Politecnico di Torino

Integrated Systems Architecture

A.A. 2020/2021

Prof. Guido Masera
Prof. Maurizio Martina

Lab 1: Digital Filter

19 Nov 2020

Antona Gaspare	275336
Barrera Alessandro	275337
Liu Huicai	273375

Link for GitHub: `https://github.com/Dragosk97/ISA-laboratories.git`

Contents

1 Reference model development

The aim of the laboratory is to design a Low-Pass FIR filter with:

- a cut-off frequency, $f_c = 2\,kHz$;
- a sample frequency, $f_s = 10\,kHz$.

The order chosen is $N = 8$ and the number of bits is 8. The coefficients of the FIR filter are obtained using a MATLAB function, then they are quantized to 8 bits, which introduces an error to the filter respect to the original one. The transfer function of the two filters are shown in figure 1. Their behaviours are quite different, especially at high frequencies, however the important thing is that the cut-off frequency is at $2\,kHz$ as required.

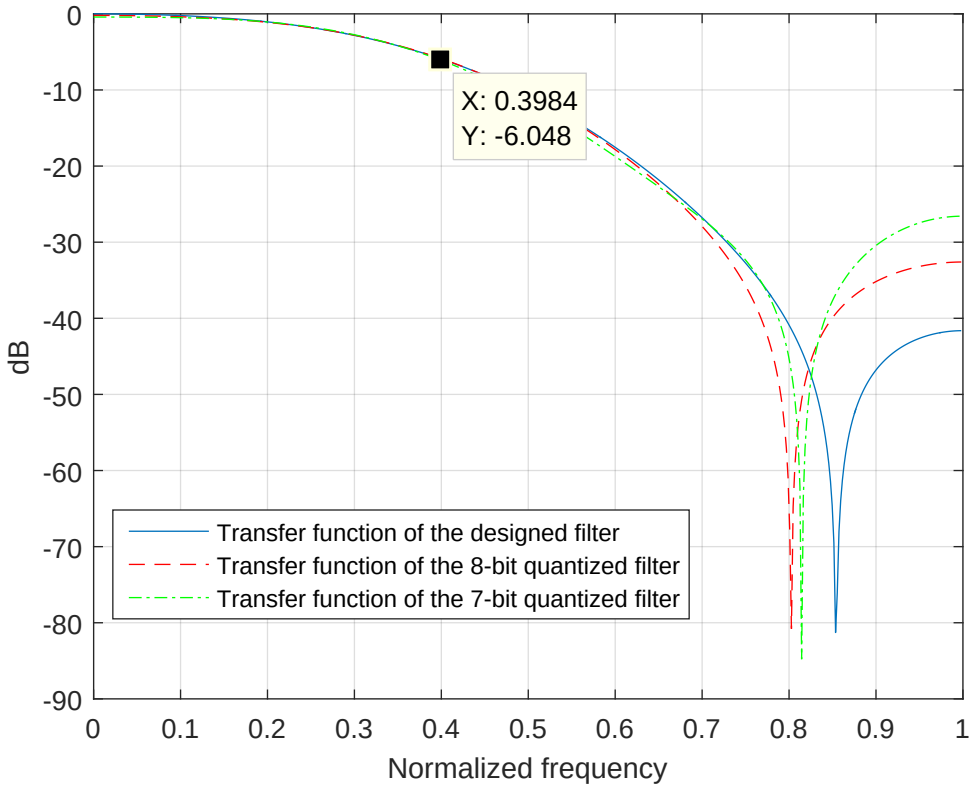

Figure 1: Transfer function of the FIR filter

The sum of two sinusoids at frequencies of 500 Hz and 4.5 kHz, in-band and out-band respectively, is taken as input of the digital filter. Both the input samples and the output values are saved in *.txt* files.

To simulate the correct behaviour of the digital filter to be designed, it is needed a fixed point model. For this purpose a C language program is exploited, which is fed

with the input file generated by MATLAB and the outputs are again saved in a *.txt* file.

Then the THD, total harmonic distortion, is evaluated for both the outputs of MATLAB and C program, with the goal of a maximum value equal to -30 dBc. Since the THD for the number of bits equals to 8 is much smaller than the requirement, a way of optimization is to size the internal bit representation of the filter in order to minimize the area of the architecture.

The values of THD are obtained with the *thd* function of MATLAB, and it has been done for a few values of number of bits. All the results are shown in the table 1. With the graphs in figure 2 it is also possible to see the values of THD for 7 bits.

Number of bits	THD - Matlab	THD - C program
8	-47.1325 dBc	-38.8026 dBc
7	-38.2542 dBc	-31.9525 dBc
6	-32.09 dBc	-23.5756 dBc

Table 1: Total Harmonic Distortion

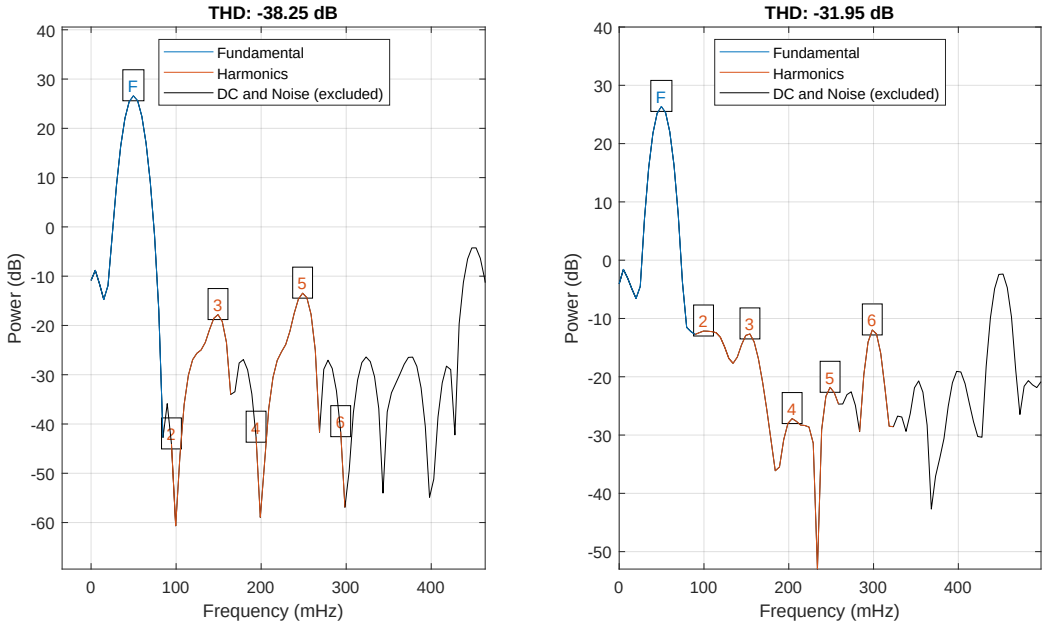

Figure 2: Total Harmonic Distortion for 7 bits

The results given respectively by MATLAB and C program are much different, because they use two different models to compute the final outputs. In fact, MATLAB works with high precision, while the C program, in order to model the fixed point

implementation, has some approximations to minus infinity. In fact, as showing in figure 3, the values from the C program are smaller than those from the MATLAB script. Because of these approximations, the evaluated THD of MATLAB output is much larger than the C program's one. The total harmonic distortion that have to be taken in consideration is the second one, since it simulates the wanted behaviour of the filter.

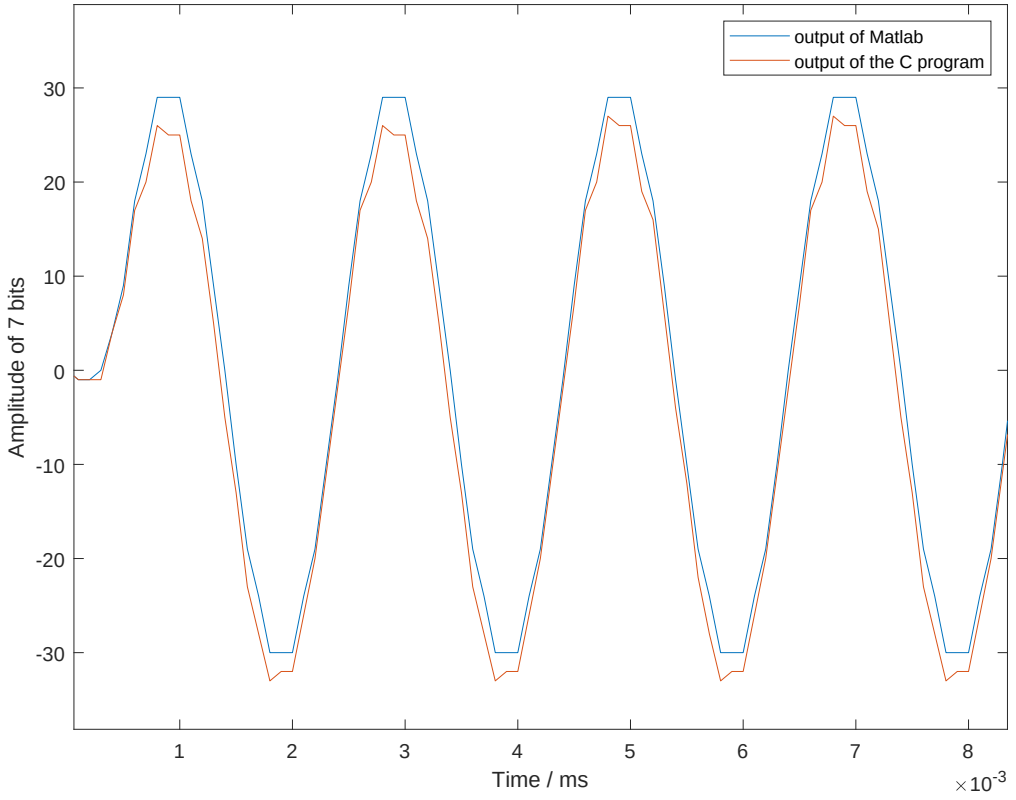

Figure 3: Output of the filter - 7 bits

So the minimum internal bits required to obtain a maximum THD of $-30\,dBc$ is **seven**.

2 VLSI implementation

2.1 Starting architecture development

The interface of the filter to be designed has to be equal to the one in figure 4. The requirements are described as follow:

- the samples (DIN) enter one each clock cycle with a validation signal (VIN);
- when VIN = '1' a new sample is loaded into the architecture, so if VIN = '0', the filter should "freeze" all the internal registers;
- the output DOUT contains the result of the filtering and VOUT is a validation signal, that is '1' when DOUT is ready.
- all the inputs and the outputs of the filter must be loaded/produced by registers. The data are represented as 2-complement normalized-fixed-point values, where the weight of the most significant bit, MSB, is -2^0, while for the least significant bit, LSB, it is 2^{-nb+1}.

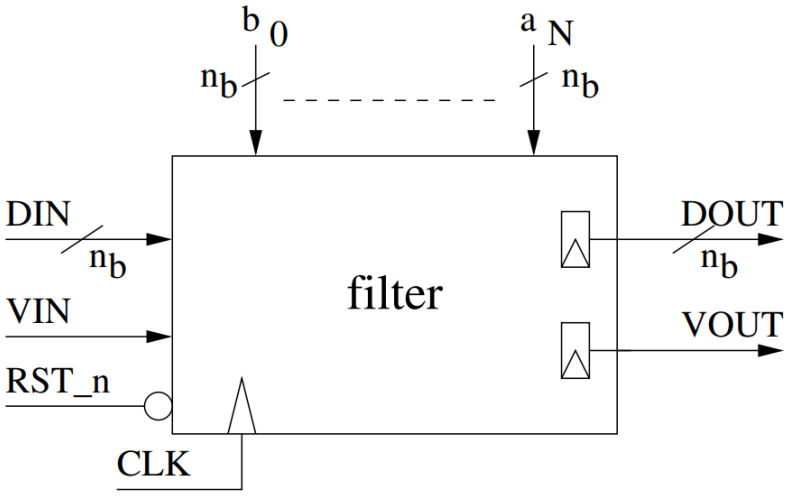

Figure 4: Filter interface

As it has been said in section 1, the internal bits of the filter are 7, while for the external interface the parallelism is 8-bit. To save more area for the architecture, the LSBs of all the input data (including the coefficients) are cut out and only the remaining 7 bits are saved into the registers.

To obtain the correct timing behaviour described before, a finite state machine is used, which cooperates with the datapath, as showing in figure 5.

The evolution of the state machine depends only on the signal "VIN" and obviously on "RST_n". The FSM generates all the enable signals for the registers in the datapath and "VOUT" is asserted when "DOUT" is ready. The details of this two blocks are described in the following sections.

