*	FIR_DOUT_tmp_reg_2_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	8.228/*	0.036/*	FIR_DOUT_tmp_reg_1_/D	1
MY_CLK(R)->MY_CLK(R)	9.894	8.499/*	0.036/*	FIR_DOUT_tmp_reg_0_/D	1

Report 2.3: Extract of Timing Report - hold

# Format: clock	timeReq	slackR/slackF	holdR/holdF	instName/pinName	# cycle(s)
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_7_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_6_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_0_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_4_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_1_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_5_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_2_/D	1
MY_CLK(R)->MY_CLK(R)	0.073	*/0.006	*/-0.003	FIR_DOUT_tmp_reg_3_/D	1

2.4.3 Connectivity and Design rules verification

The last check that has to be made is the verification of connectivity and design rules. The **Connectivity** verification produces **no violations**, this means that there are no errors like floating wires.

```
******** Start: VERIFY CONNECTIVITY ********
Start Time: Thu Oct 15 16:21:09 2020

Design Name: full_FIR
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (81.1300, 80.0800)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Thu Oct 15 16:21:10 2020
Time Elapsed: 0:00:01.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.3  MEM: 0.000M)
```

The same thing holds for **Geometry**, this means that there are no errors like wrong constraints on the geometric feature during the place and route design flow.

```
 *** Starting Verify Geometry (MEM: 1118.1) ***

**WARN: (IMPVFG-257): verifyGeometry command is replaced by verify_drc command. [...]
  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 2160
VG: elapsed time: 4.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.
```

```
**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:01.0  MEM: 120.5M)
```

2.4.4 Area evaluation

Subsequently, the **area and gate count** are saved. The results are the following:

```
Gate area 0.7980 um^2
[0] full_FIR Gates=3702 Cells=1516 Area=2954.7 um^2
```

The computed Area of $2954.7\mu m^2$ is quite similar to the $2975.74\mu m^2$ evaluated by Synopsis: both of them are a reliable order of magnitude to estimate the area occupation.

Finally, the post place and route verilog netlist and the *.sdf* file with delay annotation are saved.

2.4.5 Modelsim simulation and swithing activity recording

To simulate the circuit taking into account the Place&Route operation, the produced netlist must be compiled. This operation is done with the following commands:

```
vcom -93 -work ./work ../tb/clk_gen.vhd
vcom -93 -work ./work ../tb/tb_output_data_checker.vhd
vcom -93 -work ./work ../tb/tb_data_maker.vhd

#compile the verilog

vlog -work ./work ../innovus/full_FIR.v
vlog -work ./work ../tb/TB_FILTER.v
```

Then the functional model of the cells in the technology library is included in the project:

```
vsim -L /software/dk/nangate45/verilog/msim6.2g work.tb_fir
```

To obtain an accurate switching activity report the delay file is linked:

```
vsim -L /software/dk/nangate45/verilog/msim6.2g -sdftyp /tb_fir/UUT=../innovus/full_FIR.sdf work.tb_fir
```

Before runnig the simulation a vcd file is opened. Here, switching information are written:

```
vcd file ../vcd/design.vcd
```

After running the simulation, the activity information are written in the `design.vcd` file. The simulation brings out **no errors**, all the results produced by the netlist generated by Innovus correspond exactly to the results of the C model (given the same input).

2.4.6 Power consumption estimation

In this final step the *.vcd* file written by Modelsim is used by Innovus to estimate the power consumption of the filter. After loading it in Innovus and running the power analysis, the power report is generated. An offprint of it can be see in report 2.4.

Report 2.4: Extract of the Power report produced by Innovus

```
*
*       Power Units = 1mW
*
*       Time Units = 1e-09 secs
```

```
*
*        report_power -outfile power_report_correct -sort total -hierarchy all -cell_type all -clock_network all
*
```

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.06607	0.02742	0.007005	0.1005	13.14
Macro	0	0	0	0	0
IO	0	0	0	0	0
Combinational	0.338	0.2734	0.05277	0.6642	86.86
Clock (Combinational)	0	0	0	0	0
Clock (Sequential)	0	0	0	0	0
Total	0.404	0.3009	0.05978	0.7647	100

The estimated Total power of $764.7\mu W$ is comparable to the $603.88\mu W$ computed by Synopsis. There are some cases where the power computed by Innovus is 3 times the power evaluated by Synopsis, but that's not our case. This is due to the simplicity of the filter.

CHAPTER 3

Advanced architecture development

3.1 Implementation of architecture techniques on the FIR

The contents related to the following implementation are located in the branch 'unfolding'. In order to improve the architecture of the FIR in terms of throughput, two techniques are exploited:

1. 3-Unfolding;
2. Pipelining.

