Politecnico di Torino

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

INTEGRATED SYSTEMS ARCHITECTURE

Laboratory 1

Bernunzo Angela 198721
Busignani Fabio 197883
Gianoglio Emanuele 200090

February 2, 2014

Contents

Introduction

The aim of this laboratory is to design a *FIR* filter which respects some constraints shown below.
This project is divided into three different steps:

1. Simulation and verification of our system (Sec.2).
2. Synthesis of the filter (Sec.3).
3. Place and route of the FIR (Sec.4).

Before starting with the previous steps, we had to design by hand the FIR (Sec:1). The design specifications provided to us are the following ones:

- number of taps for the filter (n_t) equal to 7;
- number of bits to represent the input/output data (n_b) equal to 8;
- number of pipeline levels n_p equal to 2.

In order to achieve this project we used five different softwares:

- **Modelsim** to compile and simulate VHDL code sources of the filter;
- **Matlab** and **Code::Blocks** to verify results obtained by simulation;
- **Synopsys Design Compiler** for synthesising of the FIR, to find the area of the design, and to compute the first valuation of power consumption;
- **Cadence SOC Encounter** to achieve the place and route the FIR and to compute the second evaluation of power consumption.

In the following sections the steps that we have followed are shown.

1 Design by Hand

The first step to realize the filter is given by the design of it by hand.

1.1 A brief introduction to FIR

FIR (Finite Impulse Response) is a set of digital filters, an important class of LTI (Linear Time-Invariant) systems designed to modify the frequency properties of a signal. Differently from the continuous-time filters, realized by resistors and reactive components, digital filters are composed of adders, multipliers, and delay elements (registers).
There are two kinds of digital filter: FIR and IIR. The difference between them is given by presence of feedback in seconds. So, while output signal in IIR filter is given by the following equation:

$$y_i = \sum_{j=0}^{n_t-1} x_{i-j} \cdot H_j - \sum_{j=1}^{m_t-1} y_{i-j} * K_j; \tag{1}$$

in case of FIR filter the equation become:

$$y_i = \sum_{j=0}^{n_t-1} x_{i-j} \cdot H_j. \tag{2}$$

So, for a FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. In (Eq:2), H_j are the *tap weights* which define the filter's behavior.

1.2 Design

In this project the number of tabs is equal to 7, thus the FIR can be represented by the block diagram in (Fig.1) where each input signals and the output are expressed by 8 bits, while the internal parallelism is such as to guarantee no information loss inside.