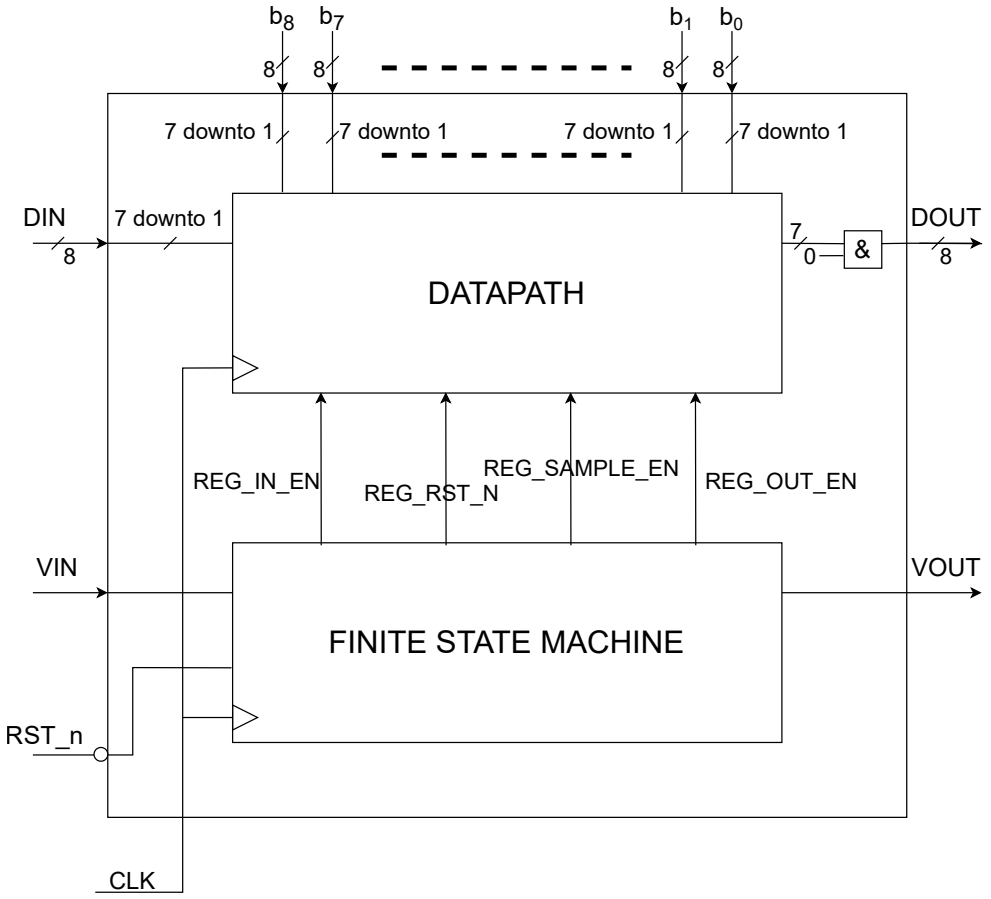

Figure 5: FIR filter

2.1.1 Datapath of the filter

The behaviour of a FIR filter can be described as:

$$y[n] = \sum_{i=0}^{N=8} b_i \cdot x[n-i]$$

Each sample is stored in a register and shifted in time, then multiplied by the coefficient, and finally all the results are summed. In order to do these operations, 8 internal registers, 9 multipliers and 8 adders are needed. The datapath designed is shown in figure 6. The input data are stored in an input register and also the outputs are produced by a register. It is possible to notice that both the input data and the coefficients are on 7 bits. The output of the multiplier is on 14 bits, but since all the data are represented as 2-complement normalized-fixed-point values, the MSB should not be considered. In order to optimize the area usage, the 13 bits at the output of the multiplier has been truncated to 7 bits. Since there is no overflow during the generation of the filter output, all the adder are on 7 bits.

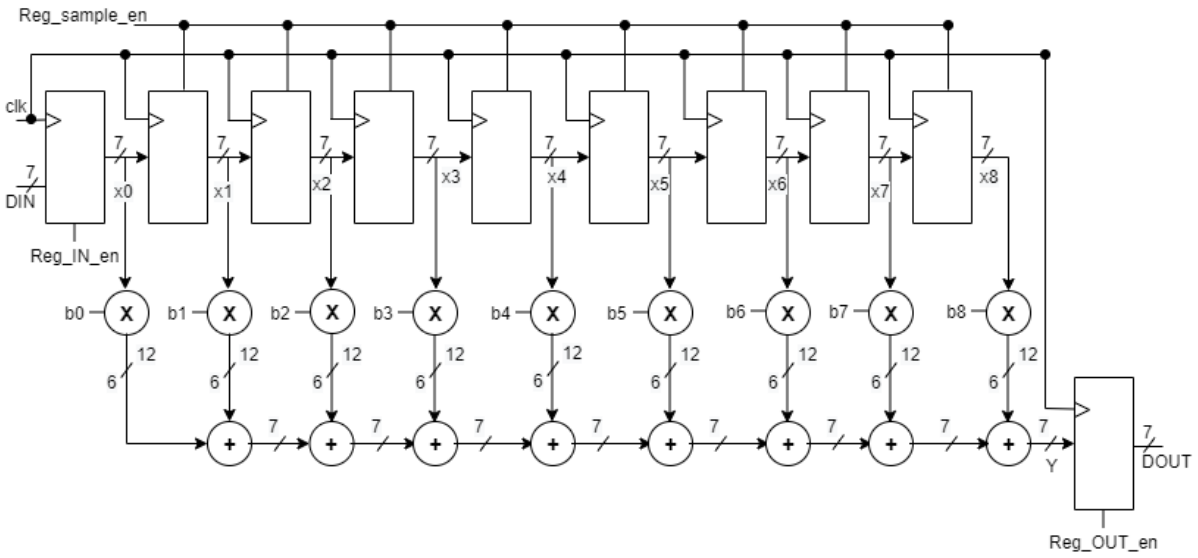

Figure 6: FIR filter - datapath

2.1.2 Finite state machine

As showing in figure 7, the FSM consists of 5 states:

- RESET: the FSM is in this state when a signal "rst_n" received from outside is asserted. All the registers of the datapath are initialized to zero;
- IDLE: then the FSM evolves in the IDLE state where the registers for the input data and for the coefficients are enabled. This is necessary because the data are available in input in the same clock cycle when the validation signal "VIN" is asserted;
- EXEC: in this state the entire computation is performed on the input data previously stored, therefore the output register is enabled in order to sample the result. The internal registers of the datapath are enabled as well, so that the samples stored within the filter are shifted. The signal "VOUT" should be asserted in the next state so as to validate the output;
- EXEC_CONT: where "CONT" stays for continuous. The FSM remains in this state until VIN goes to '0' or "rst_n" is asserted. It is almost equal to the EXEC state, except for the signal "VOUT" that is asserted;
- WAIT_STATE: the FSM enters in this state if VIN is sampled equal to '0' during a computation, meaning from the EXEC or EXEC_CONT states. The signal VOUT is high in order to validate the data of the output register coming from the previous cycle.

The Finite State Machine evolves according to the validation signal VIN:

- it moves from the IDLE state to EXEC when a valid sample is loaded in the architecture, so "VIN" = '1', otherwise it remains in the IDLE state.

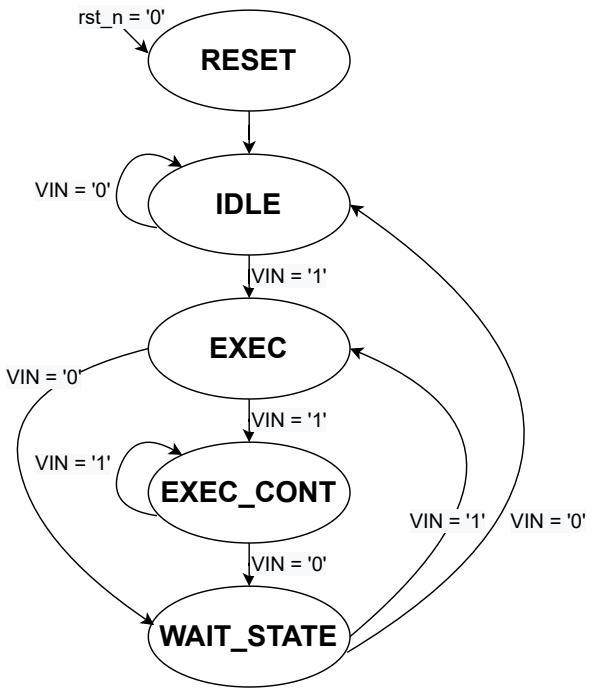

Figure 7: Finite State Machine of the filter

- when the machine is in the EXEC state, it goes in WAIT_STATE if no sample is available, so the filter gives the previous output data, but no new data is processed. On the other hand, if "VIN" = '1' it reaches the state EXEC_CONT;
- once EXEC_CONT is the present state, it can evolve in WAIT_STATE if "VIN" = '0', otherwise it remains in the current state if "VIN" = '1';
- if the present state is WAIT_STATE, it becomes EXEC when "VIN" = '1', otherwise it will be IDLE.

In order to clarify the evolution of the computation, the following timings are presented. The basic functionality is shown in figure 8. Here it is possible to see the evolution of the FSM starting from the RESET state resulted from the external *reset_n* signal, with the consequent change of the values stored in the registers *Reg_IN* and *Reg_OUT*.

When the *VIN* signal is asserted, the FSM goes into the EXEC state, setting the enable signal of the output register, which samples the first computed value *DOUT1* from the signal *Reg_OUT_D*. It is worth noting that the data *D0* is passed in input before the validation signal, therefore its elaboration is not returned as output. The enable signal *Reg_sample_en* is used to shift the input samples within the filter, therefore it has to be asserted following the validation signal *VIN*.

Lastly, the FSM goes into the EXEC_CONT state. As previously discussed, this state is equal to the EXEC state except for asserting the *VOUT* signal in order to validate the stored output.

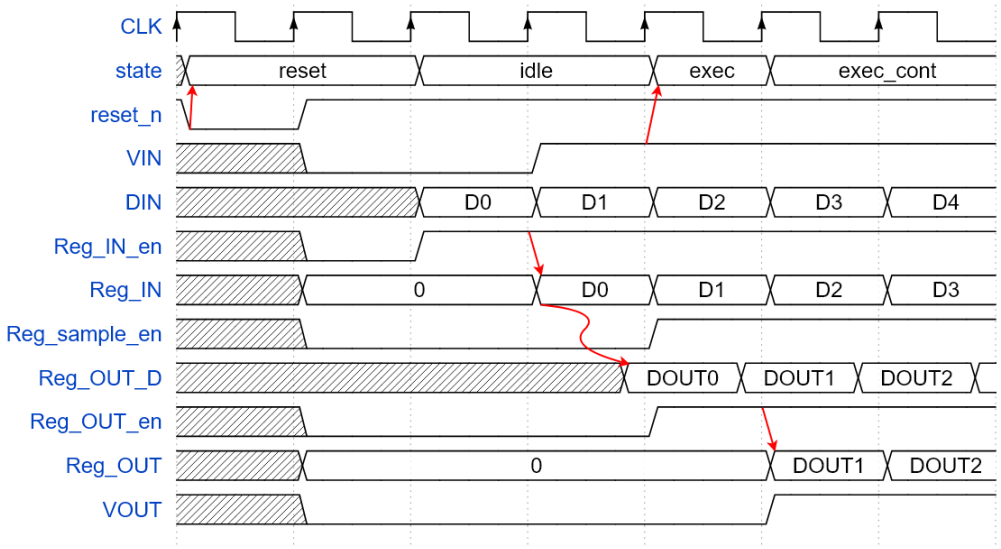

Figure 8: Basic FIR timing.

As it is possible to notice in figure 7, the FSM is capable to stop the computation following a *VIN* transition to '0', both when the present state is EXEC or EXEC_CONT, and to restart it whenever *VIN* goes to '1'.

In the timing shown in figure 9, the validation signal goes to logic '0' when the present state is EXEC and it does not change for at least two clock cycles. Here, in fact, the FSM goes to WAIT_STATE, where *VOUT* is asserted in order to validate the previously computed output. Then it goes back to the IDLE state.

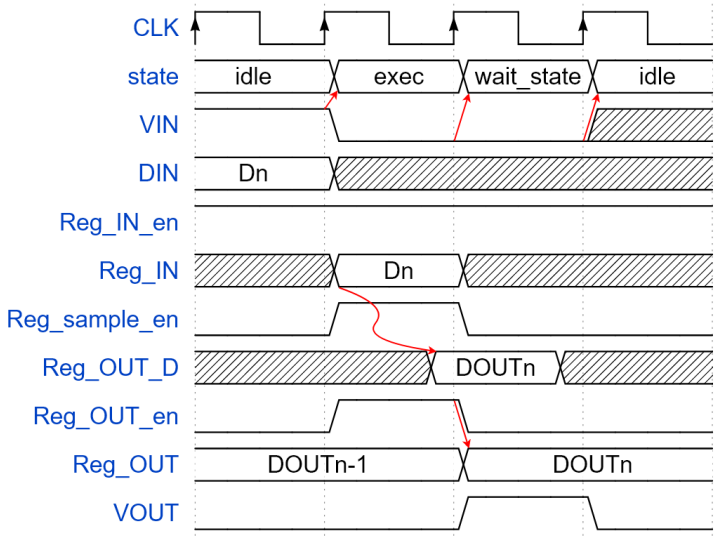

Figure 9: VIN signal goes to '0' for two or more clock cycles starting in the EXEC state.

Differently, if the *VIN* signal goes to '0' for one clock cycle, the computation has to be restarted immediately after the WAIT_STATE, as shown in figure 10. From this

point, the normal functionality is resumed.