3.1.1 Unfolding

Unfolding technique is applied to the precedent architecture through the analysis on the discrete time equations. Based on the equation of the FIR of order 8 (eq.3.1),

$$y[n] = \sum_{i=0}^{8} b_i \, x[n-i] \tag{3.1}$$

the three parallel architecture is evaluated through these three equations:

$$\begin{aligned} y[3k] &= \sum_{i=0}^{8} b_i \, x[3k-i] \\ &= b_0 \, x[3k] + b_1 \, x[3(k-1)+2] + b_2 \, x[3(k-1)+1] + b_3 \, x[3(k-1)] \\ &\quad + b_4 \, x[3(k-2)+2] + b_5 \, x[3(k-2)+1] + b_6 \, x[3(k-2)] \\ &\quad + b_7 \, x[3(k-3)+2] + b_8 \, x[3(k-3)+1] \end{aligned} \tag{3.2}$$

$$\begin{aligned} y[3k+1] &= \sum_{i=0}^{8} b_i \, x[3k+1-i] \\ &= b_0 \, x[3k+1] + b_1 \, x[3k] + b_2 \, x[3(k-1)+2] + b_3 \, x[3(k-1)+1] \\ &\quad + b_4 \, x[3(k-1)] + b_5 \, x[3(k-2)+2] + b_6 \, x[3(k-2)+1] \\ &\quad + b_7 \, x[3(k-2)] + b_8 \, x[3(k-3)+2] \end{aligned} \tag{3.3}$$

$$
\begin{aligned}
y[3k+2] &= \sum_{i=0}^{8} b_i \; x[3k+2-i] \\
&= b_0 \; x[3k+2] + b_1 \; x[3k+1] + b_2 \; x[3k] + b_3 \; x[3(k-1)+2] \\
&\quad + b_4 \; x[3(k-1)+1] + b_5 \; x[3(k-1)] + b_6 \; x[3(k-2)+2] \\
&\quad + b_7 \; x[3(k-2)+1] + b_8 \; x[3(k-2)]
\end{aligned}
\tag{3.4}
$$

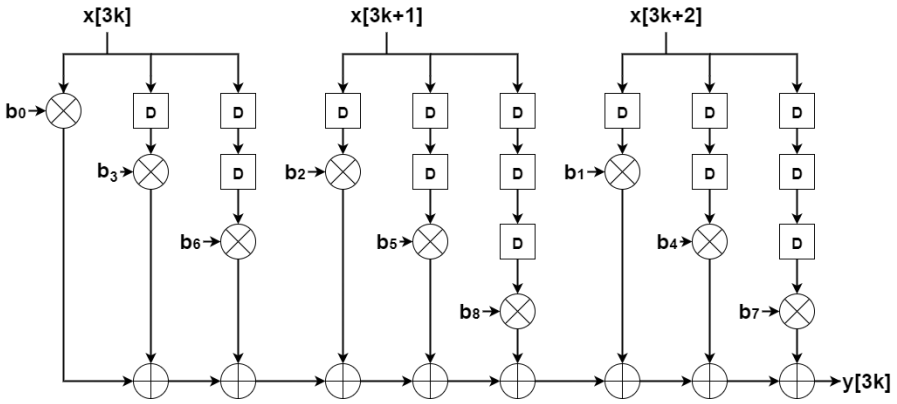

Figure 3.1: *DFG for y[3k]*

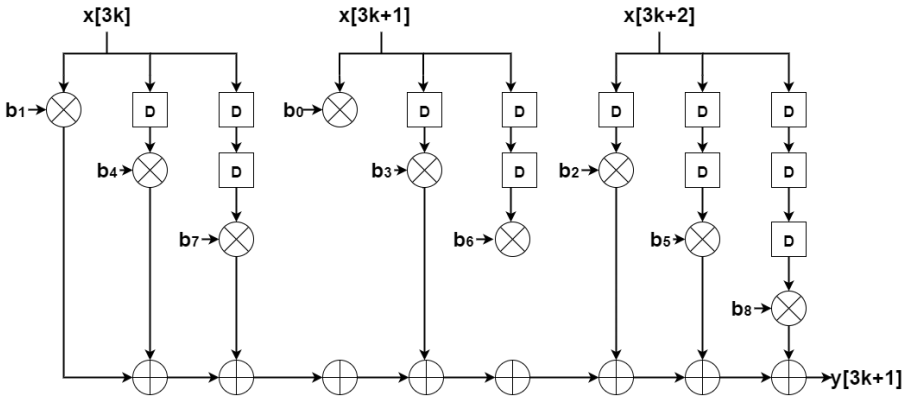

Figure 3.2: *DFG for y[3k+1]*

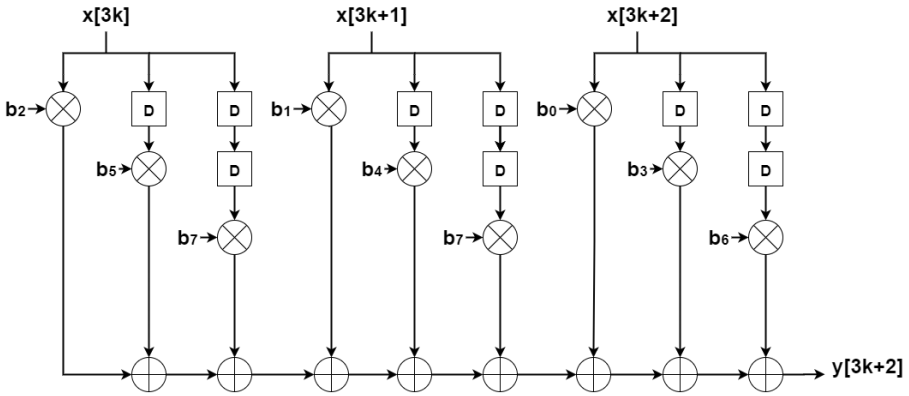

Figure 3.3: *DFG for y[3k+2]*

3.1.2 Pipelining

In order to have a more effective pipeline insertion a tree structure has been implemented to realize the sum operation instead of the chain structure. For example, considering the DFG for the evaluation of the output $y[3k]$, the tree structure DFG becomes the following one.