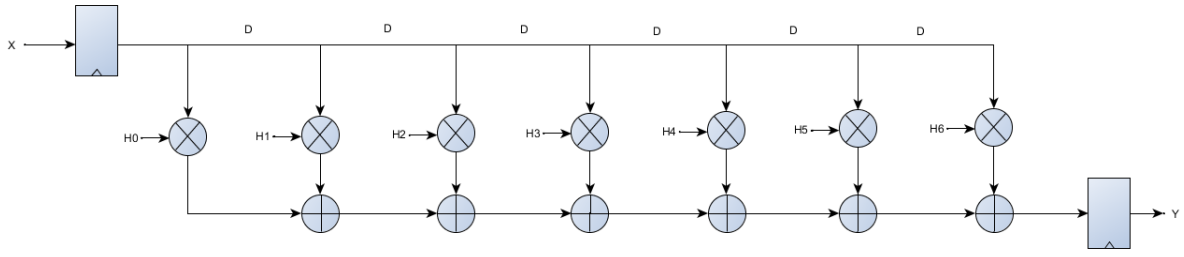

Figure 1: Block diagram of a 7-taps FIR

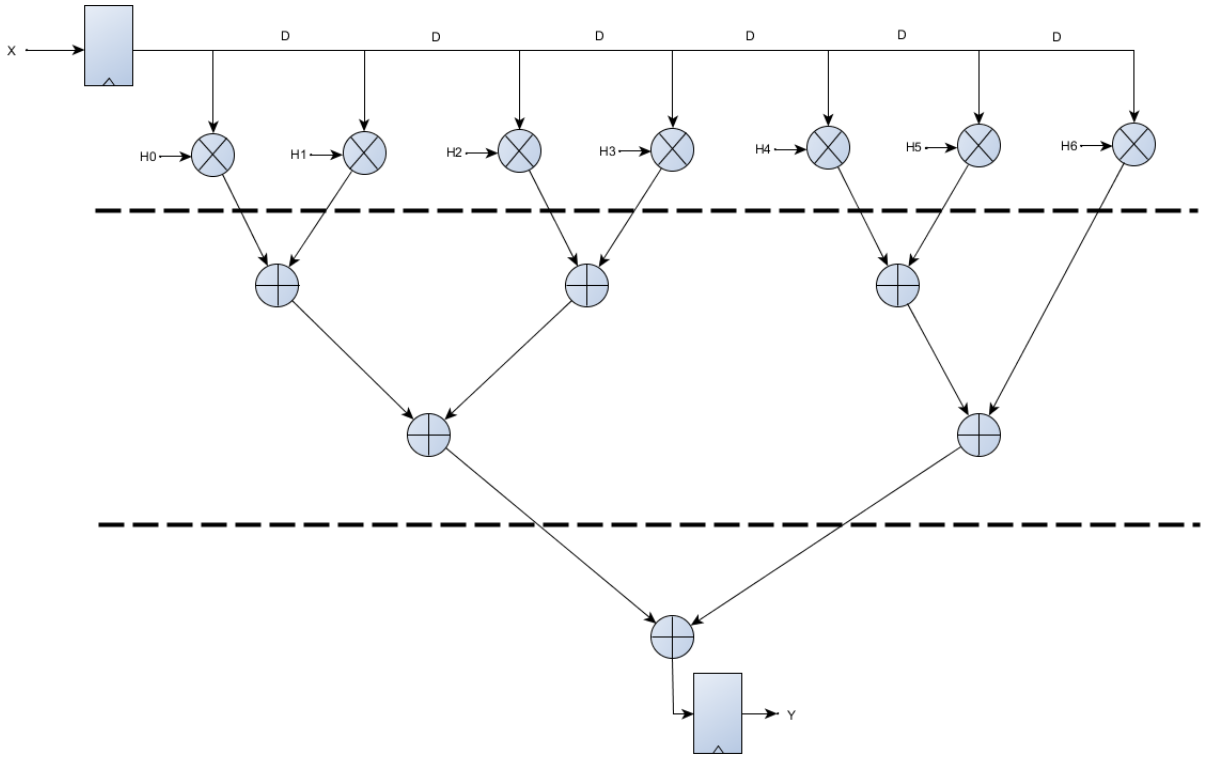

Figure 2: Pipelined FIR

1.2.1 Pipelining

Pipelining leads to a reduction in the critical path. In fact considering the first version of filter (Fig.1) the critical path is limited by one multiply and six adders.

$$T_{cp_1} = T_M + 6 \cdot T_A \tag{3}$$

Where:

- $\mathbf{T_M}$ is the time taken for multiplication;
- $\mathbf{T_A}$ is the time taken for addition.

In order to achieve a good pipelined architecture the previous block diagram is changed to realize a balancing path (Fig:2).
Considering the balacing path structure, without any pipelining latches, the critical path decrease becomes:

$$T_{cp_1} = T_M + 3 \cdot T_A \tag{4}$$

Now, starting by this architecture, pipelining registers can be introduced along the datapath for reducing the critical path.
Assuming that multiplication takes twice the time needed by the addition ($T_M \simeq 2 \cdot T_A$), a first pipelining level can be inserted between multipliers and adders. So, the critical path became:

$$T_{cp_2} = 3 \cdot T_A \tag{5}$$

The second, and last, pipelining level can be inserted in two different *feed-forward cutset* obtaining the same critical path reduction: between first and second adders level and between second and third adder level. In order to limit the power consumption and the required area, the best choice is shown in (Fig:2), where the dashed line indicates pipelining registers. In fact, in this case the pipelining registers are two instead of four.
Finally, the time required by critical path is:

$$T_{cp3} = 2 \cdot T_A \simeq T_M \tag{6}$$

1.2.2 Handshake

To communicate with the remaining part of the circuit two different signals are exploited:

1. *VIN*, Validation Input, setted when a new input sample is available, neglected otherwise;

2. *VOUT*, Validation Output, setted when output signal is available, neglected otherwise.