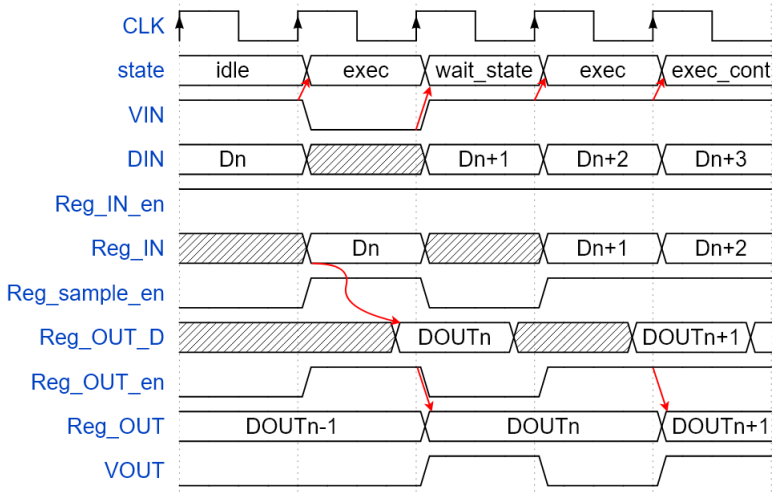

Figure 10: VIN signal is sampled '0' only during the EXEC state.

Moreover, if *VIN* signal is negated during the EXEC_CONT state, the behavior of the FSM is equivalent to the previous cases. An example is reported in figure 11, where the validation signal is negated for one clock cycle.

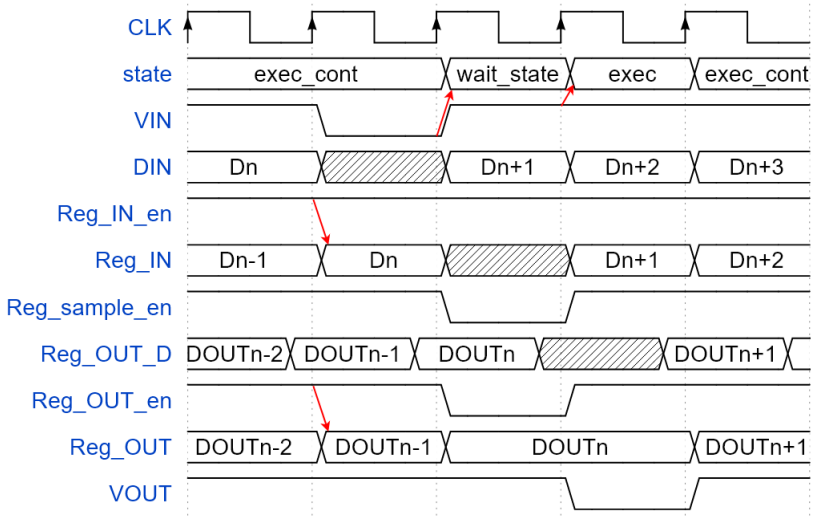

Figure 11: VIN signal is sampled '0' during EXEC_CONT state.

2.1.3 Description in VHDL

The datapath and the FSM are now described in VHDL, respectively in **datapath.vhd** and **fsm.vhd**, then they are connected into the file **FIR8_nbit_TOT.vhd**. It is interesting to notice that all the multipliers and the adders are described in behavioural.

2.2 Simulation

Once the filter had been described in VHDL, a simulation using ModelSim Mentor was necessary to validate the design. In order to achieve this, the following VHDL entities were used:

- **clk_gen:** it generates the clock and reset signals;
- **data_maker:** it reads the input from a text file which are used to drive the input of the device-under-test and generates the validation signal VIN;
- **data_sink:** it is used to collect the output and writes it into another text file when VOUT is asserted.

Finally, a top-level Verilog test-bench **tb_fir** is exploited in order to collect these entities along with the description of the filter.

With this method, it was possible to simulate the designed circuit using the same input file fed to the C model. Therefore, the validation of the circuit is achieved by comparing the two output files using the Python script *output_check.py*, shown in figure 12. In particular, this script checks the number represented in each line and, in the case there are any differences, they are displayed, reporting the line number and the two values found. At the end, it is reported whether the comparison outcome was successful or failed.

```python
ref_file = open("./output_c_7.txt", 'r')
dut_file = open("./results.txt", 'r')

error_flag = 0
line = 1

for dut_line in dut_file:
    dut_line = dut_line.strip()
    input_line = ref_file.readline().strip()

    dut_res = int(dut_line)
    exp_res = int(input_line)

    if exp_res != dut_res:
        error_flag = 1
        print(f"Error at line: {line}")
        print(f"dut result: {dut_res}")
        print(f"exp result: {exp_res}")
        print()
    line += 1

if error_flag == 1:
    print("File check failed!")
else:
    print("File check successful!")
```

Figure 12: Python script *output_check.py*.

The described procedure was followed in order to validate the circuit after each design step, meaning VHDL architecture design, synthesis and place & route.

In each of these cases, the first step is to compile all the VHDL and Verilog files and to start the simulation with the *vsim* command. Using the *add wave* * instruction, it was possible to show the main signal waves. In the logic simulation, an arbitrary clock cycle of 10 ns is used. At last, the simulation can be launched with the *run* command, specifying the simulation duration. In the presented work, the results were always equal to the one of the C model.

In figure 13 it is shown the first clock cycles of the simulation in which it is possible to see the correspondence with the timing reported in figure 8.

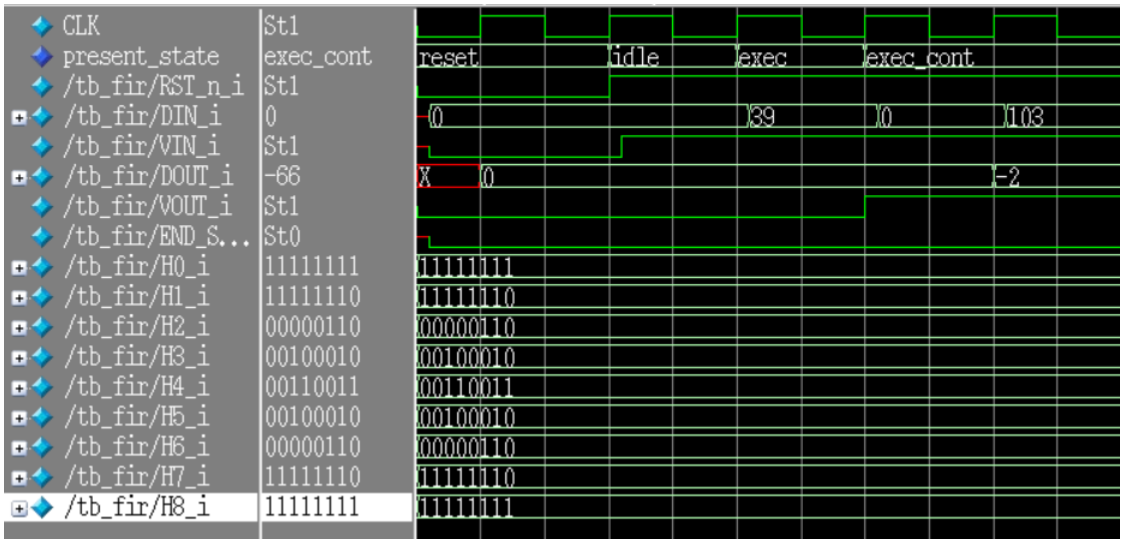

Figure 13: Simulation of the filter.

This test-bench is also used to verify the correct behaviour of the architecture in case of sampling VIN = '0' at any point of the computation. Using a counter variable inside the **data_maker** entity, it was possible to simulate all the different cases in which this may happen. One representative case is shown in figure 14, in which VIN goes to '0' during the EXEC_CONT state. It can be noticed that the output register is not updated, behaving exactly like in the timing reported in figure 11. For the sake of simplicity, no new data from the input file is read, in order to be able to re-use the output file as to verify the correct computations with the C model.

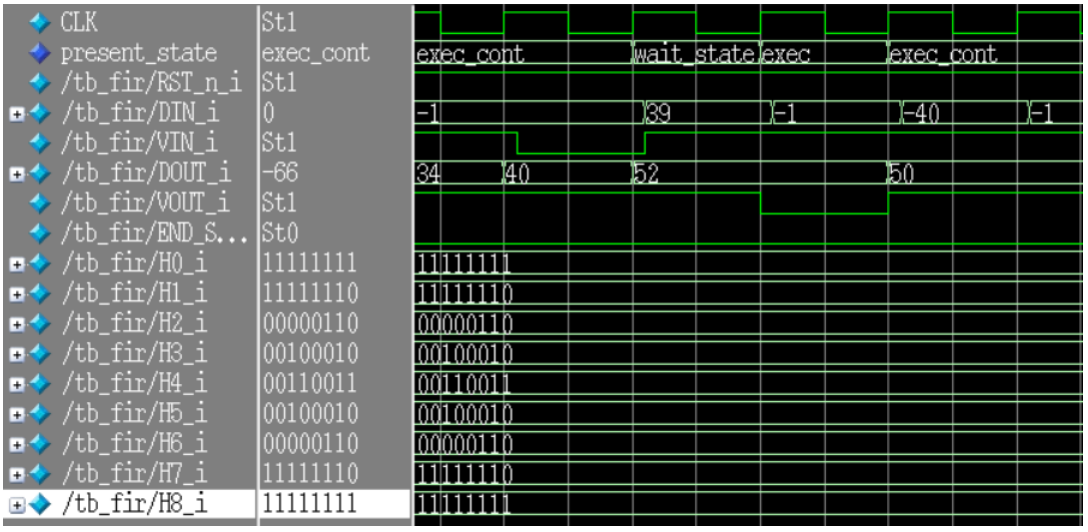

Figure 14: Simulation of the filter: VIN transation to '0'.

2.3 Implementation

2.3.1 Logic synthesis

The filter implemented has been synthesized, so the first step is to import the source files in Synopsys Design Compiler, applying the constraints and then starting the synthesis. In Design Compiler there are different possibilities to apply the constraints. One of these is to create a symbolic clock signal with a certain period and bind it to a real clock pin in the design.

At first a clock period is assumed equal to $0\,ns$ affected by jitter equal to $0.07\,ns$. Moreover, each input and output signals have some delay in relation to the clock equal to $0.5\,ns$. Finally, the load for each output is set assuming it is the input capacitance of a buffer, *BUF_X4*, in the library *NangateOpenCellLibrary*. A *script.scr* was implemented to do these operations.

At this point the critical path of the system was evaluated doing the command *report_timing*, the result is displayed in figure 15. The report shows some parameters that gives information about the critical path:

- **data required time**: latest time at which a signal can arrive without making the clock cycle longer than desired;
- **data arrival time**: time elapsed for the data to arrive at the output;
- **slack**: difference between the required time and the arrival time. In this case the slack is negative, which means the timing constraint was violated.

The clock period was constrained at $0\,ns$ in order to obtain the maximum frequency. The resulting negative slack is used in order to re-synthesize the design with the clock period which is expected to be the minimum one.

script.scr

```
analyze -f vhdl -lib WORK ../src/GenericReg.vhd
analyze -f vhdl -lib WORK ../src/datapath.vhd
analyze -f vhdl -lib WORK ../src/fsm.vhd
analyze -f vhdl -lib WORK ../src/FIR8_nbit7_TOT.vhd
set power_preserve_rtl_hier_names true
elaborate FIR8_nbit7_TOT -arch structural -lib WORK > ./elaborate.txt
uniquify
link
create_clock -name MY_CLK -period 0 clk
set_dont_touch_network MY_CLK
set_clock_uncertainty 0.07 [get_clocks MY_CLK]
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 OLOAD [load_of NangateOpenCellLibrary/BUF_X4/A]
set_load $OLOAD [all_outputs]
compile
```

```
clock MY_CLK (rise edge)                                 0.00       0.00
clock network delay (ideal)                              0.00       0.00
clock uncertainty                                       -0.07      -0.07
datapath_FIR/Reg_out/q_reg[6]/CK (DFF_X1)                0.00      -0.07 r
library setup time                                      -0.04      -0.11
data required time                                                 -0.11
--------------------------------------------------------------------------
data required time                                                 -0.11
data arrival time                                                  -1.54
--------------------------------------------------------------------------
slack (VIOLATED)                                                   -1.65
```

Figure 15: Timing report with clock constrain of 0 ns

This is repeated until the slack is equal to 0. After different attempts, the minimum clock period is set to 1.88 ns, corresponding to the maximum clock frequency equal to 531.92 MHz, as it is possible to see in the figure 16.

```
clock MY_CLK (rise edge)                                 1.88       1.88
clock network delay (ideal)                              0.00       1.88
clock uncertainty                                       -0.07       1.81
datapath_FIR/Reg_out/q_reg[6]/CK (DFF_X1)                0.00       1.81 r
library setup time                                      -0.04       1.77
data required time                                                  1.77
--------------------------------------------------------------------------
data required time                                                  1.77
data arrival time                                                  -1.77
--------------------------------------------------------------------------
slack (MET)                                                         0.00
```

Figure 16: Timing report with clock period of 1.88 ns

It was also found that the total area corresponding to the maximum clock frequency with the command report_area is equal to 3115.66 μm^2, in figure 17:

```
Number of ports:                          943
Number of nets:                          2921
Number of cells:                         1779
Number of combinational cells:           1595
Number of sequential cells:               136
Number of macros/black boxes:               0
Number of buf/inv:                        343
Number of references:                       2

Combinational area:               2496.942009
Buf/Inv area:                      186.732001
Noncombinational area:             618.715979
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                  3115.657988
Total area:                 undefined
```

Figure 17: Report_area

Then, the clock period is set to $7.52\,ns$ corresponding to the clock frequency $f_M/4 = 132.98\,MHz$, as it is required from the specifications. To verify that the filter is still working properly after the synthesis, it is necessary to start a new simulation using the *Verilog* test-bench. As expected, the output results obtained are the same of the VHDL code.