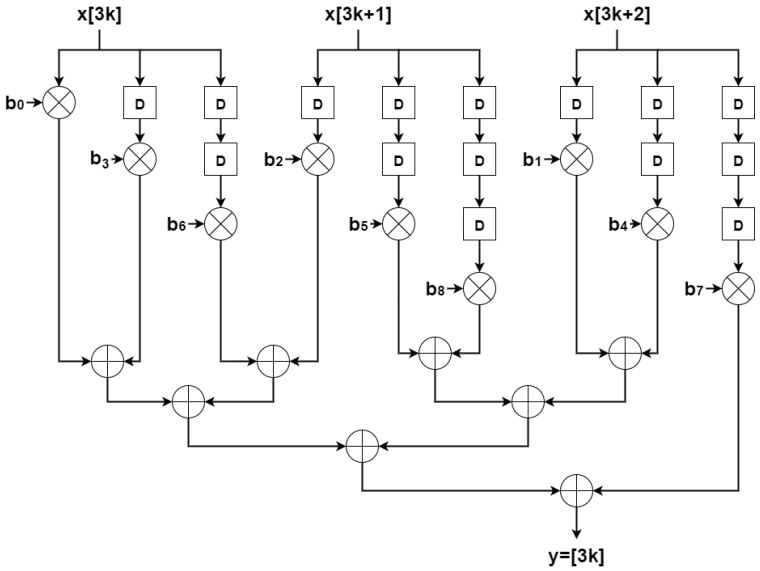

Figure 3.4: *DFG for y[3k] with tree structure for the sum operation*

Registers are inserted between the levels of adders to apply pipeline technique as shown in Fig.3.5.

Since we have considered a tree structure, the number of clock cycles required to implement the sum operation is reduced to four cycles.

3.2 Development of the VHDL model of the filter and verification

3.2.1 VHDL model

The VHDL implementation was done for the DFG in Fig.3.5.

The control signals (represented in red in the Fig.3.5) are generated with a proper delay of the received signal VIN. In especially, three signals are generated:

- **EN_INPUT_REG**: it enables the internal registers both used to delay the input signals and the pipe registers used for the sum operation.
- **EN_OUT**: it enables the register responsible of saving the result.
- **VOUT**: it is the output signal asserted when the result is ready.

