Politecnico di Torino

Integrated Systems Architecture

Report laboratory 1
Design and implementation of a digital filter

Students: Group 40

Menditto Gianluca	**274913**
Orabona Michelangelo	**275289**
Santoro Andrea Giovanni	**274916**
Virgilio Raffaele	**274914**

Link repository github: `https://github.com/Group40ISA/ISA`

Contents

1 Design and implementation of a digital filter

1.1 Reference model development

The goal of the report is to describe the necessary steps to design a digital filter with a **cut-off frequency** of 2 kHz and a **sampling frequency** of 10 kHz. According to the rules described in the general description of the laboratory: the group number is even, so $p = 0$ and the filter will be an IIR with order and number of bits given by the two following expression, where $x = 8$ and $y = 7$:

$$N = 2^p[(x \; mod \; 2) + 1] + 6p = 1$$

$$n_b = (y \; mod \; 7) + 8 = 8$$

The given Matlab scripts give the filter coefficients and a reference output to test the results of the filter described in C; the coefficients are:

$$b_1 = b_0 = 53 \quad a_0 = 128 \quad a_1 = -21$$

The coefficient a_0 will be neglected since in fractional point representation will be equal to one. The data flow graph of the IIR filter in **direct form II** is presented in figure 1, as described in the following equation:

$$y[n] = b_0 w[n] + b_1 w[n-1]$$

$$w[n] = x[n] - a_1 w[n-1]$$

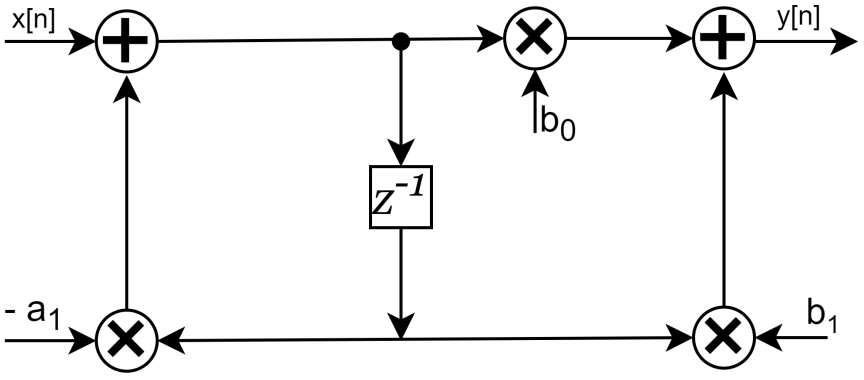

Figure 1: Data Flow Diagram of the designed IIR filter in direct form II. The Z^{-1} is the unitary delay element

After checking the outputs of the filter described by C program, the **Total Harmonic Distortion** (THD) is evaluated (using the Matlab function thd()) and it is equal to:

$$THD = -40.118 \; dB$$

1.2 VLSI implementation

First of all, the design flow starts with the implementation of the architecture at infinite resources to obtain the upper bound of the accuracy of the filter. Given the THD specification that must be maximum $-30\,dB$, the goal is to minimize the area and this can be done by reducing the internal data representation.
To do that, the partial results of the structure of the C model are analyzed. These latters lead to a change of the internal parallelism of the structure, which decreases from the 16 bit required by the infinite resources implementation to 9 bit.
This is because some partial results exceed the maximum number representable by 8 bits, so it was necessary to add a guard bit to guarantee correct computation of the filter. To validate as said before, the simulation will be performed both with maximum and minimum parallelism, and also the synthesis, to have a comparison between the two implementation and to appreciate the saved area.
In this section some interesting key points are explained in details, while the whole VHDL implementation is reported in Appendix A.1.
In according to the following figure:

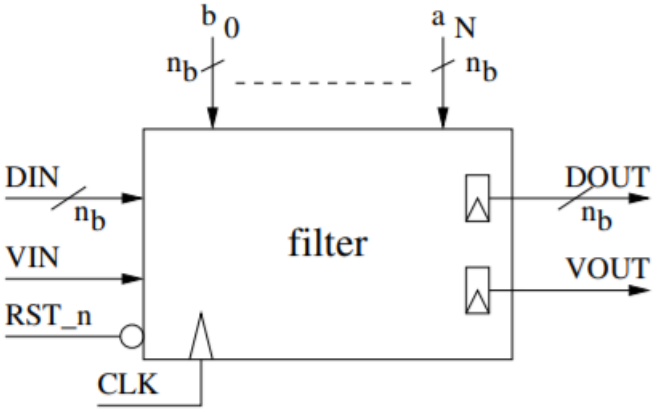

Figure 2: Filter interface

the samples (DIN) are generated one each clock cycle with a validation signal (VIN). When VIN ='1', a new sample is loaded into the architecture, in fact this input is used into load enable port of the input register. The output DOUT contains the result of the filtering and it is correct when VOUT='1', this is available after one clock cycle of latency from the corresponding entering sample. To uncouple the structure with the external, two barrier registers are used: one for DIN and another for DOUT. The signal VIN and VOUT have a temporary register to synchronize with the right timing of the data since they have an input and output register. Moreover, VIN signal, after the input register is used to disable feedback register when the input data are not valid so to not compromise the internal result.

Since a given parallelism in output and input is required, but internally it is needed to work with one more bit, some approximations are necessary and to understand how to do

these, a digression on fractional representation is useful: generally a number in fractional can be represented as

$$\frac{a}{2^y}$$

where a is the integer value and y is the number of bit after the decimal point. When two numbers in fractional are multiplied, it is obtained:

$$\frac{a}{2^y} \cdot \frac{b}{2^y} = \frac{ab}{2^{2y}}$$

This result means, as said before, the internal parallelism is higher and so double precision is required; in particular this leads to a change of weights of the bits, so the MSB bit needn't be taken into account in order to restore the weigths. The approach to these approximations in VHDL follows different rules from the implementation in C:

- **Multiplications:** after the multiplication we have a number represented on 17 bits, the MSB, as expressed before, and the less significant seven bits are truncated; this operation allows to have the results of the multiplications on nine bits.
- **Output:** the introduced guard bit is removed.

1.2.1 Simulation

At this point, the system described so far is simulated with *Modelsim* in order to verify its behavior. The figure 3 shows the circuit works correctly even when VIN from one goes to zero: the input data is not sampling by the input register; nevertheless all the operators inside are still working, so to avoid to produce a wrong result, also VOUT goes to low level. In fact, as seen in the figure below, DOUT contains a wrong result (35) among two correct ones but it is not present in the output file, like clear in the table below.