VOUT is kept low until the input shift register is completely full and the operations are done. For realizing this behaviour when *VIN* is asserted, it passes through input shift register and pipelinig level, whereupon it is provided outside as *VOUT* signal. A sort of bypassing is implemented to keep the coherence between *VOUT* and *DOUT* when *VIN* is at low logic level.
A complete view of the filter is shown in (Fig:3).

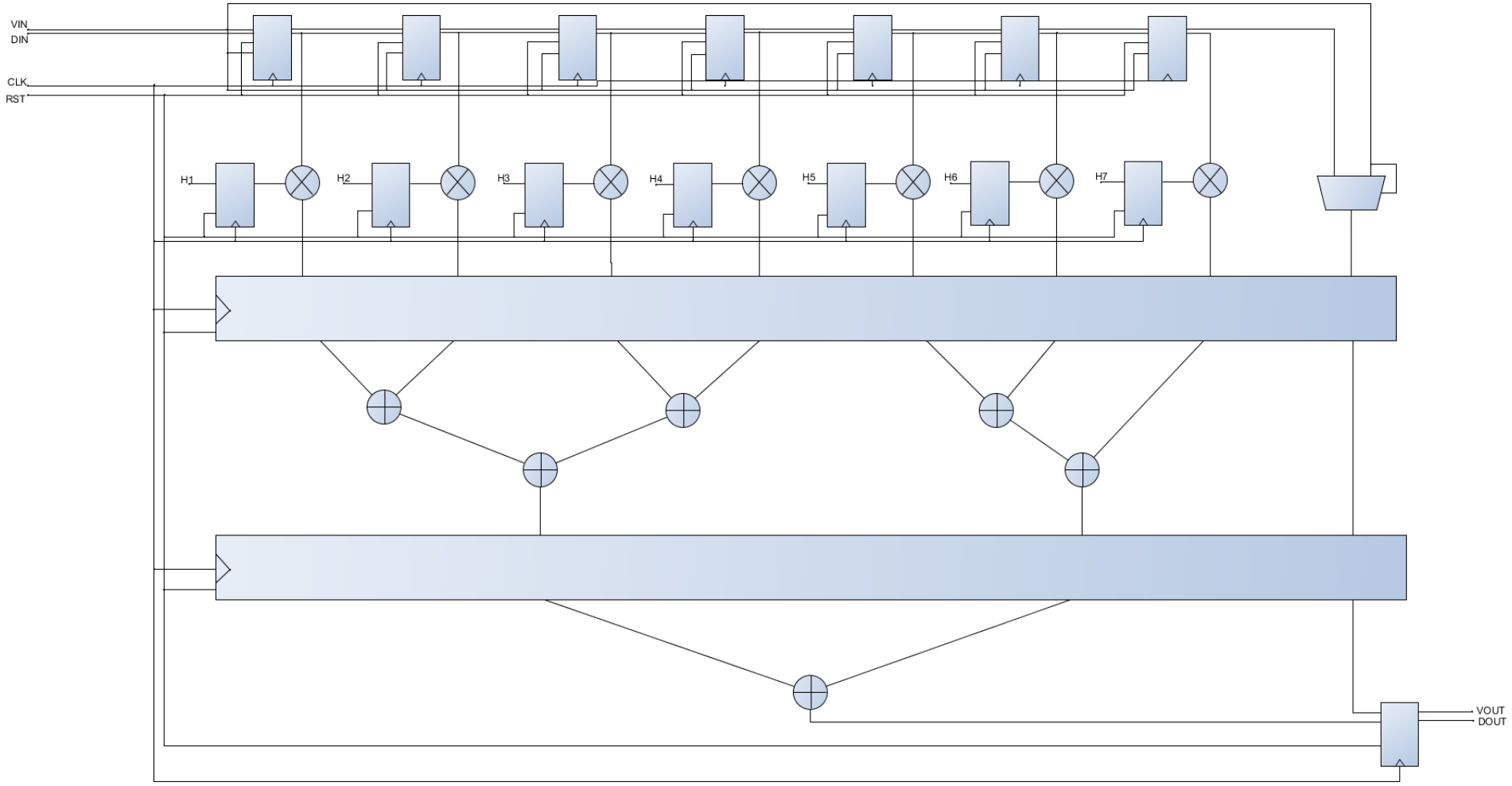

Figure 3: RTL view of the architecture

2 Simulation

After the design by hand, we have to use Modelsim in order to verify our system and to simulate it.
Starting with the simulation of our system we used the provided verilog test bench that includes clock generator, data generator and data sink. We linked our system following the provided instructions and then we performed the 100 samples simulation.

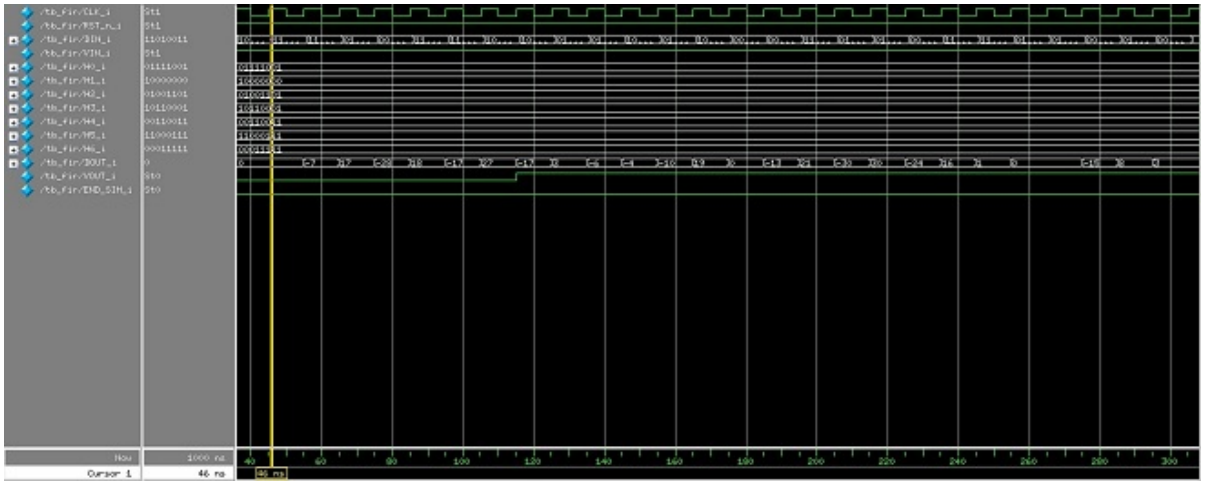

Figure 4: Simulation

Through the waveform viewer we have checked that our system works correctly according to the expected timing evolution and then we started to analyze the numerical results.
The output of *data_sink* block is a text file containing the results obtained through the Modelsim simulation in 2's complement 8 bit integer format converted into decimal base e.g.: 11101111 = -17.
In order to verify the correctness of this results, we have implemented by Matlab the SW model of our system.
First of all we have written a program (TB.m) that reads the inputs from the text file generated by *data_generator* block and computes the results considering all the input data like 2's complement 8 bit integer numbers converted into decimal base e.g.: 10000000 = -128. At the end of this computation the results are truncated in order to keep the parallelism of 8 bit, like in the HDL implementation, and are written onto a text file into decimal base format according to the output format of Modelsim simulation.
In this way we can compare the two output text files: the one provided by Modelsim and the other provided by Matlab.
This comparison is made by C language program that opens the two files and compares their contents. The output of the C program is that the comparison is successfull and thus we are sure that our HW system works correctly according to the SW model.
Moreover we want to be sure that the achieved results are not very different from