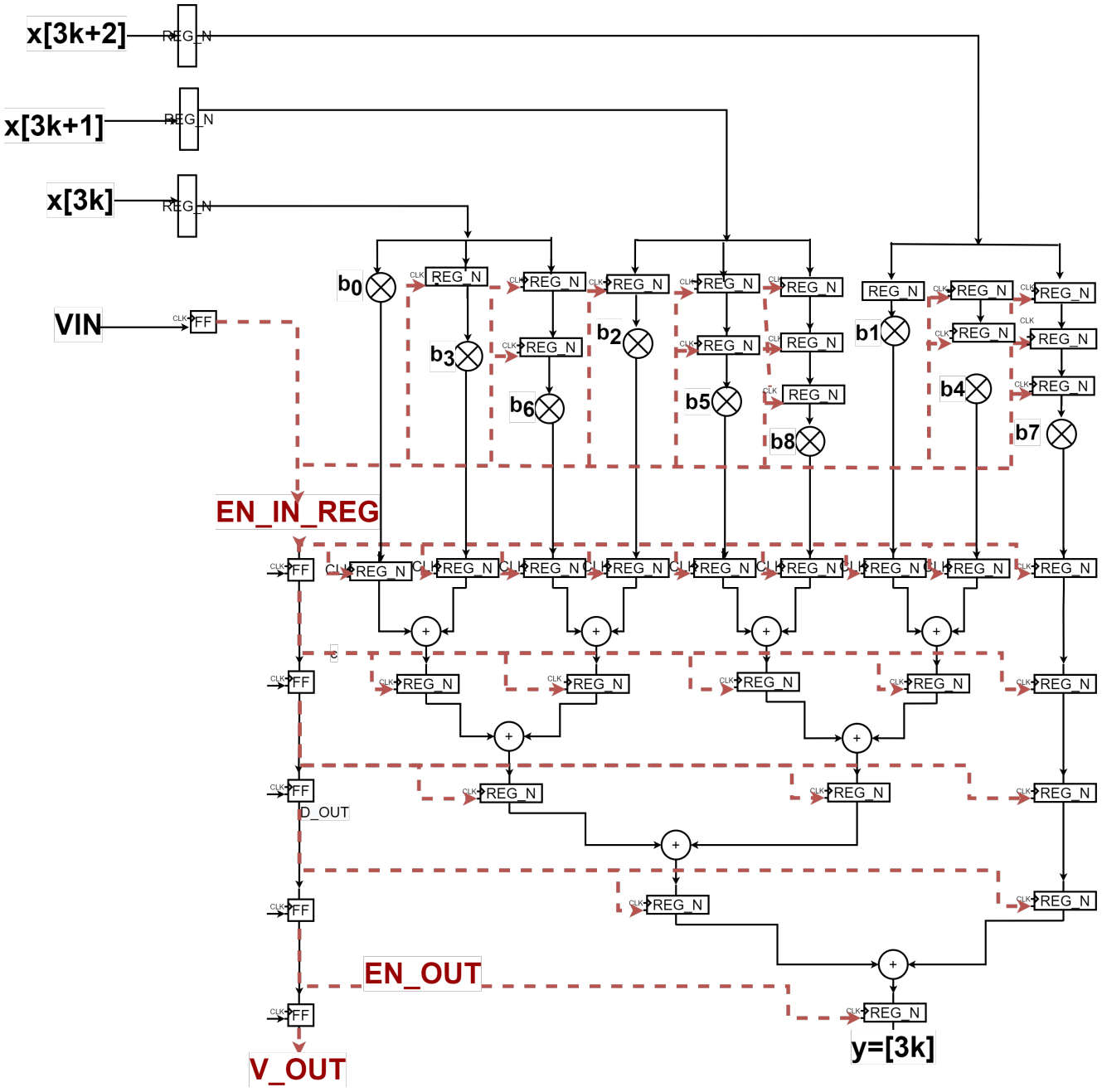

Figure 3.5: *Final DFG for y[3k]*

3.2.2 TestBench design and VHDL model verification

Since this advanced DUT requires 3 input values and generates 3 output values, the test bench must be modified to adapt *tb_data_maker* and *tb_output_data_checker* to its inputs/outputs interfaces.

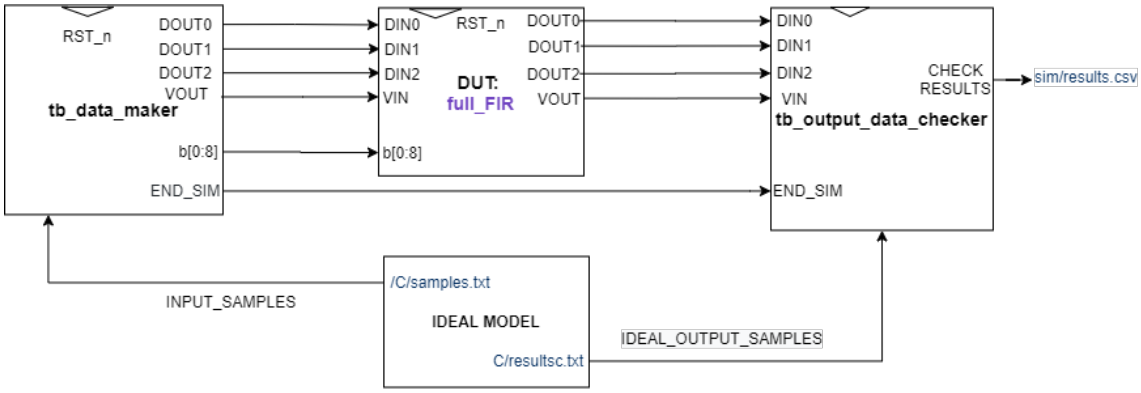

Figure 3.6: *TestBench Unfolding-Pipelined Architecture*

The Modelsim simulation of the DUT $full\_FIR$ is shown in fig.3.7.

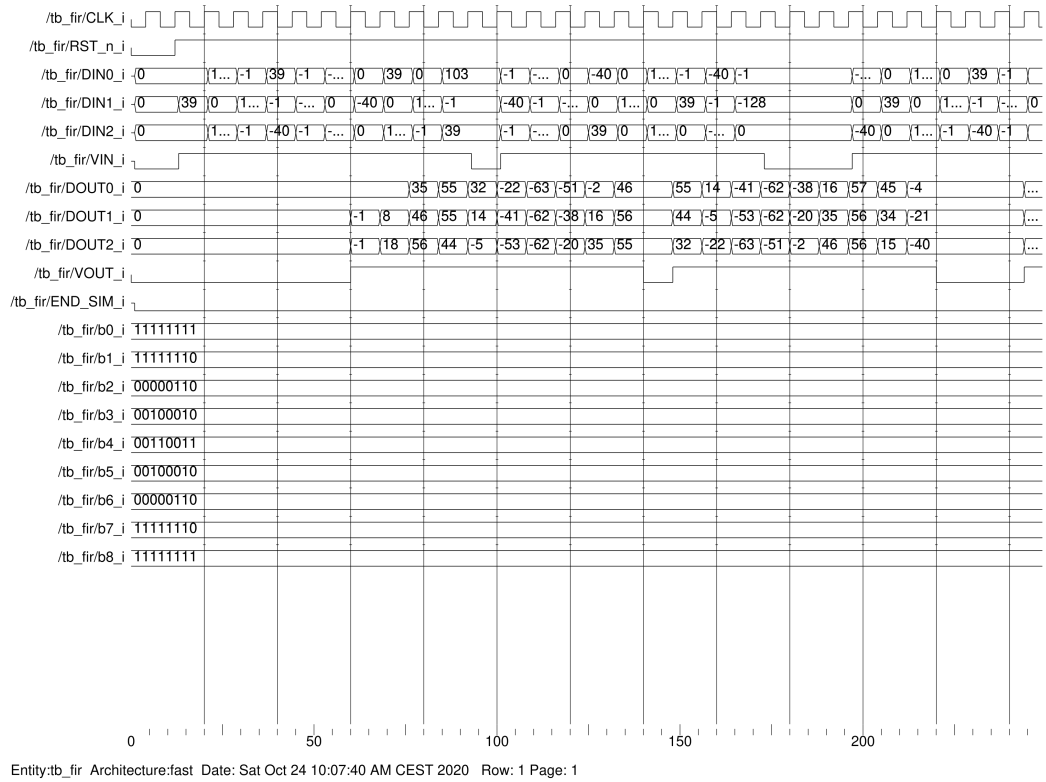

Figure 3.7: *Modelsim Simulation*

The timing diagrams shows the effect of the pipelining for which more than one clock cycle are

required to obtain the result but since the critical path is reduced we can reduce the duration of the clock cycle and then improve the throughput.