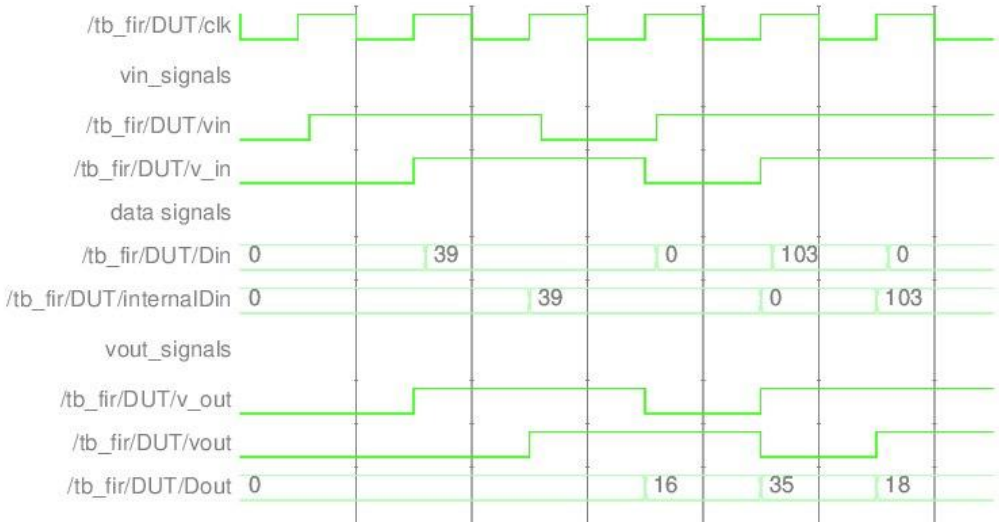

Figure 3: Correct in coming data sampling

n°	1	2	3	4	5
C	0	16	18	45	50

Table 1: Extract of first some results

In closing the results produced are compared with the ones obtained in C language, and they are the same, as required, as showed in figure 4:

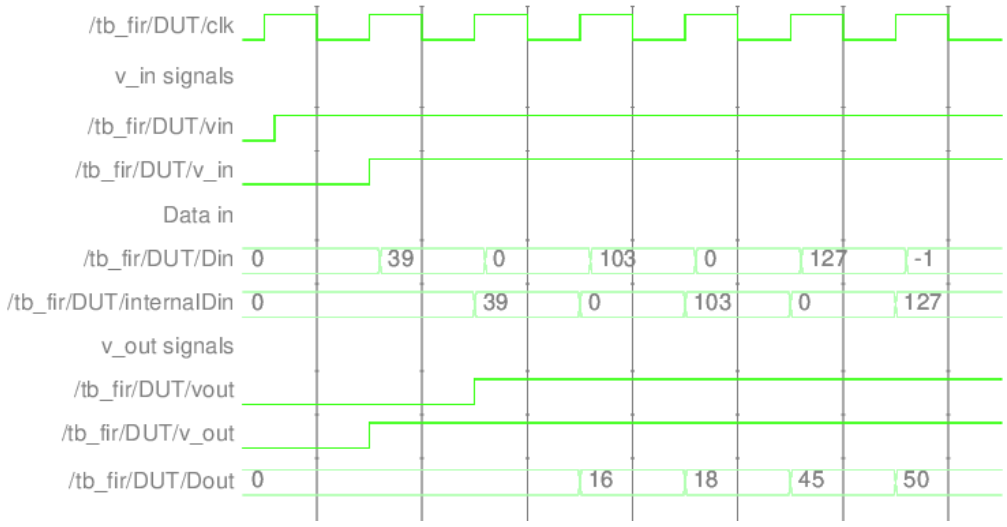

Figure 4: Some first results

1.2.2 Logic Synthesis

After verifying the correct behavior of the circuit, the next step is the synthesis. Before that, some constraints have to be applied: first of all, the clock period is set at 10 ns; then to be closer to reality, uncertainty of clock, both input and output delay and the load of each output are applied. Hence the synthesis is performed in order to find the maximum clock period, which result is showed in report extract inserted below:

```
-------------------------------------------------------------
   data required time                                 14.42
   data arrival time                                  -3.60
-------------------------------------------------------------
   slack (MET)                                        10.81
```

From this data, the maximum clock period is obtained:

$$T_{clk_{max}} = data\ arrival\ time + libray\ setup\ time = 3.60 + 0.03 = 3.63\ ns$$

and observing the specifics, the period is set as below

$$T_{clk} = T_{clk_{max}} * 4 = 14.52\ ns.$$

Backward annotation: From now on, all the analysis were done by considering this clock period, by which all the backward annotation process is done: after having analyzed elaborated and compiled the current design, the synthesizer generetes the **Standard Delay Format file** (SDF) which contains detailed information about the delay of the circuit, and the *Verilog* netlist generated with a specific technological library. Once *Modelsim* has read the .sdf file and after the simulation, it generates the **Value Change Dump** (VCD), in which are annotated all the toggles of nodes. Since *Synopsys* is able to read .saif file containing the same information of the .vcd file, we need to convert it. In the last step the circuit power is analyzed by considering the .saif file and the netlist generated before. All these steps are done automatically by using the scripts reported in appendix B.

Analysis of the results: To be able to do a comparison, the above process is performed also for the structure developed with maximum resources.

Area with reduced parallelism

```
Combinational area:          884.982005
Buf/Inv area:                 46.010000
Noncombinational area:       143.640005
Macro/Black Box area:          0.000000
```

Total cell area: 1028.622009

Area at infinite resources

```
Combinational area:         1231.580007
Buf/Inv area:                 46.550000
Noncombinational area:       175.560006
Macro/Black Box area:          0.000000
```

Total cell area: 1407.140012

Power with reduced parallelism [uW]

```
Cell Internal Power  = 87.4379
Net Switching Power  = 57.6360
```

Total Dynamic Power = 145.0740

Cell Leakage Power = 20.5863

Power at infinite resources [uW]

```
Cell Internal Power  = 122.9963
Net Switching Power  =  96.0296
```

Total Dynamic Power = 219.0259

Cell Leakage Power = 28.2748

As expected, it is obtained an area reduction about 27% than to other solution: it happens because operators and registers are more simple and so this leads to lower power dissipation, that decreases about 24%. In details, the main reduction is related to *Net switching power*, which decreases because a lower number of toggling nodes thanks to the reduced parallelism of the architecture.

1.2.3 Place & Route

Now, the place and route (PnR) is performed. At the beginning, *Innovus* set the area of the cell, and then several steps are realized to complete the process. The first one is the insertion of power rings: one for V_{dd} and one for V_{ss} and so the horizontal wires to connect them to all cells. So the placement and routing is realized, and in figure 5 is presented the final result of the PnR.

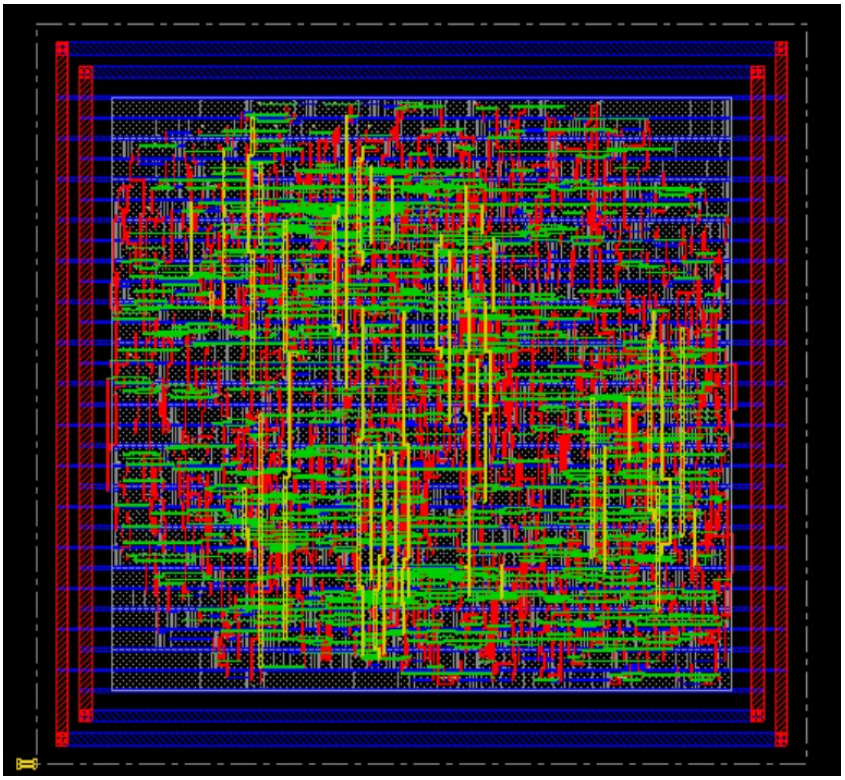

Figure 5: Final result of the PnR

At this point, the same analysis done during synthesis are repeated again, in particular the backward annotation is performed by employing *Innovus*' verilog netlist. After that, the area and power reports are obtained with the optimization made by Innovus. The extract of them are reported below:

Report area

Combinational area:	884.982005
Buf/Inv area:	46.018000
Noncombinational area:	143.640005
Macro/Black Box area:	0.000000

Total Cell Area: 1028.622009

Report power

Cell Internal Power	=	94.5434 uW
Net Switching Power	=	62.3197 uW

Total Dynamic Power = 156.8631 uW

Cell Leakage Power = 20.5695 uW

Definitively, as seen, the area obtained is the same one produced during synthesis, and instead about the estimation of power, in this case is possible to notice a tolerable increase respect to the previous one due to the higher precision in the generation of .sdf file.

1.3 Advanced architecture development

Now the goal is to improve the previous structure, increasing throughput. To achieve this purpose, the method used is the "*Look ahead*". Before applying this approach, it is useful to refer to some theoretical issues. Considering the figure 1:

$$T_{cp} = T_m + T_a$$

where T_a denotes the delay of adder and T_m the delay of multiplier; evaluating the *iteration bound*:

$$T_{inf} = T_m + T_a$$

For this reason, no improvement is possible because $T_{inf} = T_{ck}$. At this point the *look ahead method* become significant: in fact, it allows to substitute recursive equation in the main one and so, the final structure has

$$T_{inf} < T_{clk}$$

and some enhancement are possible. Let's apply the method:

$$y[n] = b_0 w[n] + b_1 w[n-1]$$

$$w[n] = x[n] - a_1 w[n-1]$$

Recursively:

$$w[n-1] = x[n-1] - a_1 w[n-1]$$

So substituting in the $y[n]$ equation:

$$y[n] = b_0 x[n] - a_1 b_0 x[n-1] + b_0 a_1^2 w[n-2] + b_1 x[n-1] - a_1 b_1 w[n-2]$$

$$y[n] = b_0(x[n] - a_1 x[n-1] + a_1^2 w[n-2]) + b_1 x[n-1] - a_1 b_1 w[n-2] \quad (1)$$

The architecture which implements the previous equation is:

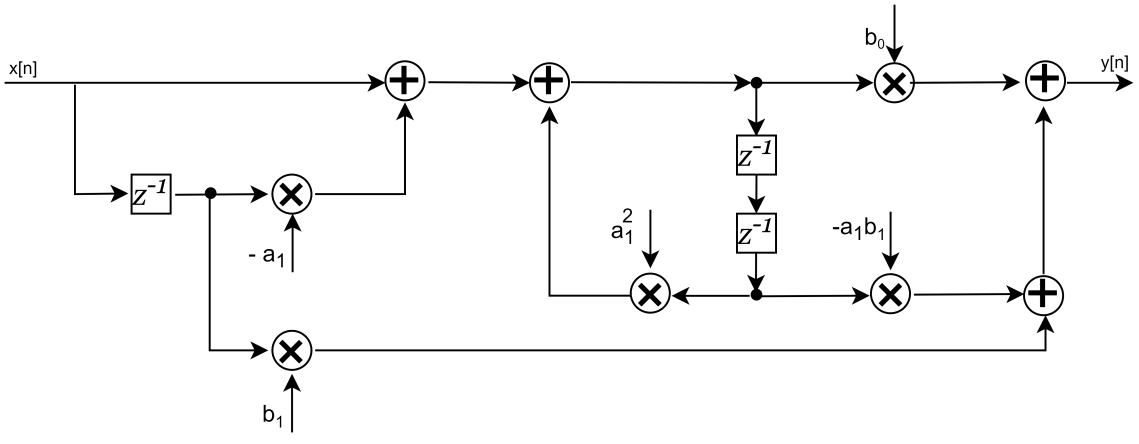

Figure 6: The structure after look ahead

In this case the iteration bound, as said before, is lower:

$$T_{inf} = (T_a + T_m)/2$$

and it means that now universal techniques can be applied.
At this point, it is need to choose the number of pipeline stages to achieve the maximum throughput. Initially, the pipe stages are inserted keeping mind to save area, so the level II, III and IV are fixed. Also, analyzing the structure, notice that there are some registers sequentially in the same branch to balance properly the delay line of all other branches.

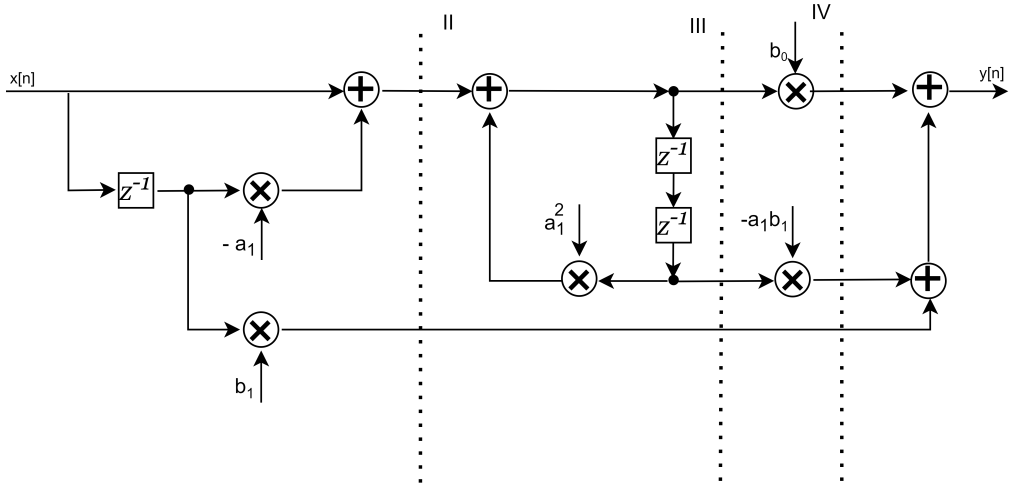

Figure 7: The structure with highlighted pipeline stage

But as showed in the figure 7, T_{inf} is still the same, and so it's possible to continue the improvement using another universal technique that is the *retiming*. Let's see how move the registers by *cut-set retiming*; a *cutset* is a group of edges in DFG which when removed from graph, they are able to divide the latter in two parts no more connected. In this case, observing the loop, it is possible to identify a cutset as showed in figure 8: in according to this approach, if a register is removed by an edge that links DFG 1 (otherwise the portion of DFG circled in red), to DFG 2 (the remaining part), the registers have to be inserted in the edges which link the DFG parts in opposite way; so it implies to insert two registers after the multiplier circled, one on the right and one on the left. The figure 9 illustrates the whole structure after retiming.