Finally, it is necessary to save the data required to complete the design with the place and route and to perform the switching activity necessary to do the power estimation. First, the cells were ungrouped to flatten the hierarchy and the *Verilog* rules for the names of the internal signals were imposed. Then, it is possible to save the file *.sdf*, which contains the information about the internal delays. Now, both the netlist in *Verilog* and the constraints to the input and output ports must be saved, producing *.v* and *.sdc* files.

In order to calculate the power consumption of the filter, it is necessary to extract the information about the switching activities of the nodes from Modelsim and evaluate them with Synopsys, so the two program must be used together. More in detail the few steps are:

- launch Modelsim and record the switching activity;
- convert the file with the switching activity from *vcd* to *saif*;
- evaluate the power consumption with Synopsys Design Compiler.

A *script_activity.do* was implemented to do the first operation.

After the compilation of the *.vhd* and *.v* files, the delay one *.sdf* is included, so Modelsim generates a *.vcd* file with the switching activity information. This one has been converted into a *.saif* that Synopsys is able to read and to use in order to estimate the total power consumption of the circuit.

script_activity.do

```
vcom -93 -work ./work ../tb/clk_gen.vhd
vcom -93 -work ./work ../tb/data_maker_new.vhd
vcom -93 -work ./work ../tb/data_sink.vhd
vlog -work ./work ../netlist/FIR8_nbit7_TOT.v
vlog -work ./work ../tb/tb_fir.v
vsim -L /software/dk/nangate45/verilog/msim6.2g
-sdftyp /tb_fir/UUT=../netlist/FIR8_nbit7_TOT.sdf work.tb_fir
vcd file ../vcd/FIR8_nbit7_TOT_syn.vcd
vcd add /tb_fir/UUT/*
```

It is necessary to read the netlist, the *.saif* file generated by the Modelsim simulation and to set the clock signal in the design. Finally, the *report_power* command is executed for the design at frequency $f_M/4$.

Power report at f/4

```
Cell Internal Power  = 303.4311 uW   (62%)
Net Switching Power  = 186.0526 uW   (38%)
                       ---------
Total Dynamic Power    = 489.4837 uW  (100%)

Cell Leakage Power     =  62.3434 uW

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power
-----------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000
memory             0.0000           0.0000            0.0000             0.0000
black_box          0.0000           0.0000            0.0000             0.0000
clock_network      0.0000           0.0000            0.0000             0.0000
register         126.1648          24.3160        1.0724e+04           161.2048
sequential         0.0000           0.0000            0.0000             0.0000
combinational    177.2662         161.7368        5.1619e+04           390.6224
-----------------------------------------------------------------------------
Total            303.4310 uW      186.0527 uW     6.2343e+04 nW       551.8273 uW
```

It is possible to see in evidence that the *Total Dynamic Power* = 489.48 uW, due to the sum of the *Cell Internal Power* of all the elements in the circuit and the *Net Switching Power* due to the commutation of the signals. There is also an amount of *Cell Leakage Power* = 62.34 uW. From this two contributions, the total power was derived as:

$$\textit{Total Power} = \textit{Total Dynamic Power} + \textit{Cell Leakage Power} = 551.83\,uW.$$

2.3.2 Place & Route

After the synthesis of the design, the last step of implementation is the place and route at $f_M/4$. It can be performed with the Cadence Innovus tool. There are several steps that must be followed.

Importing the design At first, it was necessary to import the design of the FIR, loading the files *design.globals*, customized for the submitted design, and *mmm_design.tcl*, which specifies the setup for the timing analysis, figure 18.

Structuring the floorplan For the next step, Innovus defines the area where the power supply will be positioned. This process is called *Floorplan*. The dimensions are set accordingly to the specifications. At this step, it is possible to see gray horizontal lines that specify how many rows will be needed for the design, figure 19.

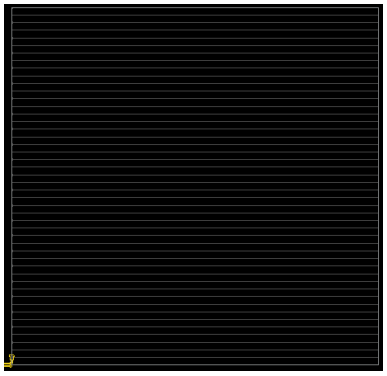

Figure 18: Design area

Figure 19: Floorplan

Inserting Power Rings Now it is possible to insert two metal *power rings*, power supply VDD and ground VSS, all around the die. These stripes could be connected with VDD and VSS pads if using a wirebonding package. Also, vias are created in the corner, useful for the connection between metal levels, figure 20.

Standard cell power routing It is also possible to proceed to place the horizontal wires preparing the VDD and VSS wires for the standard cells. These will be connected to the two rings creating the power routing, figure 21.

Placement The next step requires to proceed with the *Placement*, which means to allocate all the cells of the design in the defined area. In this way, each one of them obtains a unique position in the rows, figure 22.

Post Clock-Tree-Synthesis optimization Before running the routing, the design could be optimized in order to achieve the required timing constraints using the *Post-CTS* optimization, figure 23.

Place filler For technological reasons, it is necessary to complete the placement, filling the holes to ensure continuity in n+ and p+ wells in each row. For this purpose, filler cells are used, figure 24.

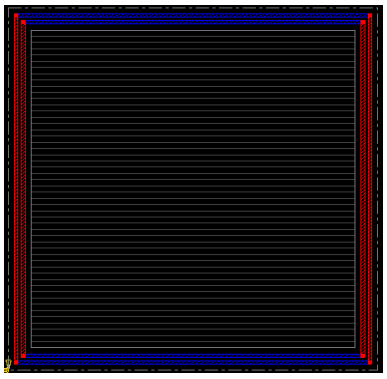

Figure 20: Power rings

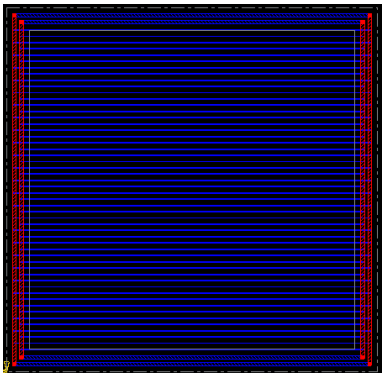

Figure 21: Power routing

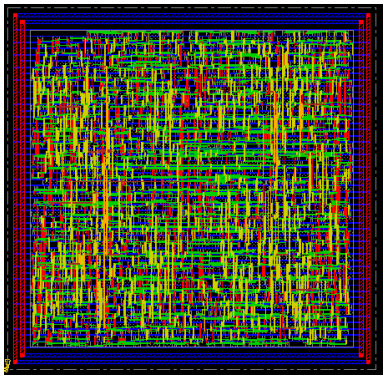

Figure 22: Placement

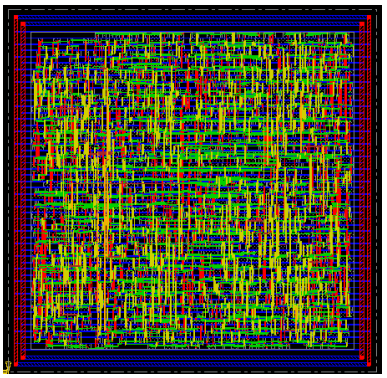

Figure 23: Clock-Tree-Synthesis

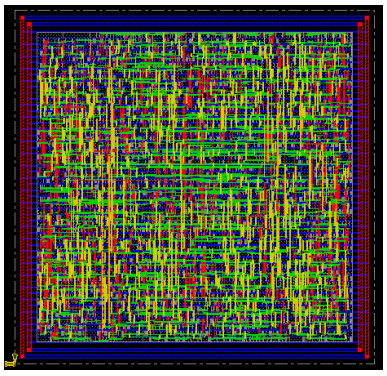

Figure 24: Place filler

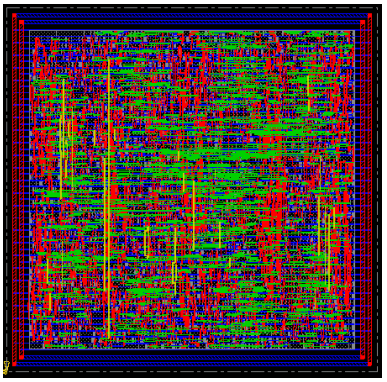

Figure 25: Routing

Routing The connections among the cells are created using the available metal layers, figure 25.

Post routing optimization Once the design is completed, it can be optimized in order to achieve the required timing constraints using *Post-Route*, figure 26.

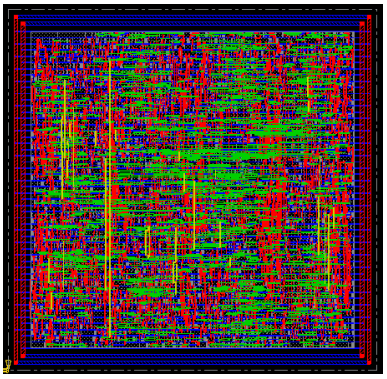

Figure 26: Post routing optimization

Parasitics extraction Now it is possible to proceed analyzing the time behaviour, using the parasitic resistance and capacitance values for each metal wire, which are evaluated performing a *Parasitics extraction* using technology and geometry information.

Timing analysis Then the timing analysis is performed for both setup and hold times. The slack was always positive as noticeable in the files **FIR8_nbit7_TOT_postRoute.slk** and **FIR8_nbit7_TOT_postRoute_Hold.slk**. Differently, it would mean that the timing constraints are not respected.

Design analysis and verification The next step is the design analysis and verification. Therefore, it was verified that both the connectivity and the geometry have no violations. At last, the area and the gate count data are computed using the *Gate Count* option in the Innovus tool.

Verify Connectivity

```
******** Start: VERIFY CONNECTIVITY ********
Design Name: FIR8_nbit7_TOT
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (83.0300, 81.4800)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
```

As it is possible to see in figure 27, the $Area = 3064.1\,\mu m^2$ is smaller than the previous one computed during the synthesis, due to a greater accuracy of this estimation after the place and route.

Verify Geometry

```
*** Starting Verify Geometry (MEM: 1177.7) ***

**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: 10.00
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**********
```

```
Gate area 0.7980 um^2
Level 0 Module FIR8_nbit7_TOT
Gates=  3839 Cells=  1666 Area=  3064.1 um^2
```

Figure 27: Area_post_route

Finally, it is required to evaluate the power consumption post place and route doing similar steps to the ones of the procedure followed after the synthesis. In this case, it is not necessary to translate in the *.saif* format, because Innovus is already able to communicate with ModelSim using *.vcd* files. Therefore ModelSim can store the information related to the switching activity estimation of the circuit in the *vcd* directory. The script *script_post_route.do* was exploited in order to perform all of these operations.

script_post_route.do

```
vcom -93 -work ./work ../tb/clk_gen.vhd
vcom -93 -work ./work ../tb/data_maker_new.vhd
vcom -93 -work ./work ../tb/data_sink.vhd
vlog -work ./work ../innovus/FIR8_nbit7_TOT.v
vlog -work ./work ../tb/tb_fir.v
vsim -L /software/dk/nangate45/verilog/msim6.2g
-sdftyp /tb_fir/UUT=../innovus/FIR8_nbit7_TOT.sdf work.tb_fir
vcd file ../vcd/design.vcd
vcd add /tb_fir/UUT/*
```

After the simulation, it is possible to go to the *innovus* directory and use the *Cadence Innovus* tool to do a power estimation using the switching activity evaluation saved in the *.vcd* file. Therefore, the *report_power_post_route.txt* is generated.

report_power_post_route.txt

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:        0.57343115     57.0774%
Total Switching Power:       0.37051248     36.8796%
Total Leakage Power:         0.06071203      6.0431%
Total Power:                 1.00465567
-----------------------------------------------------------------------------------------

Group                            Internal   Switching     Leakage       Total  Percentage
                                 Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                         0.1349     0.02666     0.01072      0.1722       17.14
Macro                                   0           0           0           0           0
IO                                      0           0           0           0           0
Combinational                      0.4386      0.3438     0.04999      0.8324       82.86
Clock (Combinational)                   0           0           0           0           0
Clock (Sequential)                      0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                              0.5734      0.3705     0.06071       1.005         100
-----------------------------------------------------------------------------------------

Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                       1.1     0.5734      0.3705     0.06071       1.005         100
```

In evidence, there are different contributions of power: the *Total Dynamic Power* = $0.944\,mW$, due to the sum of the *Total Internal Power* and the *Total Switching Power*, and the *Cell Leakage Power* = $0.061\,mW$. From this two contributions, the total power was derived as:

$$\begin{aligned} \textit{Total Power} = &\textit{Total Internal Power} + \textit{Total Switching Power} + \\ &+ \textit{Total Leakage Power} = 1.005\,mW \end{aligned}$$

This value is greater than the *Total Power* estimated after the synthesis, because the evaluation after the place and route is more accurate and it also takes into account the power of the interconnections.

3 Advanced architecture development

3.1 Starting architecture development

In order to improve the throughput of the FIR filter described in section 2 some techniques can be applied. In this section, a 3-order unfolding is exploited and after that the pipeline is used.

3.1.1 Unfolding of order 3

Since the order is 3, the architecture of the FIR filter should be triplicated and then, using the unfolding technique, the registers are placed in their proper path.

Numerate the three blocks as 0, 1, 2, and then call:

- $\boldsymbol{i}$ the number of the block of the beginning of an arc and $\boldsymbol{j}$ its destination.
- $\boldsymbol{N}$ the order of the unfolding, in this case it is 3;
- $\boldsymbol{w}$ the number of registers in the original arcs and $\boldsymbol{w_i}$ the number of registers in the new arcs.

The destination j is computed as $j =\mod(\frac{i+w}{N})$, which means that j is equal to the reminder of the division between $(i + w)$ and N. Instead, w_i is obtained by $w_i = \left\lfloor \frac{i+w}{N} \right\rfloor$, so w_i is equal to the greatest integer smaller than the result of the division $(i + w)$ over N.

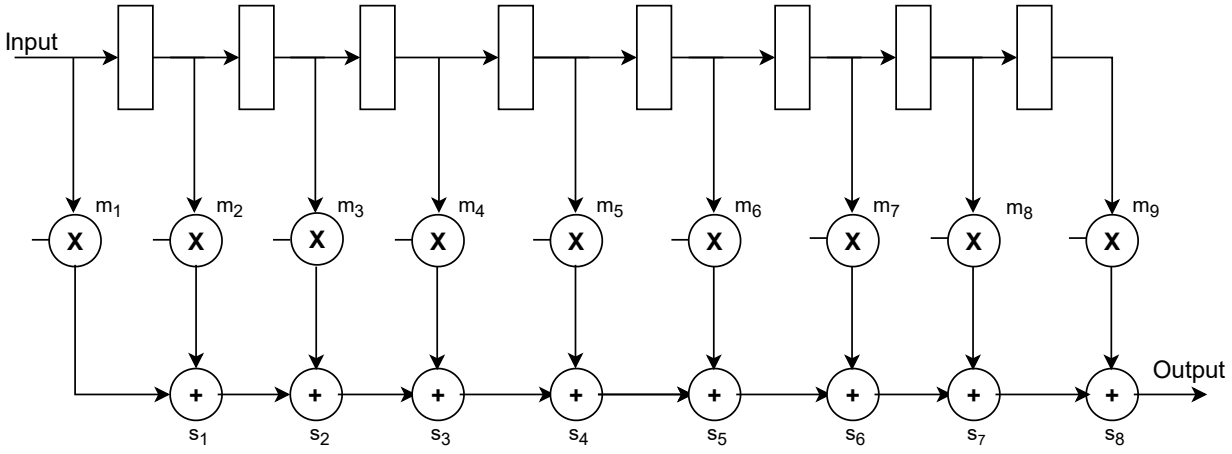

Figure 28: FIR filter without optimization

To use the unfolding technique described before, the figure 28 is taken as a reference point. It is interesting to notice that if $w = 0$ the destination is $j =\mod(\frac{i}{N})$, but since i is a number between 0 and $N - 1$, the final result is that $j = i$ and $w_i = 0$. This means that all the arcs with no registers will not become edges across two different blocks, therefore it will be a direct connection in the same block.

Block i	Arc	# of registers w	Destination j	# of registers w_i
0,1,2	$m_1 \rightarrow s_1$	0	i	0
	$m_2 \rightarrow s_1$	0	i	0
	$m_3 \rightarrow s_2$	0	i	0
	$m_4 \rightarrow s_3$	0	i	0
	$m_5 \rightarrow s_4$	0	i	0
	$m_6 \rightarrow s_5$	0	i	0
	$m_7 \rightarrow s_6$	0	i	0
	$m_8 \rightarrow s_7$	0	i	0
	$m_9 \rightarrow s_8$	0	i	0
	$s_1 \rightarrow s_2$	0	i	0
	$s_2 \rightarrow s_3$	0	i	0
	$s_3 \rightarrow s_4$	0	i	0
	$s_4 \rightarrow s_5$	0	i	0
	$s_5 \rightarrow s_6$	0	i	0
	$s_6 \rightarrow s_7$	0	i	0
	$s_7 \rightarrow s_8$	0	i	0
0	$Input \rightarrow m_1$	0	0	0
	$Input \rightarrow m_2$	1	1	0
	$Input \rightarrow m_3$	2	2	0
	$Input \rightarrow m_4$	3	0	1
	$Input \rightarrow m_5$	4	1	1
	$Input \rightarrow m_6$	5	2	1
	$Input \rightarrow m_7$	6	0	2
	$Input \rightarrow m_8$	7	1	2
	$Input \rightarrow m_9$	8	2	2
1	$Input \rightarrow m_1$	0	1	0
	$Input \rightarrow m_2$	1	2	0
	$Input \rightarrow m_3$	2	0	1
	$Input \rightarrow m_4$	3	1	1
	$Input \rightarrow m_5$	4	2	1
	$Input \rightarrow m_6$	5	0	2
	$Input \rightarrow m_7$	6	1	2
	$Input \rightarrow m_8$	7	2	2
	$Input \rightarrow m_9$	8	0	3
2	$Input \rightarrow m_1$	0	2	0
	$Input \rightarrow m_2$	1	0	1
	$Input \rightarrow m_3$	2	1	1
	$Input \rightarrow m_4$	3	2	1
	$Input \rightarrow m_5$	4	0	2
	$Input \rightarrow m_6$	5	1	2
	$Input \rightarrow m_7$	6	2	2
	$Input \rightarrow m_8$	7	0	3
	$Input \rightarrow m_9$	8	1	3

Table 2: Results of the unfolding technique

By applying the unfolding method, the results are shown in the table 2, it is easy to notice that the total number of registers inserted is exactly equal to the non-optimized version, which is 9. A clear view is shown in figure 29.

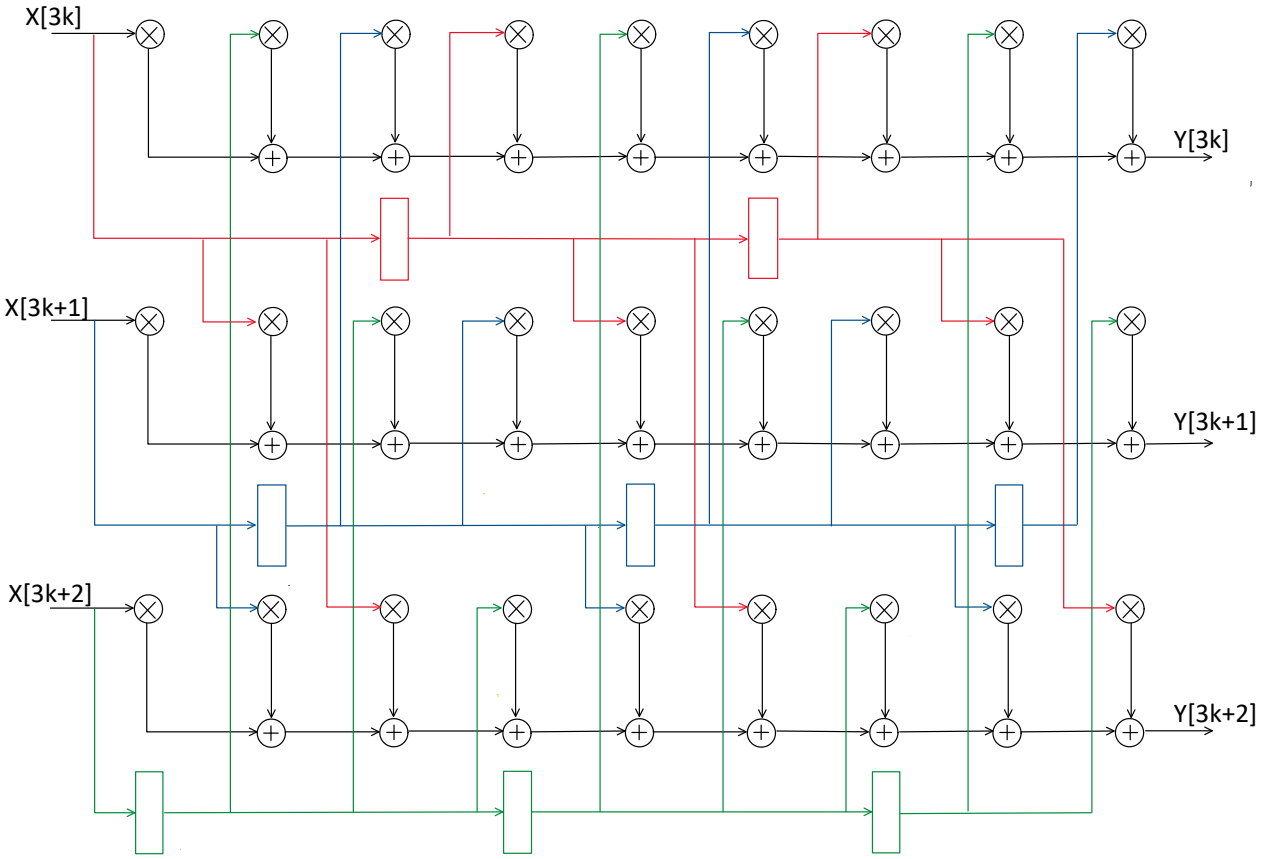

Figure 29: FIR filter with order 3 unfolding technique

3.1.2 Pipeline

It is interesting to notice that the critical path in this case is exactly the same as the non-optimized one, $T_{cp} = T_m + 8 \cdot T_a$, where T_a is the delay of the adder and T_m the delay of the multiplier. But since the number of outputs have been triplicated, also the final throughput is triplicated, $Th = 3/T_{cp}$.

In order to get an even faster architecture, the pipeline technique can be applied, therefore reducing the critical path. The aim is to minimize the combinational delay by applying the formal pipelining method. Some feed-forward cut-set have been identified and on each edge one register has been inserted. The final result reached is shown in figure 30. It is easy to notice that the critical path delay has been reduced to only one operator, therefore it is $T_{cp} = max\{T_a, T_m\}$.

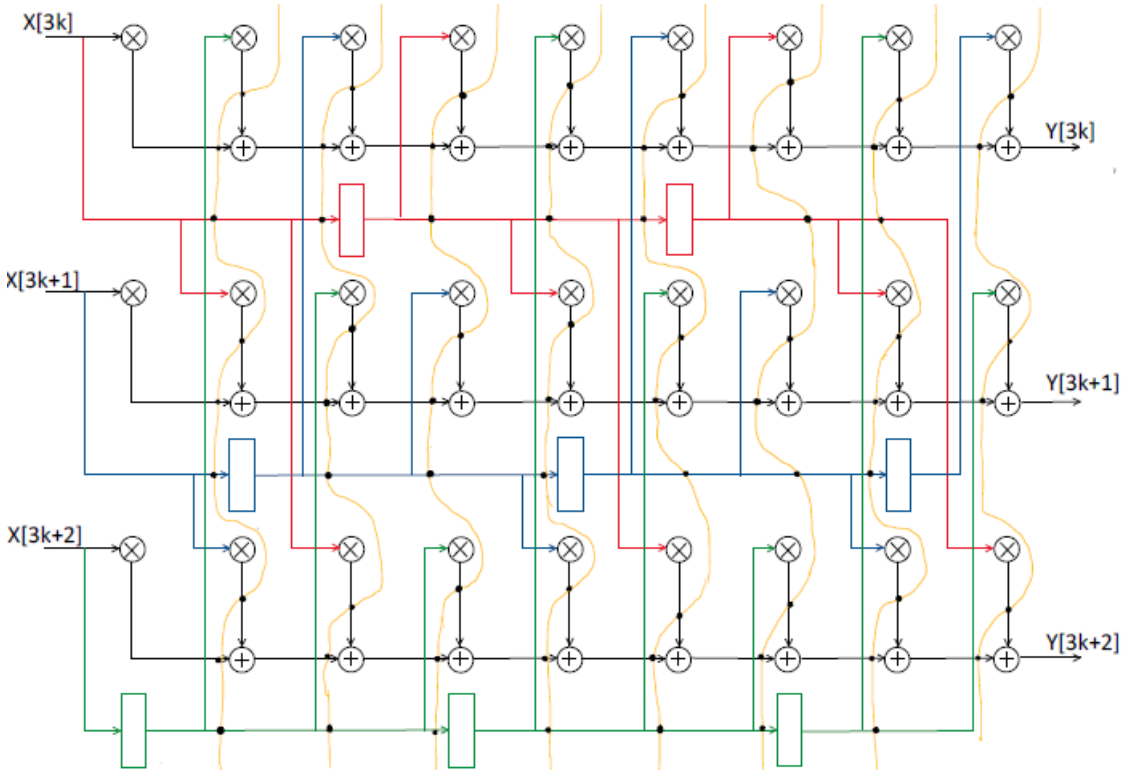

Figure 30: Advanced architecture

3.1.3 Datapath

The datapath derives directly from the figure 30, inserting three registers for the inputs, three registers for the outputs and only nine registers for the coefficients, since they are the same for all the three blocks. Also a counter is needed since the pipeline introduces a higher latency. The final datapath is shown in figure 31.

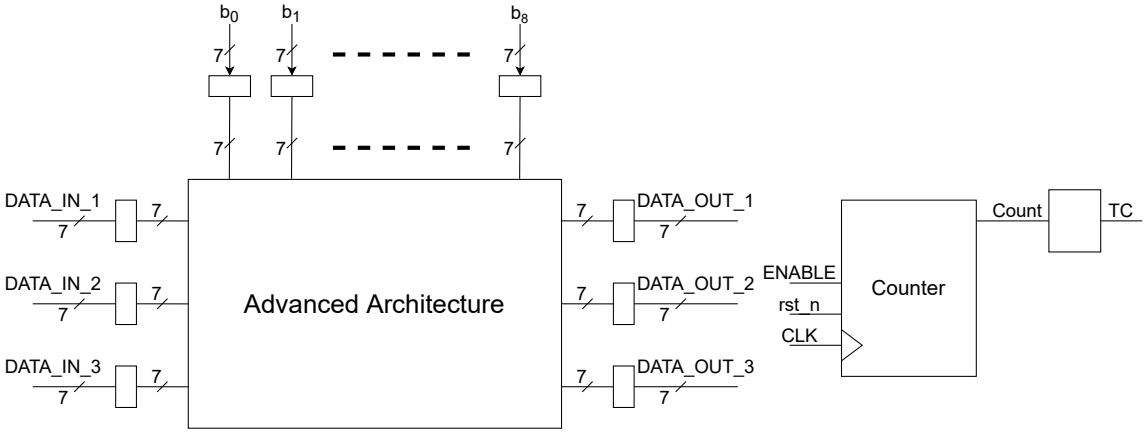

Figure 31: Datapath of the optimized filter

3.1.4 Finite State Machine

The Control Unit of the improved architecture consists in a Finite State Machine similar to the previous one, but with the complication of the pipeline. In particular, it has to wait for the first input data to go through the entire pipeline before returning any valid output. After this first stage the pipeline is full and returns three valid output data for each clock cycle as long as the validation signal VIN is asserted. Moreover, the possible sampling of VIN to logic '0' at any moment was taken into account in order to avoid the evolution of the samples within the filter, therefore permitting to resume the operation without any computational error.

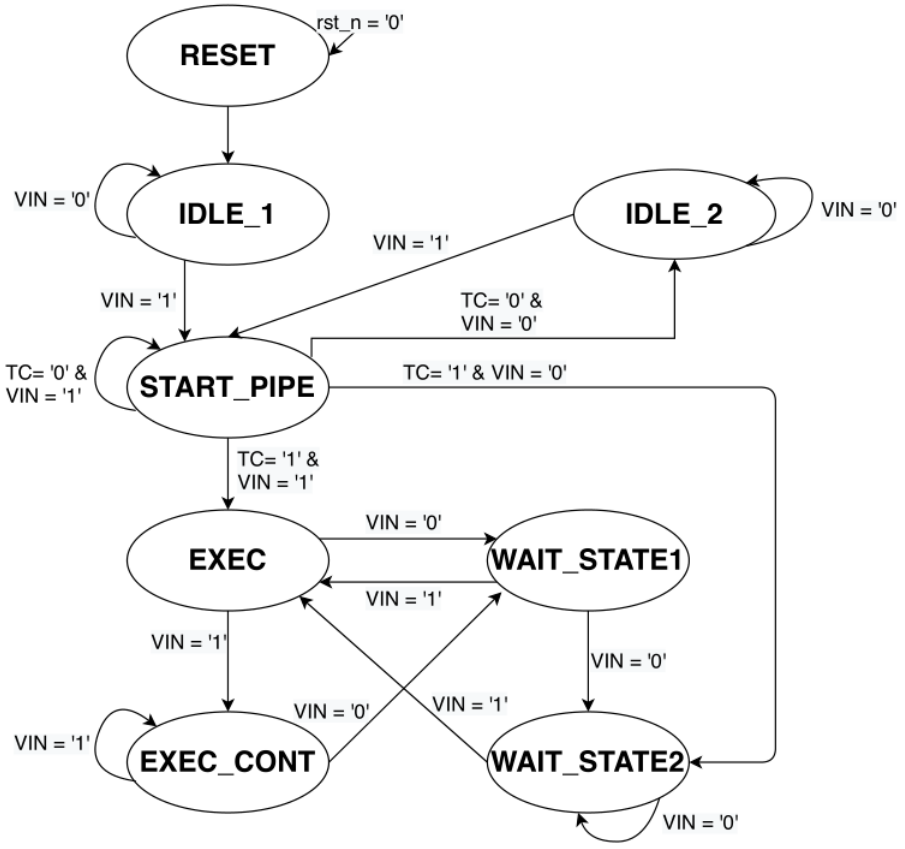

Figure 32: Finite State Machine of advanced architecture.

The designed FSM is shown in figure 32. Here it is possible to notice that 8 states were necessary to implement the CU as follows:

- RESET: it is the starting point of the FSM whenever the signal *rst_n* is asserted. All the registers are initialized to 0, therefore the pipe is emptied.
- IDLE_1: corresponds to the previous FSM's IDLE state. This state is used to enable the three input registers in order to sample the input data, which will be ready in the same clock cycle when VIN is sampled equal to '1'. Moreover, the coefficients are stored into the corresponding registers.

- START_PIPE: this state is used to sample 8 sets of three input data and the internal registers are enabled in order to fill the pipeline. A counter is used to wait for the correct amount of cycles, taking into account a possible VIN transition to '0'. In that case, the FSM goes into IDLE_2 state. When the counter has finished, a terminal counter *TC* signal is used to let the FSM moves to the EXEC state. In case VIN = '0' and TC = '1' in the same cycle, the FSM goes into WAIT_STATE2.
- IDLE_2: the FSM moves into this state only if during the START_PIPE stage VIN was sampled equal to '0'. This state freezes the counter and prevents the samples stored within the filter to shift. Once VIN is sampled equal to '1', the FSM goes back to the START_PIPE state and resumes its normal operation.