the real results that we would had if the input data were handled like fixed point numbers and with none truncation during the computation.
In order to do that, first of all we have transformed the integer results obtained by Modelsim and Matlab model into 8 bit fixed point format (Q4.4) numbers and then into decimal base through the function *int2fix4* (see Matlab code) e.g.: 11101111 = 1110.1111 = -1.0625.
Then we have written another Matlab program (FIR.m) that implements our system handling the input data like fixed point Q(1.7) numbers through the function *int2fix* (see Matlab code) e.g.: 1.0000000 = -1.
The input data, transformed into decimal base numbers, are processed by SW with none truncation mechanism.
These results are compared with the previous results, where truncation were been performed.
Comparing the first ten results in order to have an idea of the consequences of the truncation, we can state that the obtained result is acceptable, in fact there isn't a relevant difference between the numbers (see the table).

Result with truncation	Results without truncation
-1.0625	-1.0049
0.1875	0.2084
-0.3750	-0.3593
-0.2500	-0.2321
-0.6250	-0.6005
1.1875	1.2380
0.0000	0.0135
-0.8125	-0.7745
1.3125	1.3155
-1.8750	-1.8184

Table 1: Results comparison

3 Synthesis

After that simulation is done, the next step is given by the **synthesis** of the filter, in order to perform it the *Synopsys Design Compiler* software is exploited.
First of all we have prepared a file named **.synopsys_dc.setup** which contains the names and the file path of the technology libraries used during the synthesis.
The content of this file is the following:

```
define_design_lib WORK -path ./work
set search_path [list . /software/synopsys/syn_current
/libraries/syn/software/s$
set link_library [list "*" "fast.db" "fast.db"
"dw_foundation.sldb" ]
set target_library [list "fast.db" "fast.db" ]
set symbol_library [list "tsmc090.sdb" ]
set synthetic_library [list "dw_foundation.sldb"]
```

After, we have continued this step importing the synthesizable source files, which describe our architecture in VHDL language, and setting the constraints, like clock frequency, uncertainty of clock signal (due to be affected by jitter), delay of input and output ports, load capacitance of each output port. To ease all of this we realized a source file (*.src*) contains the follows code (which also starts the synthesis):

```
analyze -f vhdl -lib WORK ../src/adder.vhd
analyze -f vhdl -lib WORK ../src/mult.vhd
analyze -f vhdl -lib WORK ../src/mux.vhd
analyze -f vhdl -lib WORK ../src/package_vett_reg.vhd
analyze -f vhdl -lib WORK ../src/reg_pipe.vhd
analyze -f vhdl -lib WORK ../src/ShiftIN_bit.vhd
analyze -f vhdl -lib WORK ../src/ShiftIN.vhd
analyze -f vhdl -lib WORK ../src/SHIFT.vhd
analyze -f vhdl -lib WORK ../src/FIR.vhd
set_ultra_optimization true
set power_preserve_rtl_hier_names true
elaborate FIR -lib WORK > ./elaborate.txt
uniquify
link
create_clock -name MYCLK -period 10.0 CLK
set_dont_touch_network MYCLK
set_clock_uncertainty 0.07 [get_clocks MYCLK]
set_input_delay 0.5 -max -clock MYCLK [remove_from_collection
 [all_inputs] CLK]
set OLOAD [load_of fast/BUFX4/A]
set_load $OLOAD [all_outputs]
set_wire_load_model -name tsmc090_wl40
compile_ultra
```

3.1 Results

At this point we exploited the synthesis to view and analyze its results.

3.1.1 Timing

From *timing report* two important parameters are highlighted and can be show in the last line of the document:

```
  -----------------------------------------------------------
  data required time                                    9.87
  data arrival time                                    -4.47
  -----------------------------------------------------------
  slack (MET)                                           5.41
```

First parameter is given by slack. Since its value is positive (5.41) our filter met all the applied constraints, thus it can be well work at 100 MHz. Second parameter is the critical path delay, its value give us the maximum clock frequency achievable from our design. This upper bound frequency is:

$$f_{MAX} = \frac{1}{T_{CP}} = \frac{1}{4.47\ ns} \simeq 224\ MHz \tag{7}$$

Thus, the maximum working frequency is more than double that of the reference frequency.