At this point, another consideration is needed; in fact now the loop is no more a critical path that instead is represented by the link between multiplier and adder on the left of the structure and by the vertical edge that links the adder to multiplier on the right of DFG (see the red edges in figure 9): for this reason it is efficient to add other two pipe stages as showed in figure 10 which allows to achieve:

$$T_{cp} = max(T_a, T_m)$$

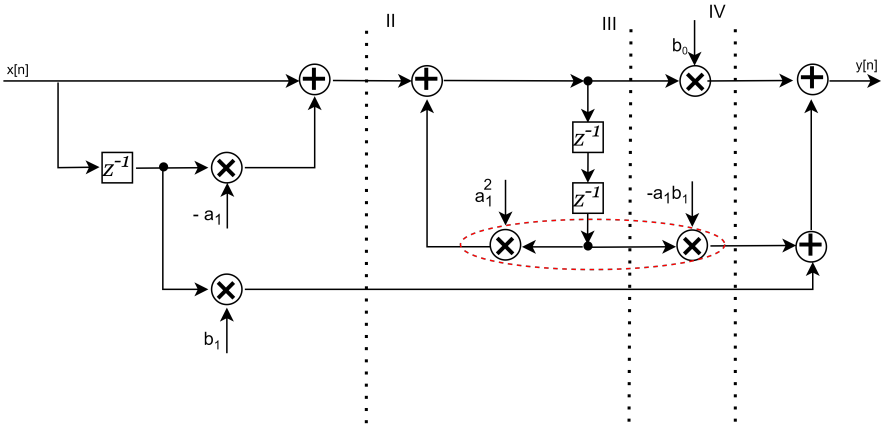

Figure 8: Cutset retiming

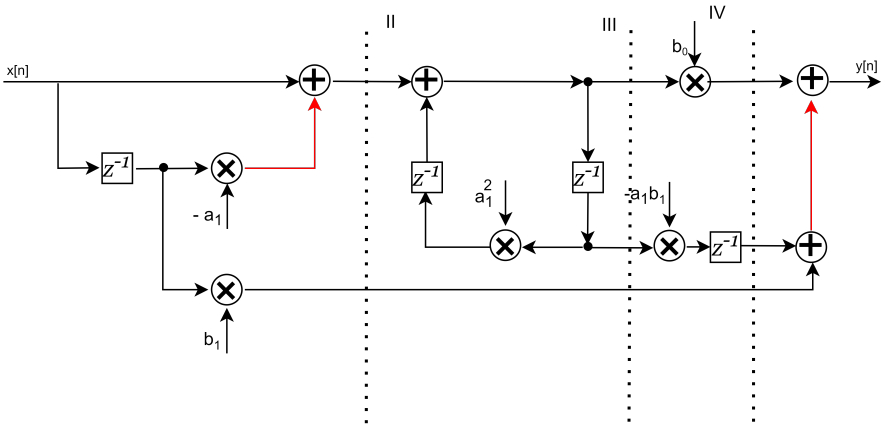

Figure 9: DFG after retiming

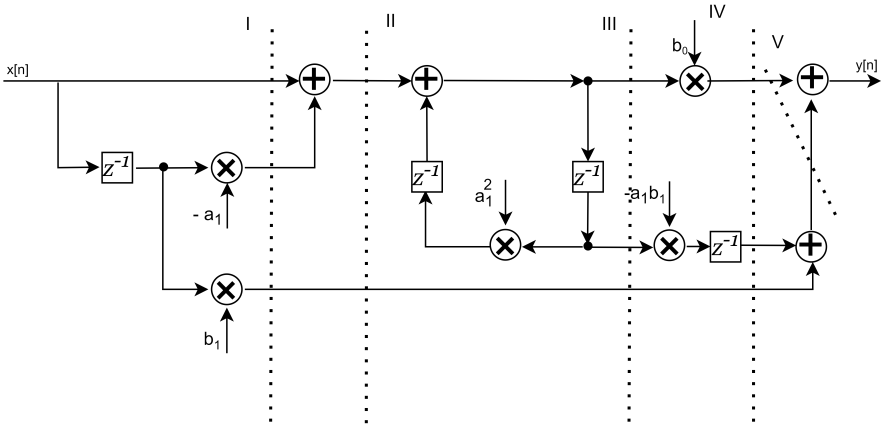

Figure 10: Final DFG

1.4 VLSI implementation

The listings of the VHDL code implementing the filter and all the required component are reported in Appendix A.1:

- **Reg & Reg_en:** they are the register used to implement the pipe level and the delay element in the structure. The registers with the enable are required to allow the unit to work well when VIN goes low.
- **Shift register:** used for the synchronization of the VOUT with the output and to drive the enable of the registers.

About the new structure, there are some considerations to do. At first, the interface, as required remains the same, but there are two new coefficients $(a_1^2, -a_1 b_1)$ as seen in the equation 1.
Moreover, since several pipeline registers are inserted, and they lead to synchronization problems:

- **one related to VOUT** that denotes a valid data at output, so it needs to be delayed by a shift register that implements the pipe levels (exactly five levels).
- **one related to VIN** in particular when it toggles from high level to low for a given number of clock edges, a bubble propagates in the shift register and it leads to stop properly the registers, to avoid them to be corrupted by incorrect data. To optimize the area, the stopped registers are only the feedback registers;

Regarding the internal parallelism of the structure, reproducing the same procedure done for the characterization of the previous filter, an internal parallelism of 9 bit is obtained, so truncations at the output of the multiplier and at the output of the filter are required (done with the same procedure described before).

1.4.1 Simulation

At this point, the system described so far is simulated with Modelsim in order to verify its behavior. The figure 11 shows the correct behavior of the filter, and also some results.
As clear the result of the structure are slightly different from the previous one, that's due to the necessary approximation for the new coefficients. To do a proper comparison with a C model, the given one is modified to implement the same equation obtained by the Look Ahead method. (The code is reported in the appendix C.1).In general the structure lead to a THD far above the specification:

$$THD = -42.82dB$$

In the figure 12 it is possible to see how the system works properly even if VIN goes low for one clock edges, as expressed before.