3.3 Logic synthesis

Following the same steps described in section 2.2, the logic synthesis of the actual design is obtained.

3.3.1 Max clock frequency evaluation

The minimum clock period obtained is $1.77ns$ (**max clock frequency** $f_{ck} \simeq 565MHz$), lower than the previous evaluation since the pipeline technique was applied and so the critical path is reduced.

3.3.2 Area evaluation

The **area** of the new architecture is $14964.36\mu m^2$.

3.4 Synthesis with $f_{clk} = f_{max}/4$

The synthesis is performed considering a clock period of $T_{ck} = 4T_{min} = 7.08ns$ (clock frequency $f_{ck} = f_M/4 = 141MHz$)

3.4.1 Area evaluation

In this case the area is slightly reduced to $13759.38\mu m^2$

3.4.2 Design Verification

Using Modelsim, it is possible to simulate the previous testbench using the synthesized circuit as DUT in order to evaluate the correct functionality of the advanced architecture. Analyzing the *csv* file values (reported in the par.5.2 of the Appendix), it is possible to observe that the actual implementation gives the same result of the C model.

3.4.3 Power consumption estimation

The **power consumption** is estimated. It is expressed by these two contributes:

Total Dynamic Power	2.373 mW
Cell Leakage Power	281.5654 μW

Since a more complex structure is involved, the actual implementation brings to a very high dynamic and leakage consumption. Evaluating the power consumption of the different components it is possible to observe that, differently from the last architecture, in this case the power consumption due to the registers became more relevant as expected because of the pipeline insertion.

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)	
register	872.3649	192.2681	7.6428e+04	1.1411e+03	(43.00%)	
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	688.2673	619.3855	2.0514e+05	1.5128e+03	(57.00%)	
Total	1.5606e+03 uW	811.6536 uW	2.8157e+05 nW	2.6538e+03 uW		

1

Figure 3.8: *Power estimation*

3.5 Place & Route

Following the same steps described in section 2.4, it was possible realize the layout of the actual design.

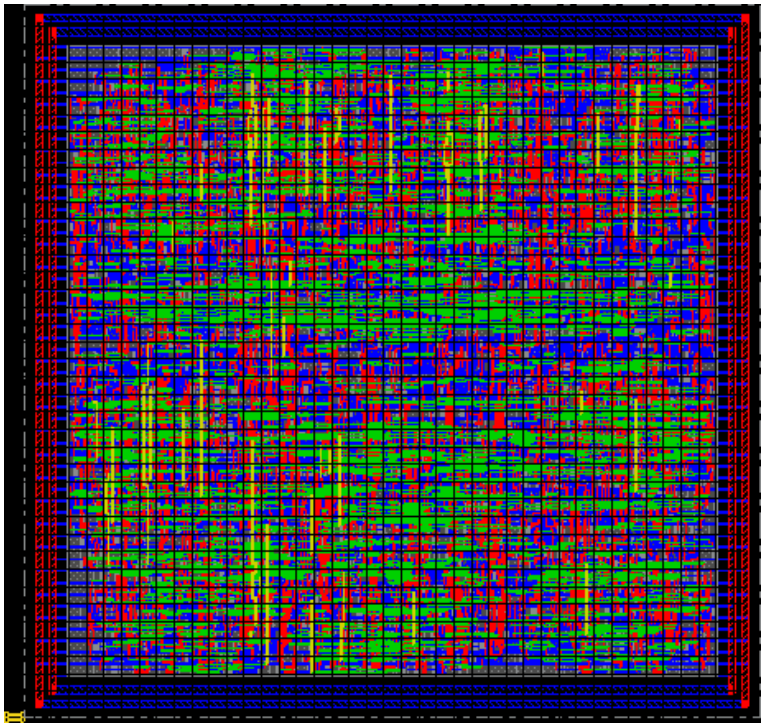

Figure 3.9: *Advanced FIR layout*

3.5.1 Parasitics extraction

The **extraction of the parasitic RC values** give this result:

Report 3.1: Extraction of the RC values

```
Number of Extracted Resistors      : 106321
Number of Extracted Ground Cap.    : 114380
Number of Extracted Coupling Cap.  : 146004
```

3.5.2 Timing Analysis

Setting the clock as $f_{ck} = f_M/4$, time constrains have been verified checking that slack time is positive and the setup, hold requirements are **met** for each of the paths.