3.1.2 Area consumption

Watching *area report* we can see the dimension of our device (which are expressed in μm^2).

```
Number of ports:              76
Number of nets:             1869
Number of cells:            1414
Number of references:         50

Combinational area:       9985.651109
Noncombinational area:    4696.473700
Net Interconnect area:    818350.937500

Total cell area:          14682.125000
Total area:               833033.062500
```

3.2 Switching-activity-based power consumption estimation

Using jointly *Modelsim* and *Synopsys Design Compiler* we performed a first power consumption estimation. With the first software we computed switching activity, which is given as an input of the second one in order to obtain the follows power report:

```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =   1.2230 mW   (81%)
  Net Switching Power  = 282.6312 uW   (19%)
                         ---------
Total Dynamic Power    =   1.5057 mW  (100%)

Cell Leakage Power     =  47.5540 uW
```

From this report we can see how the greater contribution in power consumption is given by short-circuit currents (*Cell Internal Power*).

4 Place and Route

In the last step a new directory named *soce* is required and some changes to the netlist file are applied, in order to perform the design flow of the project. The tools used are again **Encounter** and **Modelsim**. The procedure followed is:

- Import the design;
- Floorplanning;
- Power planning and routing;
- Cell placing;
- Signal routing;
- Timing and design analysis.

4.1 Import the design

After we set the top of the hierarchy and the target technology libraries, for the timing and the geometry, the file *design.conf* has been imported and loaded.

4.2 Floorplanning and Power planning and routing

By following the instructions we were able to set the area and the rings, for the power supply (V_{DD} and V_{SS}), around it. After that, we could see the height of the cells and the channels used for ring routing, set at 4 μm by the boundaries for all the four edges. Two different values of the metal layers height had been chosen, so that the power and ground signal could reach the whole chip without any congestion. In (Fig. 5) there is a list of each metal layer and the values for horizontal lines and vertical ones are specified:

```
#Auto G-grid size is set to 15
#Using automatically generated gcell grid.
#  Layer      Direction  #Track     Blocked     #Gcell    Blocked
#  ---------------------------------------------------------------
#  Metal 1        H         533           0        1332     85.06%
#  Metal 2        V         560           0        1332      0.00%
#  Metal 3        H         533           0        1332      0.00%
#  Metal 4        V         560           0        1332      0.00%
#  Metal 5        H         533           0        1332      0.00%
#  Metal 6        V         560           0        1332      0.00%
#  Metal 7        H         356           0        1332      0.00%
#  Metal 8        V         187           0        1332      0.00%
#  Metal 9        H         170           8        1332     10.44%
#  ---------------------------------------------------------------
#  Total                   3992       0.50%      11988     10.61%
```

Figure 5: Metal layers details list

Width and spacing values are set as recommended. Two new sets of stripes are routed and clicking on them all the information are displayed.
By using the command *SRoute* wires are placed for the standard cells and for connecting rings to the vertical stripes.

4.3 Cell placing

At this point everything is precisely known: the area of the single cell and where it is. In (Fig. 6) we show the result of the place step.

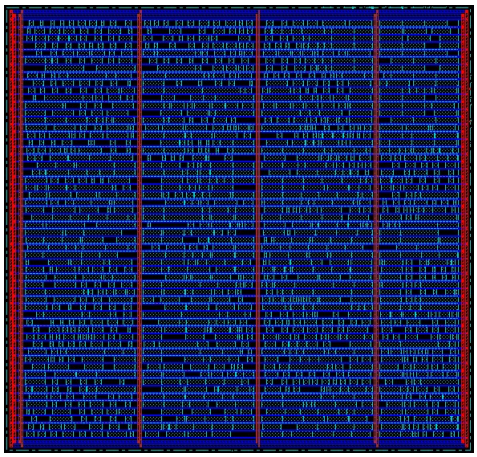

Figure 6: Result of the place step

An abstract view of cells and output and input pins is given now. By clicking on *Design Browser*, it is possible to explore the placed design.