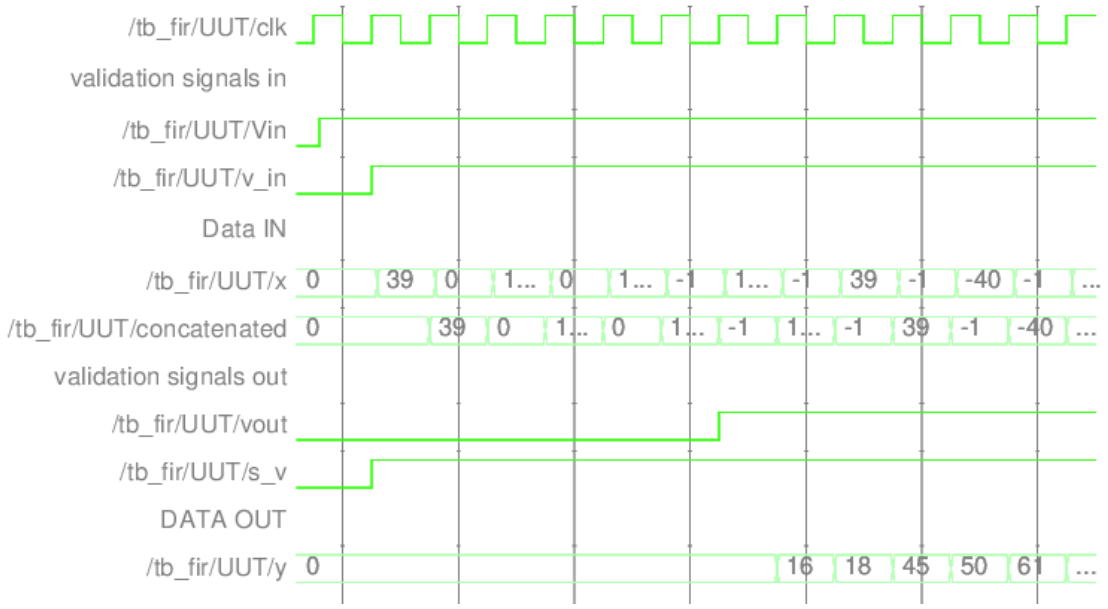

Figure 11: Simulation of the first few cycles of the filter, the output in this case is denoted by 'y'

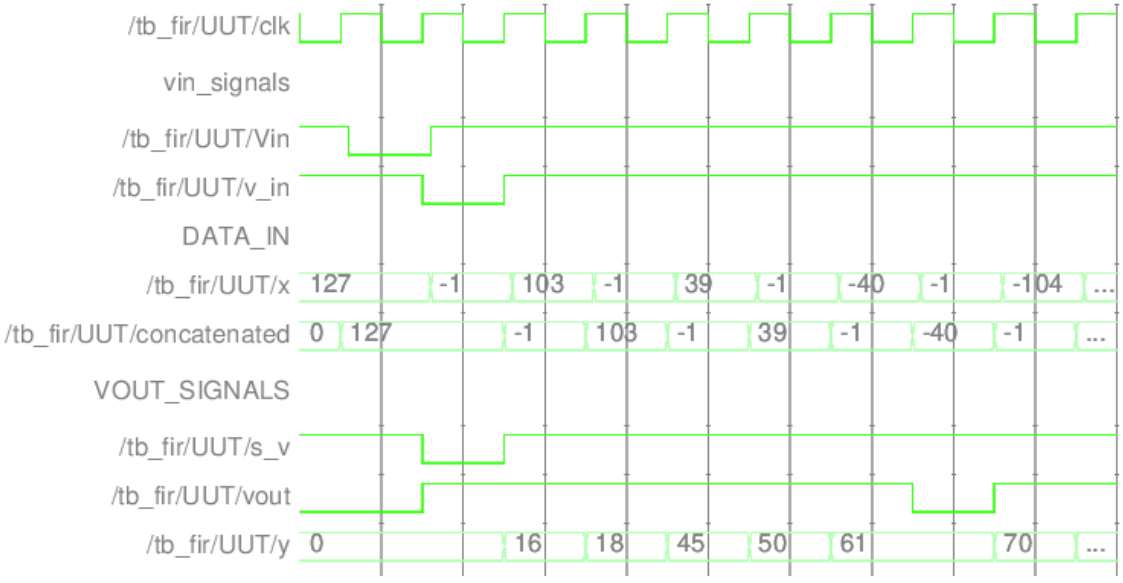

Figure 12: Simulation of the filter when VIN goes low for some clock edges

1.4.2 Logic Synthesis

Once again, the next step is the synthesis of the filter. Before all, it's set the clock period, used for the synthesis at 10 ns, and with this analysis it's obtained the minimum clock period achievable by this structure, given by the sum of the data arrival time and the library set up time.

```
-------------------------------------
data required time                9.89
data arrival time                -2.24
-------------------------------------
slack (MET)                       7.65
```

As expected the maximum achievable frequency increases by 37% with respect to the previous structure, since the critical path is reduced.
Applying the same constraints on the clock period ($4 * T_{min}$) the backward annotation process is performed, leading to the following results concerning area and power consumption:

Report area

```
Combinational area:      1535.884000
Buf/Inv area:              67.032001
Noncombinational area:    936.320030
Macro/Black Box area:       0.000000
```

Total Cell Area: 2472.204036

Report power

```
Cell Internal Power =  242.7182 uW
Net Switching Power =   99.0343 uW
```

Total Dynamic Power = 341.7525 uW

Cell Leakage Power = 47.5412

As expected the performance is at the expense of area and power consumption: the area has more than doubled, compared to the previous implementation, in fact it increases by 147% due to the more components used, which are pipe registers, multiplier, adder.
About the power there is an increasing of the *total dynamic power* by 135% and *cell leakage power* by 130%, these are due to the several toggling nodes in the structure for dynamic power, and more operator used relative to the leakage.

1.4.3 Place & Route

Finally the PnR is done, leading to the layout presented in the figure below:

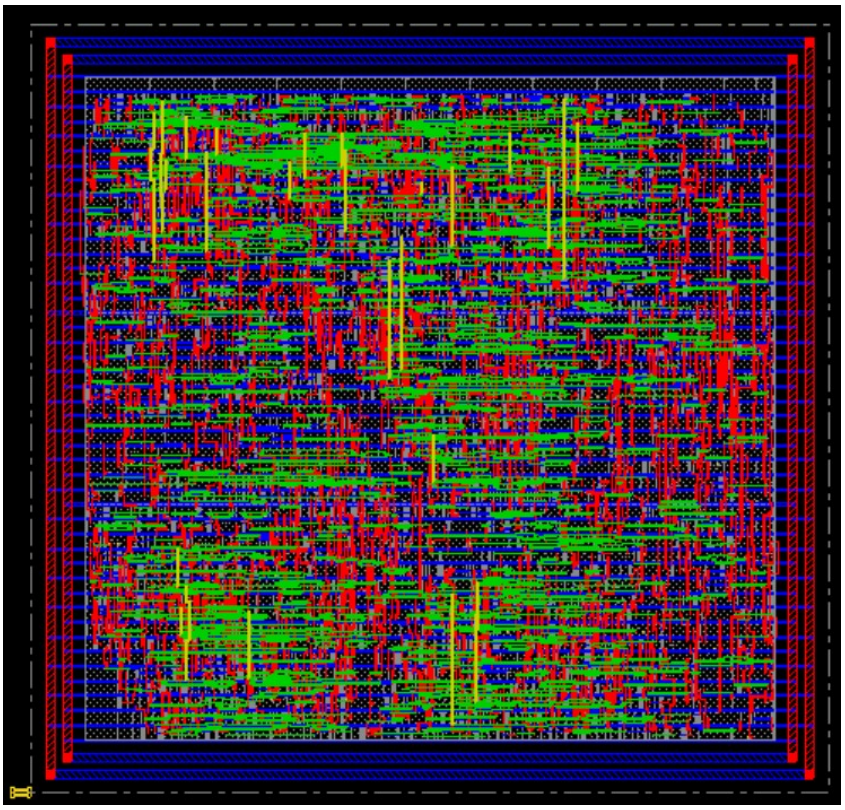

Figure 13: Layout of the IIR filter optimized by the Look Ahead method

The backward annotation process is then repeated by using the .sdf file generated by *Innovus* and the report about the area and the power consumption are showed below:

Report area

```
Combinational area:      1535.884000
Buf/Inv area:              67.032001
Noncombinational area:    936.320030
Macro/Black Box area:       0.000000
```

Total Cell Area: 2472.204036

Report power

```
Cell Internal Power = 260.1618 uW
Net Switching Power = 110.4398 uW
```

Total Dynamic Power = 370.6016 uW

Cell Leakage Power = 47.4530

As we can see, and for the same reasons described in the previous PnR section, we have an increase in power by 9% with respect to the report power done by *Synopsys*.