Report 3.2: Extraction of Timing Report setup

```
# Format: clock   timeReq   slackR/slackF   setupR/setupF   instName/pinName   # cycle(s)
MY_CLK(R)->MY_CLK(R)     6.963     */4.541          */0.047          FIR_y3k_3_pipe1_reg5_Q_reg_8_/D
1
MY_CLK(R)->MY_CLK(R)     6.963     */4.543          */0.047          FIR_y3k_3_pipe1_reg5_Q_reg_5_/D
1
MY_CLK(R)->MY_CLK(R)     6.963     */4.545          */0.047          FIR_y3k_3_pipe1_reg5_Q_reg_6_/D
1
MY_CLK(R)->MY_CLK(R)     6.963     */4.546          */0.047          FIR_y3k_3_pipe1_reg5_Q_reg_7_/D
1
MY_CLK(R)->MY_CLK(R)     6.963     */4.551          */0.047          FIR_y3k_3_pipe1_reg5_Q_reg_4_/D
1
MY_CLK(R)->MY_CLK(R)     6.964     */4.571          */0.046          FIR_y3k2_0_SR3_REG_1_Q_reg_1_/D
1
MY_CLK(R)->MY_CLK(R)     6.964     */4.573          */0.046          FIR_y3k2_3_pipe1_reg2_Q_reg_3_/D
1
MY_CLK(R)->MY_CLK(R)     6.964     */4.573          */0.046          FIR_y3k2_3_pipe1_reg2_Q_reg_1_/D
1
MY_CLK(R)->MY_CLK(R)     6.964     */4.573          */0.046          FIR_y3k2_3_pipe1_reg2_Q_reg_0_/D
1
```

Report 3.3: Extraction of Timing Report hold

```
# Format: clock   timeReq   slackR/slackF   holdR/holdF   instName/pinName   # cycle(s)
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k2_3_pipe3_reg_1_Q_reg_1_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k2_2_SR3_REG_1_Q_reg_1_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k1_3_pipe4_reg_Q_reg_2_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k2_3_pipe4_reg_Q_reg_7_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k_3_pipe3_reg_0_Q_reg_6_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k2_3_saving_Q_reg_3_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k1_3_pipe2_reg_3_Q_reg_7_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k2_2_REG2_Q_reg_1_/D
1
MY_CLK(R)->MY_CLK(R)     0.073     */0.006          */-0.003         FIR_y3k1_3_last_prod_reg_2_Q_reg_6_/D
1
```

3.5.3 Connectivity and Design rules verification

The **Connectivity verification** proves that there are not floating wires. As shown in the following report, the design produces **no violations**.

```
******** Start: VERIFY CONNECTIVITY ********
Start Time: Sat Oct 24 17:24:41 2020

Design Name: full_FIR
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (162.0700, 161.2800)
```

```
Error Limit = 1000; Warning Limit = 50
Check all nets
**** 17:24:45 **** Processed 5000 nets.

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Sat Oct 24 17:24:52 2020
Time Elapsed: 0:00:11.0

******** End: VERIFY CONNECTIVITY ********
```

The **Geometry verification** proves the respect of constraints on the geometric feature during the place and route design flow.As shown in the following report, the design produces **no errors**.

```
 *** Starting Verify Geometry (MEM: 1193.2) ***

**WARN: (IMPVFG-257):    verifyGeometry command is replaced by verify_drc command. It still works in this release
  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 2160
VG: elapsed time: 25.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
```

3.5.4 Area evaluation

The area is 13545μm^2, which is quite similar to the one evaluated by Synopsis also for this last implementation.

Report 3.4: Extract of Area and Gate Count report produced by Innovus

```
Gate area 0.7980 um^2
[0] full_FIR Gates=16973 Cells=6490 Area=13545.0 um^2
```

3.5.5 Modelsim simulation and switching activity recording

To simulate the circuit taking into account the Place&Route operation, the produced netlist must be compiled. After running the simulation, the activity information are written in the `design.vcd` file. The simulation brings out **no errors**, all the results produced by the netlist generated by Innovus correspond exactly to the results of the C model (given the same inputs).

3.5.6 Power consumption estimation

Report 3.5: Extract of Power report produced by Innovus

```
*        Power Units = 1mW
*
*        Time Units = 1e-09 secs
*
*        report_power -outfile power_report_chip_placeandrouted.txt -sort total -hierarchy all -cell_type all -clo
*
```

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	1.335	0.3465	0.07426	1.756	37.29
Macro	0	0	0	0	0
IO	0	0	0	0	0
Combinational	1.563	1.197	0.1936	2.953	62.71
Clock (Combinational)	0	0	0	0	0
Clock (Sequential)	0	0	0	0	0
Total	2.898	1.544	0.2679	4.709	100

Differently from the first implementation, in this case the power consumption evaluated after the Place & Route is doubled than the Synopsis one. The reason is related to the increased complexity of the circuit.

CHAPTER 4

Comparison of the two architectures

In the following table some comparisons are made between the "basic" implementation of the FIR seen in chapter 2 and the "advanced" implementation seen in chapter 3 . The parameters considered are:

- Minimum **clock period** T_{min}, related to the Maximum **clock frequency** $f_{max} = 1/T_{min}$.
- **Throughput**, defined as the number of samples processed per unit time.
- Estimated **area**, according to *Innovus*.
- Estimated **power** consumption, according to *Innovus*.

In the last column the ratio between the same values for the two architecture is reported. This is useful to evaluate improvements and eventual drawbacks brought by the application of pipelining and unfolding.