4.4 Timing and design analysis

Concerning the clock, it is performed a three levels solution with three different buffers which have been specified in the verilog file. Then *clkconf.cts* is needed for its synthesis. Moreover it is important to fill every place gap, this is possible by adding five kinds of fillers among which it is allowed to choose. In (Fig. 7) a detailed report of the time analysis in the preroute phase is provided:

```
############################################################
# Complete Clock Tree Timing Report
#
# CLOCK: CLK
#
# Mode: preRoute
############################################################


Nr. of Subtrees                : 0
Nr. of Sinks                   : 278
Nr. of Buffer                  : 25
Nr. of Level (including gates) : 3
Max trig. edge delay at sink(R): REG_IN_H_i_5_q_reg_1_/CK 212.3(ps)
Min trig. edge delay at sink(R): SR_SRI_regi_1_q_reg_2_/CK 201.4(ps)

                                 (Actual)              (Required)
Rise Phase Delay               : 201.4~212.3(ps)       0~10000(ps)
Fall Phase Delay               : 227.1~240.8(ps)       0~10000(ps)
Trig. Edge Skew                : 10.9(ps)              10000(ps)
Rise Skew                      : 10.9(ps)
Fall Skew                      : 13.7(ps)
Max. Rise Buffer Tran          : 116.2(ps)             10000(ps)
Max. Fall Buffer Tran          : 111.8(ps)             10000(ps)
Max. Rise Sink Tran            : 45.1(ps)              10000(ps)
Max. Fall Sink Tran            : 41.9(ps)              10000(ps)


***** NO Transition Time Violation *****
```

Figure 7: Preroute clock timing report

4.5 Signal routing

The last step is provide by the two instructions *TrialRoute* and *NanoRoute*. The first one is the arrangement of wire position; the second one is a checking phase useful to avoid violations to the design rules. In (Fig. 8) is shown the result of all these steps with a list of colours, which indicate the everything has been added on the chip.

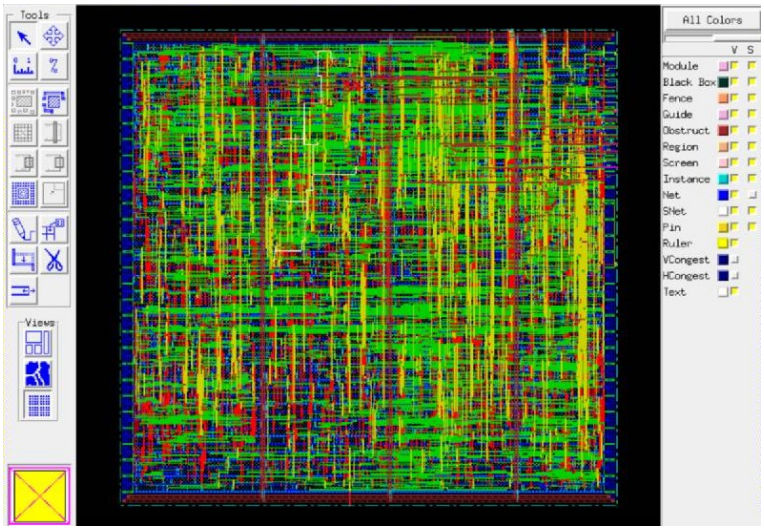

Figure 8: Result of place and route phases

After that, it was important to define the operating conditions for timing analysis: the recommended choice is on *fast* conditions.
Another capability of the tool is to extract the value of the resistance and capacitance of every rectangle.(Fig. 9)

```
#################################################
# The Instance Average Power Report for VDD net  #
#################################################
# Power calculation mode: vcd simulation
# Operating bias voltage: 1.1 Volt
# Units for power: Watts
# Format: Instance Total-Power Internal-Power Switching-Power Leakage-Power
#################################################################################

#################################################################################
# Sum:  1.810780e-03    1.195496e-03    5.676134e-04    4.767085e-05
# No. of instances: 1439
#################################################################################
```