- EXEC: this state is equivalent to the one in the previous FSM. The first three valid output are computed, therefore output registers are enabled. If VIN signal is '1', the FSM evolves to the EXEC_CONT state, otherwise it goes into WAIT_STATE1.
- EXEC_CONT: the FSM remains in this state until VIN = '0' or the reset signal is asserted. This state is equal to the EXEC state, but asserts the VOUT signal in order to validate the previous output data. In case VIN is negated, the FSM moves to WAIT_STATE1.
- WAIT_STATE1: the FSM may arrive into this state from EXEC or EXEC_CONT states. This is used to prevent the samples evolution within the filter, but asserts the VOUT signal in order to validate the previous output data. For this reason, if VIN = '0' the FSM moves to WAIT_STATE2, otherwise it goes back to the EXEC state, resuming its normal functionality.
- WAIT_STATE2: the FSM is in this state when VIN = '0' for two or more clock cycles. This state is equivalent to the WAIT_STATE1, but VOUT is negated as the last output data were already validated.

In order to clarify the behaviour of the filter, the following timings are presented. In figure 33 the basic computation is shown assuming VIN = '1'.

Differently from the previous implementation, the IDLE state is followed by a series of START_PIPE cycles, during which the signal *Counter_en* is asserted in order to enable a 3-bit counter. The signal *Counter* represents its content, which enables the terminal count signal *TC* when it is equal to 7. In this way, the FSM moves to the EXEC state which fill the last stage of the pipeline and computes the output data. In fact, the output register is enabled exactly in the same way it was for the first FSM. Lastly, the FSM goes into the EXEC_CONT state, in which VOUT signal is asserted and the output registers store the valid output data.

The validation signal VIN could provide for the freeze of the filter when it is sampled equal to '0' during three states: START_PIPE, EXEC or EXEC_CONT. The last two states behave in the same way, therefore they will be analyzed together.

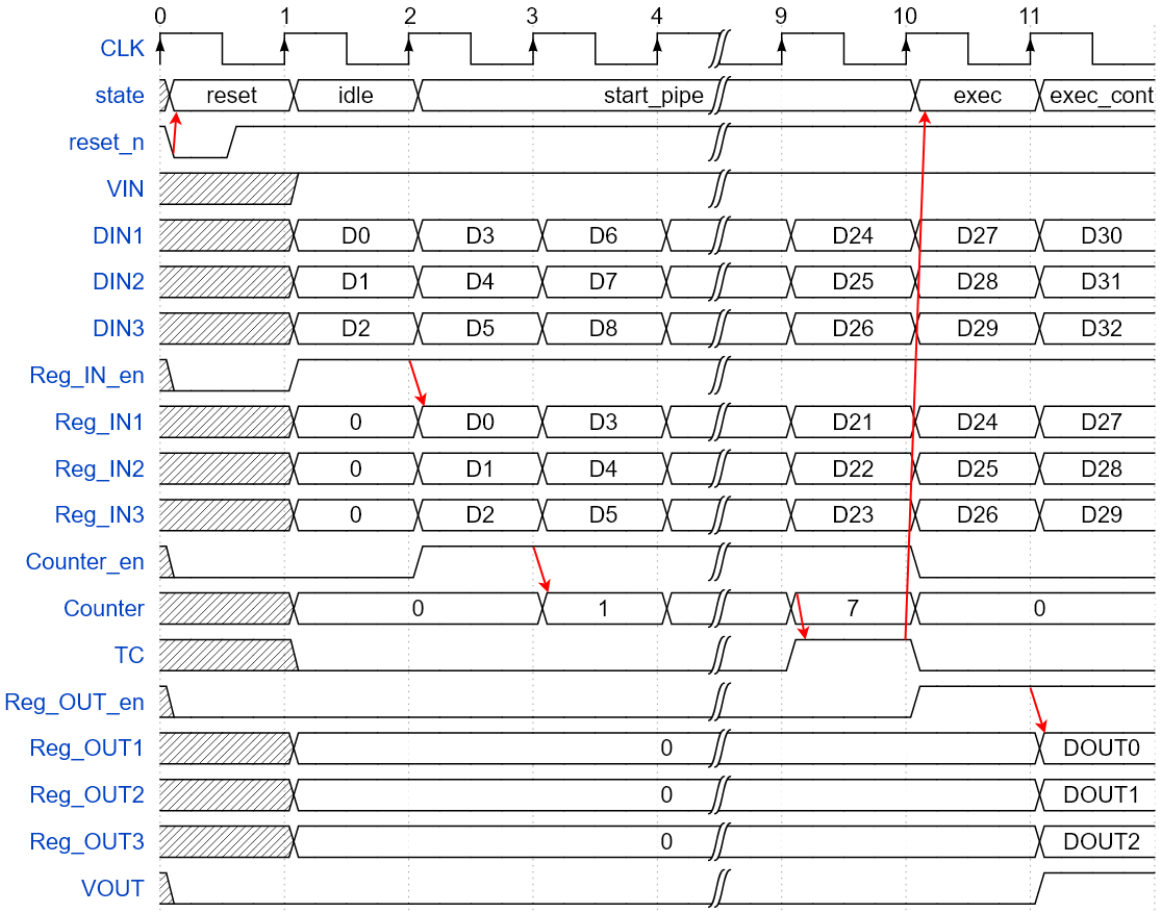

Figure 33: Advanced FIR timing.

The first case is when the VIN signal is negated in a generic cycle during the START_PIPE stage. This is represented by the timing in figure 34. Here it is possible to notice how the counter is frozen by negating the *Counter_en* signal during the IDLE_2 state.

More interesting could be the case in which VIN = '0' and TC = '1' in the same cycle. This is represented by the timing in figure 35. In fact, here the FSM goes into WAIT_STATE2, in which the samples within the filter are not shifted, and it subsequently moves into the EXEC state. It is worth noticing that the data elaborated in this state are the same than the ones in figure 33.

If the VIN signal goes to '0' when the FSM is in the EXEC or EXEC_CONT states, the next state is WAIT_STATE1. This is needed in order to validate the previously computed output by asserting the VOUT signal. In the case VIN is equal to '0' for two or more clock cycles, the FSM goes into WAIT_STATE2, where VOUT signal is negated. From this point, only when VIN is sampled equal to '1', the FSM resumes its normal operation. Two examples are shown in the timings in figures 36 and 37. Here it is possible to notice how the VOUT signal is driven with the evolution of the FSM and how the corresponding output data are validated.

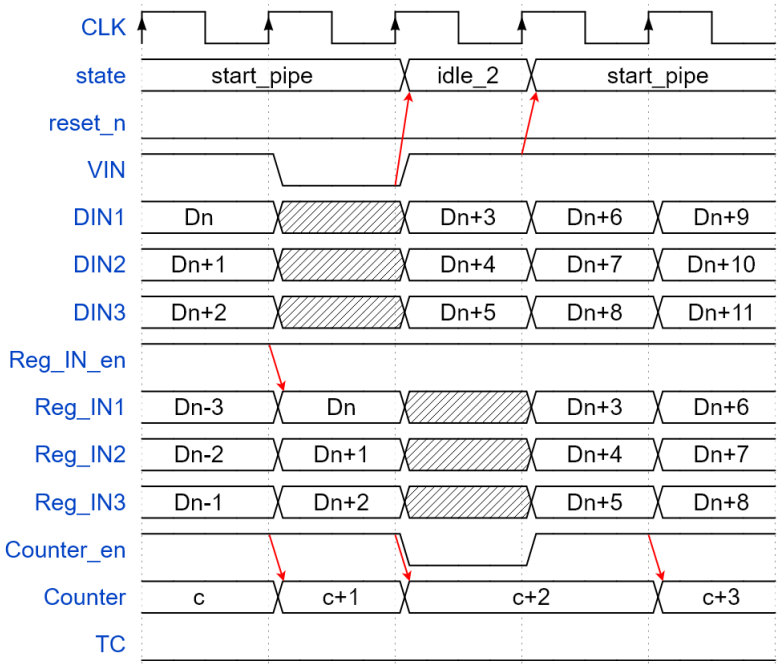

Figure 34: Advanced FIR timing: VIN goes to '0' in a generic cycle during START_PIPE.

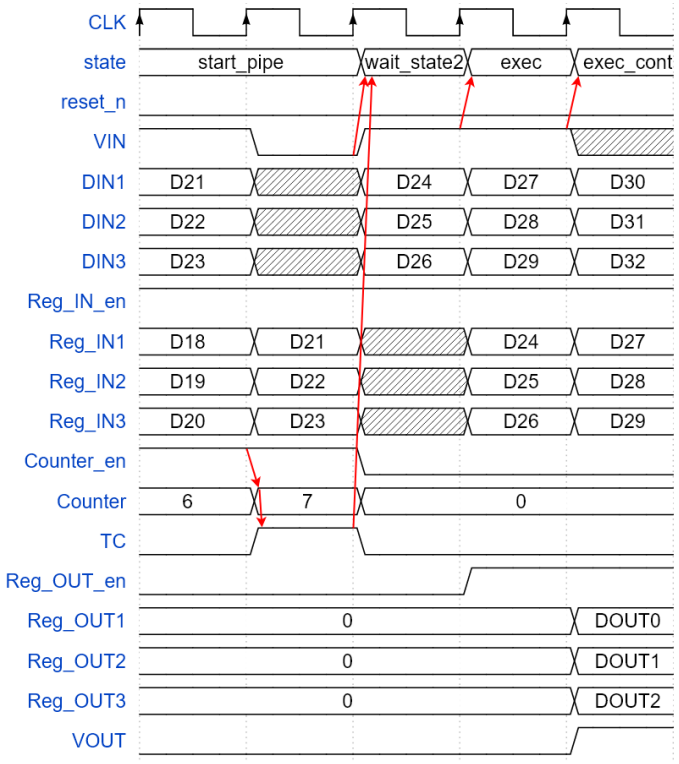

Figure 35: Advanced FIR timing: VIN goes to '0' and TC='1' during last START_PIPE cycle.

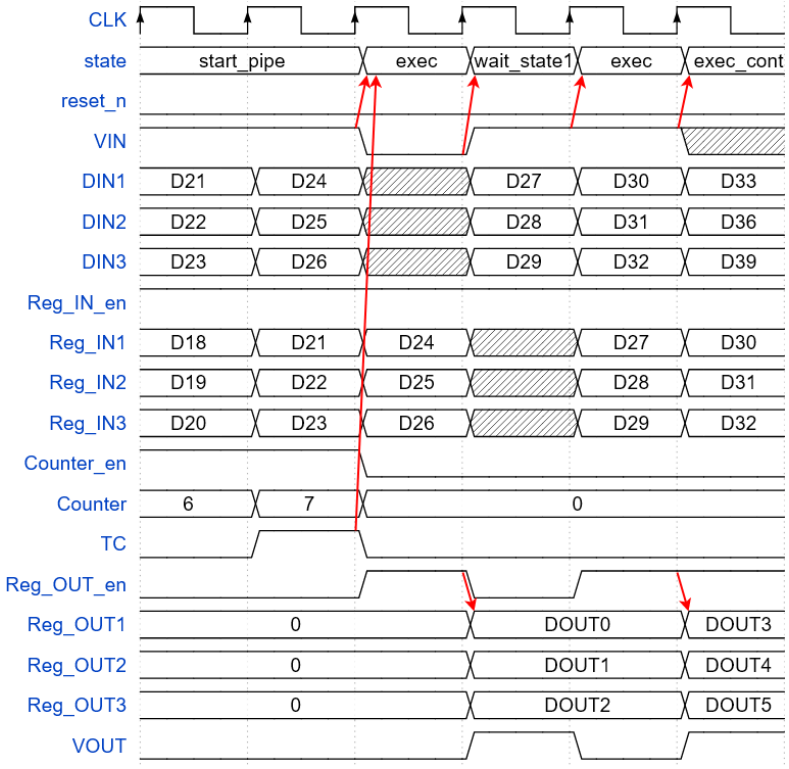

Figure 36: Advanced FIR timing: VIN goes to '0' during the EXEC state.

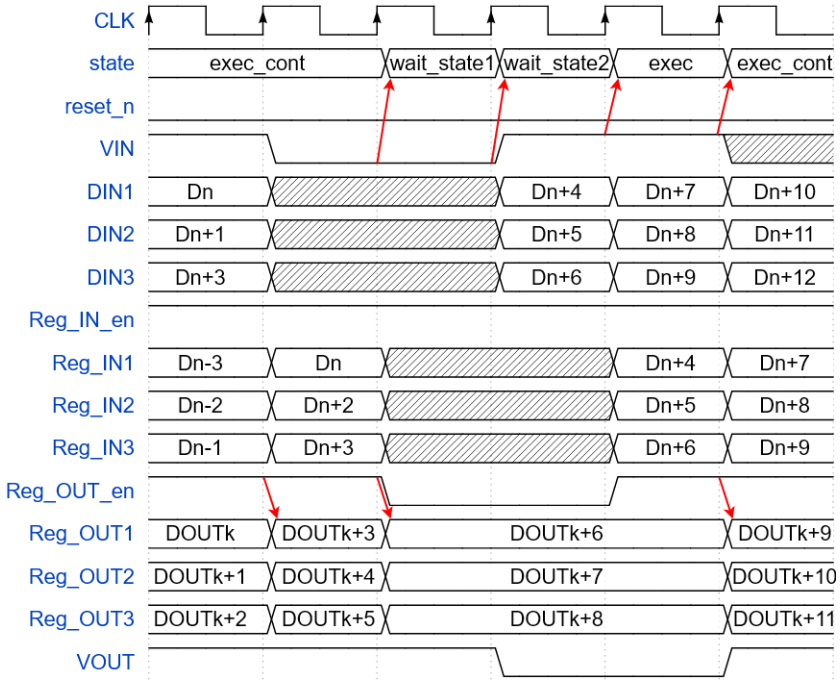

Figure 37: Advanced FIR timing: VIN goes to '0' for two clock cycles starting from the EXEC_CONT state.

3.1.5 Top-level view and VHDL description

In order to obtain the final functional block, the datapath and the FSM are connected together as shown in figure 38.

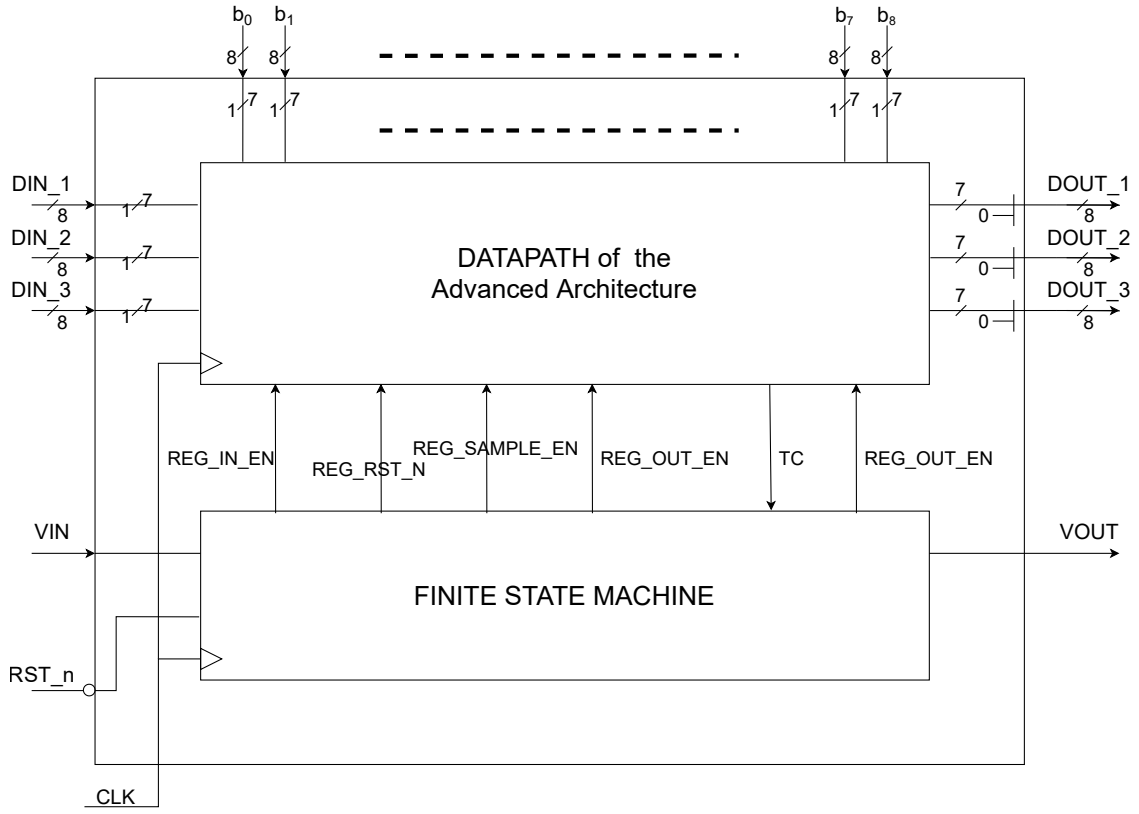

Figure 38: Optimized FIR filter

All the blocks are described in VHDL language. The datapath is defined in a structural architecture exploiting **generate** statements in order to simplify most of the signals and of the components instances. In addition, **array** data types are used in order to collect the sample data signals, the adders and the multipliers' outputs in each unfolded block. The CU is described using a three-process state machine, one of which is a synchronous process that updates the present state and the other two are combinatorial processes, which implement the output network and the future state network.

3.2 Simulation

All the simulations are performed with the same method described in section 2.2. In short, a test-bench is employed using three VHDL entities, **clk_gen**, **data_maker** and **data_sink**, which handle the signals to be fed to or read from the device-under-test. A Verilog top-level module is used to collect all of these entities together.

One main difference with the previous implementation is the fact that the input

and the output files have to be read or written by three values per clock cycle. In order to achieve this, the **data_maker** and **data_sink** entities were modified accordingly. In addition, the output file composed in this way must be equal to the one generated by the model developed in C language. Also in this case the Python script *output_check.py* is used as before. This validation routine was followed after each design step in order to assure that also after the synthesis and the place & route steps the circuit is still working.

The simulations are performed using ModelSim Mentor with the same procedure of the first design. For the advanced architecture as well, each output file resulted to be matching the output of the C model.

In order to verify the correct functionality in case VIN signal goes to logic '0', several simulation were performed, testing all the possible conditions. Two examples are shown in figures 39 and 40. In the first one VIN is equal to '0' for one clock cycle during the EXEC state, while in the latter, VIN is negated for two clock cycles starting from the EXEC_CONT state. These two cases correspond to the ones shown in the timings in figures 36 and 37 respectively.

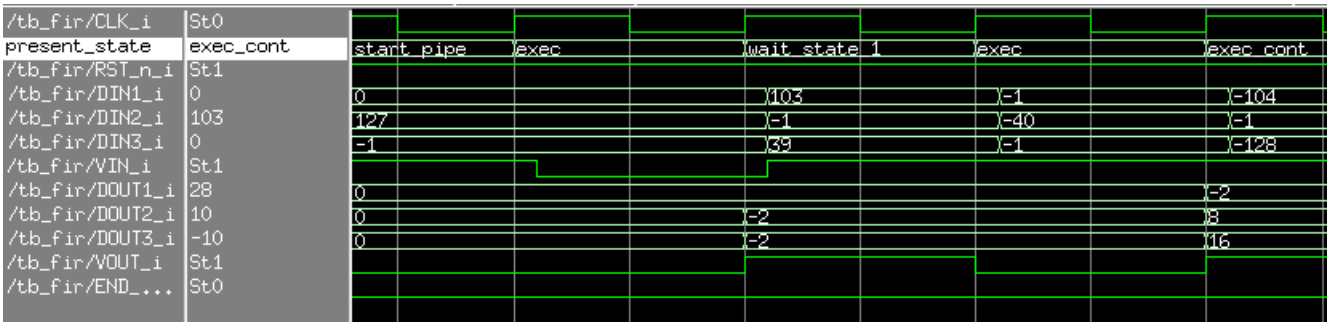

Figure 39: Advanced FIR simulation: VIN goes to '0' during the EXEC state.

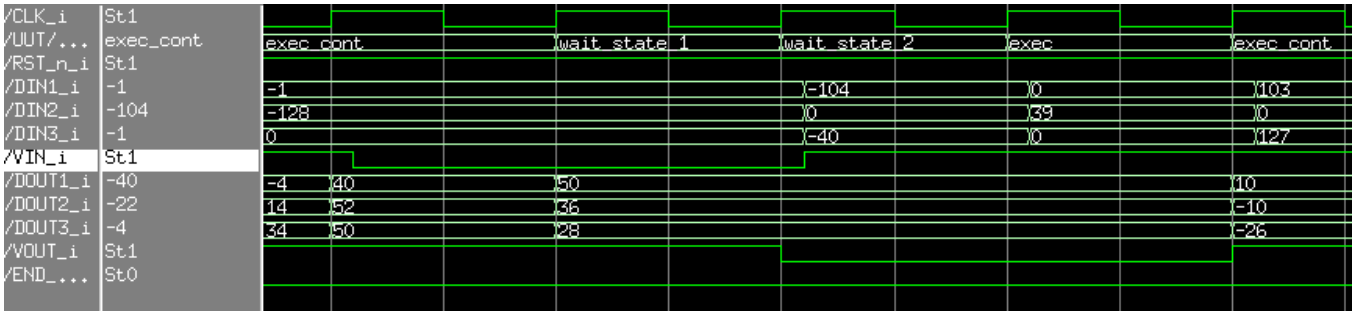

Figure 40: Advanced FIR simulation: VIN goes to '0' for two clock cycles starting from the EXEC_CONT state.

3.3 Implementation

3.3.1 Logic synthesis

Like it was done for the serial implementation, now the synthesis is performed for the optimized architecture following the same steps as before.
The first one is to impose a default clock period equal to $0\,ns$. The critical path of the system is evaluated doing the command *report_timing*, which gives back some parameters useful to characterize the architecture: *data required time*, *data arrival time* and *slack*. In this case the slack corresponds to -0.99, that implies a constraint *VIOLATED*, so it is necessary to increase the clock period until the slack became equal to 0.

After different attempts, it is possible to find the minimum clock period which is equal to $1.32\,ns$, corresponding to the maximum clock frequency of $757.58\,MHz$, figure 41. In this case, a clock frequency larger than the one of the serial implementation was found. In fact, the parallel one has a higher speed and a greater throughput due to the application of the pipeline technique, which makes the critical path as short as possible.