	"basic" FIR	"advanced" FIR	ratio *advanced/basic*
T_{clk}	2.5 ns	1.67 ns	0.67
f_{clk}	400 MHz	598 MHz	1.5
th	$400 \cdot 10^6 s^{-1}$	$1.79 \cdot 10^9 s^{-1}$	4.5
Area	2954.7 μm^2	13545 μm^2	5.22
POWER			
Leakage	59.78 μW	267.9 μW	4.48
Dynamic	704.9 μW	4.442 mW	6.3
Total	764.7 μW	4.709 mW	6.17

Table 4.1: Comparison between the "basic" and "advanced" architectures

What can be observed by the table 4.1 is that the application of pipelining and unfolding has improved the throughput by a factor of 4.5. This is due to the reduction of the critical path by means of pipelining (of a factor 1/1.5) together with the triplication of the outputs provided at a time (factor 3) thanks to 3-Unfolding.
As drawback, the area of the system increases by a factor 5.22, and also the power consumption grows (by a factor 6.17). This result could be considered acceptable or not depending on the constraints: if the target is speed at any cost one could prefer the "advanced" option. On the other hand the "basic" option grants reduced power consumption ad area: it could be advisable in "low cost" applications.

CHAPTER 5

Appendix

5.1 Results of the simulation compared to the C model output

INPUT	OUTPUT	EXPECTED OUTPUT	TEST RESULT
0	0	0	OK
39	-1	-1	OK
0	-1	-1	OK
103	0	0	OK
0	8	8	OK
127	18	18	OK
-1	35	35	OK
103	46	46	OK
-1	56	56	OK
39	55	55	OK
-1	55	55	OK
-40	44	44	OK
-1	32	32	OK
-104	14	14	OK
-1	-5	-5	OK
-128	-22	-22	OK
0	-41	-41	OK
-104	-53	-53	OK
0	-63	-63	OK
-40	-62	-62	OK
0	-62	-62	OK
39	-51	-51	OK
0	-38	-38	OK
103	-20	-20	OK
0	-2	-2	OK
127	16	16	OK
-1	35	35	OK
103	46	46	OK
-1	56	56	OK

39	55	55	OK
-1	55	55	OK
-40	44	44	OK
-1	32	32	OK
-104	14	14	OK
-1	-5	-5	OK
-128	-22	-22	OK
0	-41	-41	OK
-104	-53	-53	OK
0	-63	-63	OK
-40	-62	-62	OK
0	-62	-62	OK
39	-51	-51	OK
0	-38	-38	OK
103	-20	-20	OK
0	-2	-2	OK
127	16	16	OK
0	35	35	OK
103	46	46	OK
-1	57	57	OK
39	56	56	OK
0	56	56	OK
-40	45	45	OK
-1	34	34	OK
-104	15	15	OK
-1	-4	-4	OK
-128	-21	-21	OK
0	-40	-40	OK
-104	-53	-53	OK
0	-63	-63	OK
-40	-62	-62	OK
0	-62	-62	OK
39	-51	-51	OK
0	-38	-38	OK
103	-20	-20	OK
0	-2	-2	OK
127	16	16	OK
0	35	35	OK
103	46	46	OK
-1	57	57	OK
39	56	56	OK
-1	56	56	OK
-40	45	45	OK
-1	33	33	OK
-104	14	14	OK
-1	-5	-5	OK
-128	-22	-22	OK

0	-41	-41	OK
-104	-53	-53	OK
0	-63	-63	OK
-40	-62	-62	OK
0	-62	-62	OK
39	-51	-51	OK
0	-38	-38	OK
103	-20	-20	OK
0	-2	-2	OK
127	16	16	OK
-1	35	35	OK
103	46	46	OK
-1	56	56	OK
39	55	55	OK
-1	55	55	OK
-40	44	44	OK
0	32	32	OK
-104	14	14	OK
0	-4	-4	OK
-128	-21	-21	OK
0	-39	-39	OK
-104	-51	-51	OK
-1	-61	-61	OK
-40	-61	-61	OK
-1	-62	-62	OK
39	-52	-52	OK
0	-40	-40	OK
103	-22	-22	OK
-1	-4	-4	OK
127	15	15	OK
-1	33	33	OK
103	45	45	OK
-1	55	55	OK
39	54	54	OK
-1	54	54	OK
-40	44	44	OK
-1	32	32	OK
-104	14	14	OK
-1	-5	-5	OK
-128	-22	-22	OK
-1	-41	-41	OK
-104	-53	-53	OK
-1	-64	-64	OK
-40	-63	-63	OK
0	-64	-64	OK
39	-53	-53	OK
-1	-40	-40	OK

103	-21	-21	OK
0	-4	-4	OK
127	15	15	OK
0	34	34	OK
103	45	45	OK
0	56	56	OK
39	56	56	OK
-1	57	57	OK
-40	46	46	OK
0	34	34	OK
-104	15	15	OK
0	-3	-3	OK
-128	-21	-21	OK
0	-39	-39	OK
-104	-51	-51	OK
-1	-61	-61	OK
-40	-61	-61	OK
0	-62	-62	OK
39	-52	-52	OK
0	-39	-39	OK
103	-21	-21	OK
0	-3	-3	OK
127	16	16	OK
-1	35	35	OK
103	46	46	OK
-1	56	56	OK
39	55	55	OK
0	55	55	OK
-40	44	44	OK
-1	33	33	OK
-104	15	15	OK
-1	-4	-4	OK
-128	-21	-21	OK
-1	-40	-40	OK
-104	-53	-53	OK
0	-64	-64	OK
-40	-63	-63	OK
0	-63	-63	OK
39	-52	-52	OK
-1	-39	-39	OK
103	-20	-20	OK
0	-3	-3	OK
127	15	15	OK
0	34	34	OK
103	45	45	OK
0	56	56	OK
39	56	56	OK