A VHDL listing

A.1 Listing of the 1st order IIR not optimized

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filter is
  port(
    clk, rst    : in  std_logic;
    vin         : in  std_logic;
    Din, b0, a1 : in  signed(7 downto 0);
    vout        : out   std_logic;
    Dout        : out signed(7 downto 0)
  );
end entity filter;

architecture RTL of filter is

  signal m1, m2, ff             : signed(8 downto 0);
  signal m1_tmp, m2_tmp, ff_tmp : signed(16 downto 0);
  signal tmpfb, fb              : signed(8 downto 0);
  signal w, s1                  : signed(8 downto 0);
  signal v_out, v_in, tmprvin   : std_logic;
  signal tmpDin                 : signed(7 downto 0);
  signal tmpDout                : signed(7 downto 0);
  signal internalDin            : signed(8 downto 0);
  signal extension              : signed(0 downto 0);
  signal tmprv_out              : std_logic;
begin

  registerrvout : process(clk)
  begin
    if rst = '0' then
      tmprv_out <= '0';
    elsif (clk'event and clk = '1') then
      tmprv_out <= v_out;
    end if;
  end process registerrvout;
  vout <= tmprv_out;

  registerrvi : process(clk)
  begin
    if rst = '0' then
      tmprvin <= '0';
    elsif (clk'event and clk = '1') then
      tmprvin <= vin;
    end if;
  end process registerrvi;
  v_in <= tmprvin;

```

```
49  registerfb : process(clk)
50  begin
51    if rst = '0' then
52      tmpfb <= (others => '0');
53    elsif (clk'event and clk = '1') then
54      if v_in = '1' then
55        tmpfb <= w;
56      end if;
57    end if;
58  end process registerfb;
59  fb <= tmpfb;
60
61  registersDin : process(clk)          --@suppress
62  begin
63    if rst = '0' then
64      tmpDin <= (others => '0');
65    elsif (clk'event and clk = '1') then
66      if vin = '1' then
67        tmpDin <= Din;
68      end if;
69    end if;
70  end process registersDin;
71  -- sign extension to conform the parallelism of the
72  -- input to the parallelism of the internal repres.
73  extension   <= (others => tmpDin(7));
74  internalDin <= extension & tmpDin;
75
76  registersDout : process(clk)         --@suppress
77  begin
78    if rst = '0' then
79      tmpDout <= (others => '0');
80    elsif (clk'event and clk = '1') then
81      tmpDout <= s1(7 downto 0);
82    end if;
83  end process registersDout;
84  Dout <= tmpDout;
85
86  ------------------------------------------------------
87  -- This blocks the output whenever
88  -- the input are not correct, since the block
89  -- after samples only when Vout = 1.
90
91  v_out  <= v_in;
92  ------------------------------------------------------
93  -- the implemented equations are:
94  -- y = b0*w(n) + b1*w(n-1)
95  -- w = x(n) - a1*w(n-1)
96  ------------------------------------------------------
97  m1_tmp <= fb * a1;              --result moltiplication need 17 bit
98  m1     <= m1_tmp(15 downto 7); --restoring on 9 bit the result. It's 15
     downto 7
99  w      <= internalDin - m1;     -- and not 16 downto 8 because the dot
```

```
    shift
100   ff_tmp <= w * b0;              -- of 1 position more (e.g.
       1.00*1.00=11.0000)
101   m2_tmp <= fb * b0;
102   m2     <= m2_tmp(15 downto 7);
103   ff     <= ff_tmp(15 downto 7);
104   s1     <= ff + m2;
105   ----------------------------------------------------
106
107 end architecture RTL;
```

A.1.1 Listing of the 1st order IIR with LookAhead

Listings of the registers without enable

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  -- This is a register whitout load enable signal --
5  entity reg is
6    generic(parallelism : integer := 9);
7    port(
8      clk : in  std_logic;
9      rst : in  std_logic;
10     D   : in  signed( parallelism - 1 downto 0);
11     Q   : out signed( parallelism - 1 downto 0)
12   );
13 end entity reg;
14
15 architecture RTL of reg is
16
17   signal tmp : signed( parallelism - 1 downto 0);
18
19 begin
20   regprocess : process(clk, rst)      --@suppress
21   begin
22     if rst = '0' then
23       tmp <= (others => '0');
24     elsif clk'event and clk = '1' then
25         tmp <= D;
26     end if;
27   end process;
28   Q <= tmp;
29 end architecture RTL;
```

Listings of the registers with enable

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  -- This is a register that has a load enable signal to save the data --
5  entity reg_en is
6    generic(parallelism : integer := 9);
```

```
  port(
    clk : in  std_logic;
    rst : in  std_logic;
    D   : in  signed(parallelism - 1 downto 0);
    Q   : out signed(parallelism - 1 downto 0);
    en  : in  std_logic
  );
end entity reg_en;

architecture RTL of reg_en is

  signal tmp : signed(parallelism - 1 downto 0);

begin
  regprocess : process(clk, rst)      --@suppress
  begin
    if rst = '0' then
      tmp <= (others => '0');
    elsif clk'event and clk = '1' then
      if en = '1' then
        tmp <= D;
      end if;
    end if;
  end process;
  Q <= tmp;
end architecture RTL;
```

Listings of the shift register used to synchronize V_{out}

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity shift is
  generic(levelPipe : integer := 5);
  port(
    clk : in  std_logic;
    rst : in  std_logic;
    D   : in  std_logic;
    Q   : out std_logic;
    enable : out std_logic_vector(1 downto 0)
  );
end entity shift;

architecture RTL of shift is

  signal tmp : std_logic_vector(levelPipe - 1 downto 0);

begin
  process(clk, rst)
  begin
    if rst = '0' then
      tmp <= (others => '0');
```

```
    elsif clk'event and clk = '1' then
          tmp(levelPipe - 2 downto 0) <= tmp(levelPipe - 1 downto 1);
          tmp(levelPipe - 1)          <= D;
    end if;
  end process;
  enable(1)<=tmp(levelPipe-2);
  enable(0)  <= tmp(1);
  Q <= tmp(0);
end architecture RTL;
```

Listings of the filter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filter is
  generic(parallelism : integer := 8);
  port(
    clk               : in  std_logic;
    rst               : in  std_logic;
    B0, A1, C, A_sqrd : in  std_logic_vector(parallelism - 1 downto 0);
    Vin               : in  std_logic;
    x                 : in  std_logic_vector(parallelism - 1 downto 0);
    y                 : out std_logic_vector(parallelism - 1 downto 0);
    vout              : out std_logic
  );
end entity filter;

architecture RTL of filter is

  ------------------------------------------------------------
  component reg is
    generic(parallelism : integer := 9);
    port(
      clk : in  std_logic;
      rst : in  std_logic;
      D   : in  signed(parallelism - 1 downto 0);
      Q   : out signed(parallelism - 1 downto 0)
    );
  end component reg;

  component reg_en is
    generic(parallelism : integer := 9);
    port(
      clk : in  std_logic;
      rst : in  std_logic;
      D   : in  signed(parallelism - 1 downto 0);
      Q   : out signed(parallelism - 1 downto 0);
      en  : in  std_logic
    );
  end component reg_en;

```