```
clock MY_CLK (rise edge)                                  1.32       1.32
clock network delay (ideal)                               0.00       1.32
clock uncertainty                                        -0.07       1.25
datapath/m1_reg_4/q_reg[6]/CK (DFF_X1)                    0.00       1.25 r
library setup time                                       -0.04       1.21
data required time                                                   1.21
-------------------------------------------------------------------------
data required time                                                   1.21
data arrival time                                                   -1.21
-------------------------------------------------------------------------
slack (MET)                                                          0.00
```

Figure 41: Timing report with clock period of $1.32\,ns$

The total area corresponding to the synthesis obtained with the maximum clock frequency was found with the command *report_area*, which is equal to $11621.80\,\mu m^2$, figure 42. In this case, it is possible to see that the obtained area is $A_{parallel} > 3 \cdot A_{serial}$, because now the structure is multiplied by 3 due to the order $N = 3$ of the unfolding technique used to obtain the new architecture. Also, an addition overhead is present due to the applied pipeline technique, that reduces at minimum the critical path but increases a lot the total area with the registers added in the structure, figure 30.

Then, the clock period is set to $5.28\,ns$, corresponding to the clock frequency $f_M/4 = 189.39\,MHz$, as it is required from the specification. To verify that the filter is still working properly after the synthesis, it is necessary to start a new simulation using the *Verilog* test-bench. As expected, the output results obtained are the same of the VHDL code.