-1	57	57	OK
-40	46	46	OK
-1	34	34	OK
-104	15	15	OK
0	-4	-4	OK
-128	-22	-22	OK
0	-40	-40	OK
-104	-52	-52	OK
0	-62	-62	OK
-40	-61	-61	OK
0	-61	-61	OK
39	-51	-51	OK
0	-38	-38	OK
103	-20	-20	OK
-1	-2	-2	OK
127	16	16	OK
-1	34	34	OK
103	45	45	OK
0	55	55	OK
39	54	54	OK
0	55	55	OK
-40	45	45	OK
-1	34	34	OK
-104	16	16	OK
0	-3	-3	OK
-128	-21	-21	OK
-1	-39	-39	OK
-104	-52	-52	OK
0	-63	-63	OK
-40	-62	-62	OK
-1	-62	-62	OK

Table 5.1: Results of the simulation compared to the C model output

5.2 Results of the unfolded FIR simulation compared to the C model output

INPUT0	INPUT1	INPUT2	OUTPUT0	OUTPUT1	OUTPUT2	EXP.OUT0	EXP.OUT1	EXP.OUT2	RES.
0	39	0	0	-1	-1	0	-1	-1	OK
103	0	127	0	8	18	0	8	18	OK
-1	103	-1	35	46	56	35	46	56	OK
39	-1	-40	55	55	44	55	55	44	OK
-1	-104	-1	32	14	-5	32	14	-5	OK
-128	0	-104	-22	-41	-53	-22	-41	-53	OK
0	-40	0	-63	-62	-62	-63	-62	-62	OK

39	0	103	-51	-38	-20	-51	-38	-20	OK
0	127	-1	-2	16	35	-2	16	35	OK
103	-1	39	46	56	55	46	56	55	OK
-1	-40	-1	55	44	32	55	44	32	OK
-104	-1	-128	14	-5	-22	14	-5	-22	OK
0	-104	0	-41	-53	-63	-41	-53	-63	OK
-40	0	39	-62	-62	-51	-62	-62	-51	OK
0	103	0	-38	-20	-2	-38	-20	-2	OK
127	0	103	16	35	46	16	35	46	OK
-1	39	0	57	56	56	57	56	56	OK
-40	-1	-104	45	34	15	45	34	15	OK
-1	-128	0	-4	-21	-40	-4	-21	-40	OK
-104	0	-40	-53	-63	-62	-53	-63	-62	OK
0	39	0	-62	-51	-38	-62	-51	-38	OK
103	0	127	-20	-2	16	-20	-2	16	OK
0	103	-1	35	46	57	35	46	57	OK
39	-1	-40	56	56	45	56	56	45	OK
-1	-104	-1	33	14	-5	33	14	-5	OK
-128	0	-104	-22	-41	-53	-22	-41	-53	OK
0	-40	0	-63	-62	-62	-63	-62	-62	OK
39	0	103	-51	-38	-20	-51	-38	-20	OK
0	127	-1	-2	16	35	-2	16	35	OK
103	-1	39	46	56	55	46	56	55	OK
-1	-40	0	55	44	32	55	44	32	OK
-104	0	-128	14	-4	-21	14	-4	-21	OK
0	-104	-1	-39	-51	-61	-39	-51	-61	OK
-40	-1	39	-61	-62	-52	-61	-62	-52	OK
0	103	-1	-40	-22	-4	-40	-22	-4	OK
127	-1	103	15	33	45	15	33	45	OK
-1	39	-1	55	54	54	55	54	54	OK
-40	-1	-104	44	32	14	44	32	14	OK
-1	-128	-1	-5	-22	-41	-5	-22	-41	OK
-104	-1	-40	-53	-64	-63	-53	-64	-63	OK
0	39	-1	-64	-53	-40	-64	-53	-40	OK
103	0	127	-21	-4	15	-21	-4	15	OK
0	103	0	34	45	56	34	45	56	OK
39	-1	-40	56	57	46	56	57	46	OK
0	-104	0	34	15	-3	34	15	-3	OK
-128	0	-104	-21	-39	-51	-21	-39	-51	OK
-1	-40	0	-61	-61	-62	-61	-61	-62	OK
39	0	103	-52	-39	-21	-52	-39	-21	OK
0	127	-1	-3	16	35	-3	16	35	OK
103	-1	39	46	56	55	46	56	55	OK
0	-40	-1	55	44	33	55	44	33	OK
-104	-1	-128	15	-4	-21	15	-4	-21	OK
-1	-104	0	-40	-53	-64	-40	-53	-64	OK
-40	0	39	-63	-63	-52	-63	-63	-52	OK

-1	103	0	-39	-20	-3	-39	-20	-3	OK
127	0	103	15	34	45	15	34	45	OK
0	39	-1	56	56	57	56	56	57	OK
-40	-1	-104	46	34	15	46	34	15	OK
0	-128	0	-4	-22	-40	-4	-22	-40	OK
-104	0	-40	-52	-62	-61	-52	-62	-61	OK
0	39	0	-61	-51	-38	-61	-51	-38	OK
103	-1	127	-20	-2	16	-20	-2	16	OK
-1	103	0	34	45	55	34	45	55	OK
39	0	-40	54	55	45	54	55	45	OK
-1	-104	0	34	16	-3	34	16	-3	OK
-128	-1	-104	-21	-39	-52	-21	-39	-52	OK
0	-40	-1	-63	-62	-62	-63	-62	-62	OK

Table 5.2: Results of the unfolded FIR simulation compared to the C model output