```
42    component shift is
43      generic(levelPipe : integer := 5);
44      port(
45        clk    : in  std_logic;
46        rst    : in  std_logic;
47        D      : in  std_logic;
48        Q      : out std_logic;
49        enable : out std_logic_vector(1 downto 0)
50      );
51    end component shift;
52    ---------------------------------------------------------------------
53
54    signal extension                              : signed(0 downto 0);
55    signal concatenated                           : signed(8 downto 0);
56    signal x_internal                             : signed(parallelism - 1
       downto 0);
57    signal x_minus_1                              : signed(8 downto 0);
58    signal m1_tmp, m2_tmp, m3_tmp, m4_tmp, m5_tmp : signed(16 downto 0);
59    signal m1, m2, m3, m4, m5                     : signed(8 downto 0);
60    signal s1, s2, s3, s4                         : signed(8 downto 0);
61    signal out_reg_2, out_reg_3                   : signed(8 downto 0);
62    signal out_reg_4                              : signed(8 downto 0);
63    signal out_pipe_1, out_pipe_2, out_pipe_3     : signed(8 downto 0);
64    signal out_pipe_4, out_pipe_5                 : signed(8 downto 0);
65    signal out_pipe_6, out_pipe_7, out_pipe_13    : signed(8 downto 0);
66    signal out_pipe_8, out_pipe_9, out_pipe_10    : signed(8 downto 0);
67    signal out_pipe_11, out_pipe_12               : signed(8 downto 0);
68    signal s_v, v_in, tmp_vin                     : std_logic;
69    signal y_signed                               : signed(7 downto 0);
70    signal x_signed                               : signed(7 downto 0);
71    signal enable                                 : std_logic_vector(1 downto
       0);
72    ---------------------------------------------------------------------
73  begin
74
75    in_regs : reg_en
76      generic map(
77        parallelism => 8
78      )
79      port map(
80        clk => clk,
81        rst => rst,
82        D   => x_signed,
83        Q   => (x_internal),
84        en  => Vin
85      );
86    x_signed <= signed(x);
87
88    extension(0) <= x_internal(7);
89    concatenated <= extension & x_internal;
90
91    x_n_minus_1 : reg
```

```
92      generic map(
93        parallelism => 9
94      )
95      port map(
96        clk => clk,
97        rst => rst,
98        D   => (concatenated),
99        Q   => (x_minus_1)
100     );
101
102   out_reg : reg_en
103     generic map(
104       parallelism => 8
105     )
106     port map(
107       clk => clk,
108       rst => rst,
109       D   => (s3(7 downto 0)),
110       Q   => y_signed,
111       en  => enable(0)
112     );
113   y <= std_logic_vector(y_signed);
114
115   v_out_pipe : shift
116     generic map(
117       levelPipe => 6
118     )
119     port map(
120       clk         => clk,
121       rst         => rst,
122       D           => s_v,
123       Q           => vout,
124       enable => enable
125     );
126
127   reg_2 : reg_en
128     generic map(parallelism => 9)
129     port map(
130       clk => clk,
131       rst => rst,
132       D   => (m3),
133       Q   => (out_reg_2),
134       en  => enable(1)
135     );
136   reg_3 : reg_en
137     generic map(parallelism => 9)
138     port map(
139       clk => clk,
140       rst => rst,
141       D   => (s2),
142       Q   => (out_reg_3),
143       en  => enable(1)
```

```
144     );
145   reg_4 : reg
146     generic map(parallelism => 9)
147     port map(
148       clk => clk,
149       rst => rst,
150       D   => (m4),
151       Q   => (out_reg_4)
152     );
153
154   --------Vin Register-----------------
155   reg_vin : process(clk)
156   begin
157     if rst = '0' then
158       tmp_vin <= '0';
159     elsif (clk'event and clk = '1') then
160       tmp_vin <= Vin;
161     end if;
162   end process reg_vin;
163   v_in <= tmp_vin;
164
165   ----------------------------------------
166   pipe_1 : reg
167     generic map(
168       parallelism => 9
169     )
170     port map(
171       clk => clk,
172       rst => rst,
173       D   => (concatenated),
174       Q   => (out_pipe_1)
175     );
176
177   pipe_2 : reg
178     generic map(
179       parallelism => 9
180     )
181     port map(
182       clk => clk,
183       rst => rst,
184       D   => (m2),
185       Q   => (out_pipe_2)
186     );
187
188   pipe_3 : reg
189     generic map(
190       parallelism => 9
191     )
192     port map(
193       clk => clk,
194       rst => rst,
195       D   => (m5),
```

```
196       Q   => (out_pipe_3)
197     );
198
199   pipe_4 : reg
200     generic map(
201       parallelism => 9
202     )
203     port map(
204       clk => clk,
205       rst => rst,
206       D   => (s1),
207       Q   => (out_pipe_4)
208     );
209
210   pipe_11 : reg
211     generic map(
212       parallelism => 9
213     )
214     port map(
215       clk => clk,
216       rst => rst,
217       D   => (out_pipe_8),
218       Q   => (out_pipe_11)
219     );
220
221   pipe_12 : reg
222     generic map(
223       parallelism => 9
224     )
225     port map(
226       clk => clk,
227       rst => rst,
228       D   => (s4),
229       Q   => (out_pipe_12)
230     );
231
232   pipe_5 : reg
233     generic map(parallelism => 9)
234     port map(
235       clk => clk,
236       rst => rst,
237       D   => (out_pipe_3),
238       Q   => (out_pipe_5)
239     );
240   pipe_6 : reg
241     generic map(parallelism => 9)
242     port map(
243       clk => clk,
244       rst => rst,
245       D   => (s2),
246       Q   => (out_pipe_6)
247     );
```

```
248   pipe_13 : reg
249     generic map(parallelism => 9)
250     port map(
251       clk => clk,
252       rst => rst,
253       D   => (out_reg_3),
254       Q   => (out_pipe_13)
255     );
256   pipe_7 : reg
257     generic map(parallelism => 9)
258     port map(
259       clk => clk,
260       rst => rst,
261       D   => (out_pipe_5),
262       Q   => (out_pipe_7)
263     );
264   pipe_8 : reg
265     generic map(parallelism => 9)
266     port map(
267       clk => clk,
268       rst => rst,
269       D   => (m1),
270       Q   => (out_pipe_8)
271     );
272   pipe_9 : reg
273     generic map(parallelism => 9)
274     port map(
275       clk => clk,
276       rst => rst,
277       D   => (out_reg_4),
278       Q   => (out_pipe_9)
279     );
280   pipe_10 : reg
281     generic map(parallelism => 9)
282     port map(
283       clk => clk,
284       rst => rst,
285       D   => (out_pipe_7),
286       Q   => (out_pipe_10)
287     );
288   --
      ------------------------------------------------------------------------------
289
290   m1_tmp <= out_pipe_6 * signed(B0);
291   m2_tmp <= x_minus_1 * signed(A1);
292   m3_tmp <= out_reg_3 * signed(A_sqrd);
293   m4_tmp <= out_pipe_13 * signed(C);
294   m5_tmp <= x_minus_1 * signed(B0);
295
296   m1 <= m1_tmp(15 downto 7); -- truncation of the mult
297   m2 <= m2_tmp(15 downto 7); -- to adapt with the internal
```