```
Number of ports:                         3126
Number of nets:                         10853
Number of cells:                         7228
Number of combinational cells:           6400
Number of sequential cells:               650
Number of macros/black boxes:               0
Number of buf/inv:                       1576
Number of references:                       2

Combinational area:               8670.004010
Buf/Inv area:                      919.030000
Noncombinational area:            2951.801896
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                 11621.805907
Total area:                 undefined
```

Figure 42: Report area optimized design

In order to calculate the power consumption of the filter, it is necessary to extract the information about the switching activities of the nodes from ModelSim and evaluate them with Synopsys, so the two programs must be used together. The data are computed using the *report_power* at the frequency $f_M/4$.

Power report at f/4

```
Cell Internal Power  =   2.2537 mW
Net Switching Power  =   1.2169 mW
                         ---------
Total Dynamic Power    =   3.4706 mW
Cell Leakage Power     = 237.1445 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power
--------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000
memory             0.0000           0.0000            0.0000             0.0000
black_box          0.0000           0.0000            0.0000             0.0000
clock_network      0.0000           0.0000            0.0000             0.0000
register       1.2043e+03         243.7498         5.1538e+04         1.4996e+03
sequential         0.0000           0.0000            0.0000             0.0000
combinational  1.0494e+03         973.1591         1.8561e+05         2.2082e+03
--------------------------------------------------------------------------------
Total          2.2537e+03 uW     1.2169e+03 uW     2.3714e+05 nW     3.7077e+03 uW
```

It is possible to see in evidence that the *Total Dynamic Power* $= 3.47\,mW$, due to the sum of the *Cell Internal Power* and the *Net Switching Power*. There is also an amount of *Cell Leakage Power* $= 237.14\,\mu W$. From this two contributions, the total power was derived as:

$$Total\ Power = Total\ Dynamic\ Power + Cell\ Leakage\ Power = 3.71\,mW.$$

As expected, the total power is higher than the value obtained in the serial implementation, because the new structure, implemented with the unfolding and pipeline methodologies, request a higher number of items in exchange of better performance.

3.3.2 Place & Route

Now the place and route of the structure is performed as it was done in the serial implementation. It is necessary to follow the various steps used before, highlighted in the figures below.

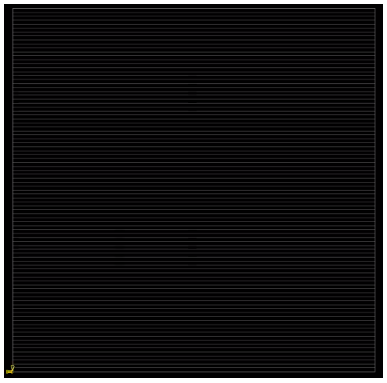

Figure 43: Design area

Figure 44: Floorplan

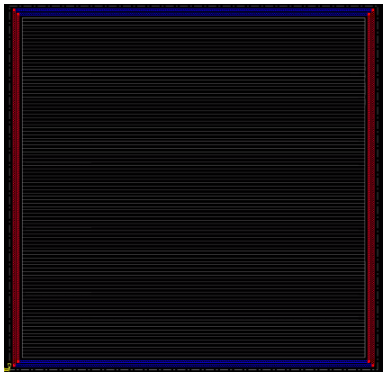

Figure 45: Power rings

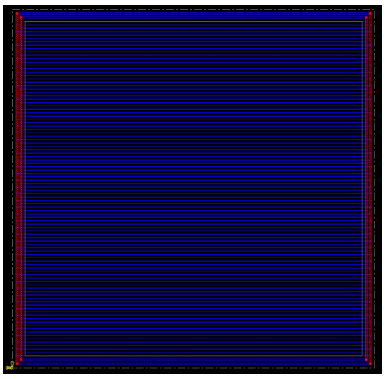

Figure 46: Power routing

Once the design is completed, it can be optimized in order to achieve the required timing constraints using the *Post-Route optimization*, figure 51.

As the different figures illustrate, the place and route of the parallel implementation is more dense compared to the serial one, due to more interconnections and more components in the structure.

Finally, it is possible to proceed with the timing analysis that are performed both for the Setup and the Hold time. The slack was always positive, so the timing constraints are respected.

At last, both the connectivity and the geometry were verified to have no violations.

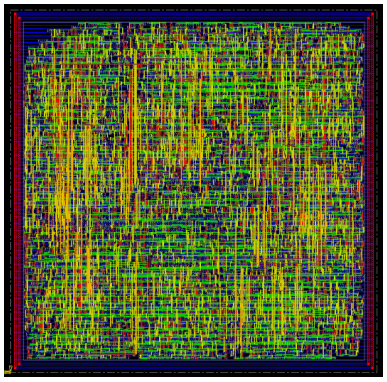

Figure 47: Placement

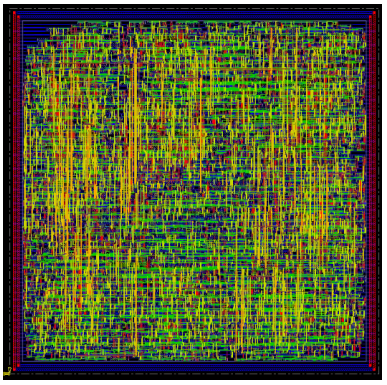

Figure 48: Clock-Tree-Synthesis

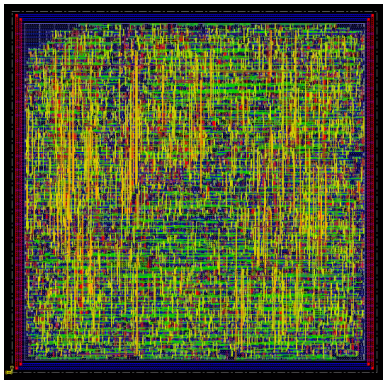

Figure 49: Place filler

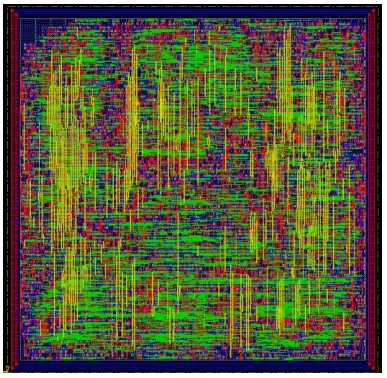

Figure 50: Routing

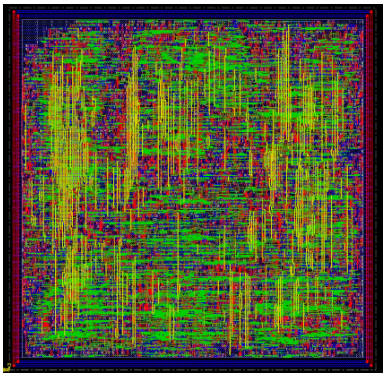

Figure 51: Post routing optimization

The area and gate count data are computed using the *Gate Count* option in the Innovus tool, figure 52. The $Area = 11078.4\,\mu m^2$ is smaller than the one evaluated during the Synthesis, due to a greater accuracy of the estimation after the place and route, but obviously it is larger than the one evaluated for the serial implementation.

The last step is the power estimation after the place and route. It is possible to proceed with the same method of the previous implementation. The report below shows the results obtained.

Verify Connectivity Advanced Architecture

```
******** Start: VERIFY CONNECTIVITY ********
Design Name: FIR_opt
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (150.1000, 148.6800)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Tue Nov 10 17:48:24 2020
Time Elapsed: 0:00:13.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
```

Verify Geometry Advanced Architecture

```
*** Starting Verify Geometry (MEM: 1246.6) ***

**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: 38.00
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**********
```

```
Gate area 0.7980 um^2
Level 0 Module FIR_opt
Gates= 13882 Cells= 6345 Area= 11078.4 um^2
```

Figure 52: Area optimized after post route

In evidence, there are different contributions of power: the *Total Dynamic Power* = $5.369\,mW$, due to the sum of the *Total Internal Power* and the *Total Switching Power*, and the *Cell Leakage Power* = $0.222\,mW$.

report_power_post_route.txt

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:        3.42835516     61.3092%
Total Switching Power:       1.94145905     34.7191%
Total Leakage Power:         0.22209484      3.9717%
Total Power:                 5.59190906
-----------------------------------------------------------------------------------------

Group                            Internal   Switching     Leakage       Total  Percentage
                                 Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                          1.457      0.2838      0.05154       1.793       32.06
Macro                                   0           0            0           0           0
IO                                      0           0            0           0           0
Combinational                       1.971       1.658       0.1706       3.799       67.94
Clock (Combinational)                   0           0            0           0           0
Clock (Sequential)                      0           0            0           0           0
-----------------------------------------------------------------------------------------
Total                               3.428       1.941       0.2221       5.592         100
-----------------------------------------------------------------------------------------

Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                       1.1      3.428       1.941       0.2221       5.592         100
```

From this two contributions, the total power was derived as:

$$\begin{aligned} Total\ Power = & Total\ Internal\ Power + Total\ Switching\ Power + \\ & + Total\ Leakage\ Power = 5.592\,mW \end{aligned}$$

This value is greater than the *Total Power* estimated after the synthesis, because the evaluation after the place and route is more accurate and it also takes into account the power of the interconnections. The *Total Power* for the unfolded and pipelined structure is larger than the previous contribution of the serial implementation, due to a greater number of components and interconnections used.

4 Conclusions

Two possible architectures for a fixed-point FIR filter of order 8 were designed with external parallelism of 8 bit. The evaluation of the THD using MATLAB tool led to a reduction of the internal representation on 7 bits in order to optimize the required area. The first design was a straightforward implementation of the FIR architecture, while the second one was obtained with the 3-unfolding and pipelining techniques. VHDL language is used to describe them. ModelSim simulations were used to verify the functionality of the circuits by comparing the results with a model in C language. In addition, the possible VIN transactions were also taken into account. Then, they were synthesized with Synopsys Design Compiler according to the given specifications, estimating timing, area and power parameters. Lastly, Cadence Innovus was used in order to perform the place and route, evaluating the area and the power consumption for both implementations.

The results obtained for the basic FIR filter and the advanced architecture are reported in the table 3. Here, the parameters post synthesis and post place and route for both designs are shown.

Version	clock period ns	Freq_max/4 MHz	Area μm^2	Throughput $Msamples/s$	Total Power mW
FIR filter Synopsys	7.52	133	3115.7	133	0.552
FIR filter Innovus	7.52	133	3064.1	133	1.005
Advanced filter Synopsys	5.28	189.4	11621.8	568.2	3.708
Advanced filter Innovus	5.28	189.4	11078.4	568.2	5.592

Table 3: Summary of the results

It is interesting to notice that the area has increased by a factor of 3.62, the throughput by 4.27 and the power consumption by 5.56.