```
298   m3 <= m3_tmp(15 downto 7); -- parallelism
299   m4 <= m4_tmp(15 downto 7);
300   m5 <= m5_tmp(15 downto 7);
301
302   s1 <= out_pipe_1 - out_pipe_2;
303   s2 <= out_pipe_4 + out_reg_2;
304   s3 <= out_pipe_11 + out_pipe_12;
305   s4 <= out_pipe_10 - out_pipe_9;
306
307   --
        ------------------------------------------------------------------------------

308   s_v <= v_in;
309
310 end architecture RTL;
```

B Backward annotation scripts

This script (create_sdf.scr) is required to compile all the components on synopsys and to create the .sdf file and the verilog netlist, require for modelsim to annotate the activity of all the internal nodes.

```
#CREATE_SDF.SCR
variable targetcompilation "../src/filter.vhd"
variable top_hierarchy "filter"
variable clock "10.0"

analyze -format vhdl $targetcompilation
set power_preserve_rtl_hier_names true

elaborate $top_hierarchy -arch RTL -lib work > elaborate_transcript.txt
create_clock -name my_clk -period $clock 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

sh mkdir clock_$clock
report_timing > clock_$clock/report_timing.txt
report_area > clock_$clock/report_area_vhd.txt
ungroup -all -flatten
change_names -hierarchy -rules verilog
write_sdf ../netlist/$top_hierarchy.sdf
write -f verilog -hierarchy -output ../netlist/$top_hierarchy.v
write_sdc ../netlist/$top_hierarchy.sdc
```

This script (fill_forward.scr) is used by Modelsim to create the .vcd file, that needs to be converted in .saif to allow synopsys to use it and to report the precise power.

```
#FILL_FORWARD.SCR
#setting env

rm -rf ../sim/work
vlib work

# compile

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/filter.v
```

```
vlog -work ./work ../tb/tb_fir.v

vsim -L /software/dk/nangate45/verilog/msim6.2g -sdftyp /tb_fir/UUT=../
   netlist/filter.sdf work.tb_fir

#create sdf

vcd file ../vcd/filter.vcd
vcd add /tb_fir/UUT/*

# run simulation
run 3 ms
```

After converting the .vcd in .saif, this last script, uses this file in order to give accurate information on the power dissipation by the structure.

```
# BACKWARD.SCR
variable clock "10.0"

sh vcd2saif -input ../vcd/filter.vcd -output ../saif/filter.saif

create_clock -name my_clk -period $clock clk

read_verilog -netlist ../netlist/filter.v
read_saif -input ../saif/filter.saif -instance tb_fir/UUT -unit ns -scale 1

report_power > clock_$clock/report_power.txt
report_area > clock_$clock/report_area_Verilog.txt
```

These script are also used for the back annotation performed after the generation of the netlist by *Innovus*, except for create_sdf.scr replace by the following script in which the command related to the creation of the .sdf file and the netlist are removed (since they are created by innovus itself)

```
#COMPILE.SCR
variable targetcompilation "../src/filter.vhd"
variable top_hierarchy "filter"
variable clock "10.0"

analyze -format vhdl $targetcompilation
set power_preserve_rtl_hier_names true

elaborate $top_hierarchy -arch RTL -lib work > elaborate_transcript.txt
create_clock -name my_clk -period $clock 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]
```

```
19
20 compile
21
22 sh mkdir clock_$clock
23 report_timing > clock_$clock/report_timing.txt
24 report_area > clock_$clock/report_area_vhd.txt
```

C C listings

C.1 C model for L.A. implementation

```
#include<stdio.h>
#include<stdlib.h>

#define NB 8  /// number of bits
#define N  1 // order of the filter

const int b0 = 53; /// coefficient b0
const int b[N]={53}; /// b array
const int a[N]={-21}; /// a array
const int ab=a[0]*b[0] >>(NB-1);
const int a_sqrd=a[0]*a[0] >>(NB-1);

/// Perform fixed point filtering assuming direct form II
///\param x is the new input sample
///\return the new output sample
int myfilter(int x)
{
  static int cont=0;
  static int sw[2]; /// w shift register
  static int sx[1];/// shift register della x
  static int first_run = 0; /// for cleaning the shift register
  int i; /// index
  int w; /// intermediate value (w)
  int y; /// output sample
  int fb, ff; /// feed-back and feed-forward results
  FILE *fd; // this file allow to know intermedial results.
  int m1,m2,m3,m4,m5; // temponary variable.

  // opening the file
  fd=fopen("val_inter.txt","a");

  /// clean the buffer
  if (first_run == 0)
  {
    first_run = 1;
    for (i=0; i<N+1; i++)
      sw[i] = 0;
    for (i=0; i<N; i++)
      sx[i] = 0;
  }

  /// compute feed-back and feed-forward
  fb = 0;
  ff = 0;
  for (i=0; i<N; i++)
  {
    m2= ((sx[i]*a[i])     >> (NB-1));
    m3= ((sw[i+1]*a_sqrd) >> (NB-1));
```

```
49      m4= ((sw[i+1]*ab)      >> (NB-1));
50      m5= ((sx[i]*b[i])      >> (NB-1));
51      fb -= -( -m2 + m3 );
52      ff +=  (  m5 - m4 );
53    }
54
55    /// compute intermediate value (w) and output sample
56    w = x + fb;
57    y = (w*b0) >> (NB-1);
58
59    /* check delle op. intermedie */
60    fprintf(fd,"For the %d samples: s1:%d s2:%d s4:%d m1:%d m2:%d m3:%d m4:%d
       m5:%d\n",cont,w-m3,w,ff,y,m2,m3,m4,m5);
61    fclose(fd);
62    cont+=1;
63
64    // compute the filtered result
65    y += ff;
66
67
68    /// update the shift register
69    for (i=N-1; i>0; i--)
70      sx[i] = sx[i-1];
71    sx[0] = x;
72    for (i=N; i>0; i--)
73      sw[i] = sw[i-1];
74    sw[0]=w;
75
76    return y;
77  }
78
79  int main (int argc, char **argv)
80  {
81    FILE *fp_in;
82    FILE *fp_out;
83
84    int x;
85    int y;
86
87    /// check the command line
88    if (argc != 3)
89    {
90      printf("Use: %s <input_file> <output_file>\n", argv[0]);
91      exit(1);
92    }
93
94    /// open files
95    fp_in = fopen(argv[1], "r");
96    if (fp_in == NULL)
97    {
98      printf("Error: cannot open %s\n",argv[1]);
99      exit(2);
```

```
100   }
101   fp_out = fopen(argv[2], "w");
102 
103   /// get samples and apply filter
104   fscanf(fp_in, "%d", &x);
105   do{
106     y = myfilter(x);
107     fprintf(fp_out,"%d\n", y);
108     fscanf(fp_in, "%d", &x);
109   } while (!feof(fp_in));
110   fclose(fp_in);
111   fclose(fp_out);
112 
113   return 0;
114 
115 }
```