Figure 9: Instance power report

Before the verification phase, thanks to the timing analysis it is possible to underline the constraints validity by observing the slack time sign and to highlight the critical path. In the file *FIR.slk* are listed the slack time values and we can notice that there are no negative slacks, so our filter can work at the specified frequency. We report in the picture below (Fig. 10) the message appeared in the shell where we were working, after the check of any possible violations.

```
**Info: no slack violation path.

Slack File   : FIR.slk
targetSlack : 0.000 ns
--------------------  -------    -------
 Slack Range (ns)       Count        Sum
--------------------  -------    -------
(   6.077 ~   6.000]         1          1
--------------------  -------    -------
Total negative slacks(TNS)=0
Worst negative slacks(WNS)=6.077
encounter 1> encounter 1> Selected path endpoint PIPE2_2_q_reg_3_/RN
*** Reported critical paths from clock "MYCLK" to clock "MYCLK" (0:00:00.0) ***
```

Figure 10: Check on slack time value

Thanks to the *Timing Slack Browser* we can analyze the clock path, as shown in the picture below (in Fig. 11).

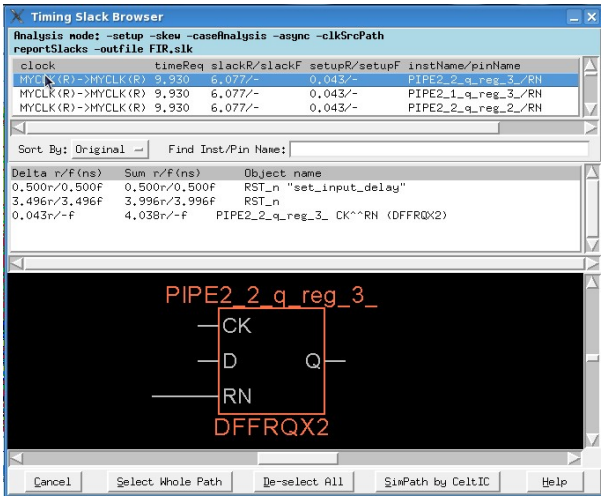

Figure 11: Clock analysis

By a double click on the first row in the file *FIR.slk* we obtain the critical path, reported below in (Fig. 12):

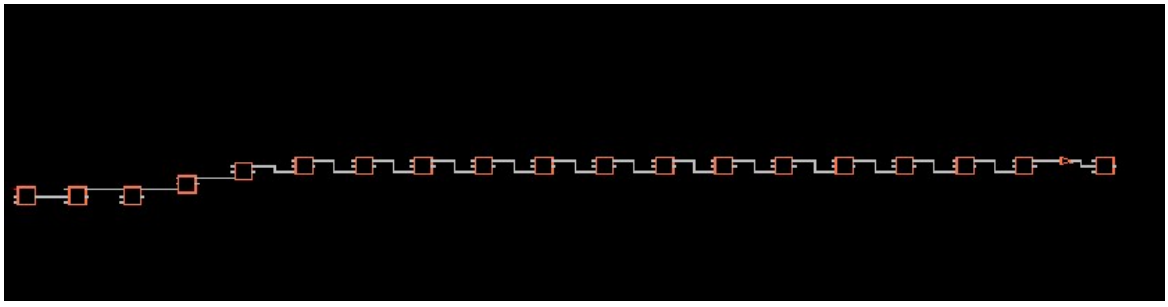

Figure 12: Critical path

Now, we demonstrate that the verification was successful by showing these images (Fig. 13) and (Fig. 14):

```
******** Start: VERIFY CONNECTIVITY ********
Start Time: Fri Dec 13 14:28:55 2013

Design Name: FIR
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (156.6850, 149.3200)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Fri Dec 13 14:28:56 2013
******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.2  MEM: 0.000M)
```

Figure 13: Connectivity verification

```
encounter 1>  *** Starting Verify Geometry (MEM: 109.2) ***

  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
  VERIFY GEOMETRY ..... SubArea : 1 of 1
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
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**********
```

Figure 14: Geometry verification

In the next figure (Fig. 15) is contained the report of a single instance.

```
Summary report for Instance: PIPE1_i_0_q_reg_2_
===============================================
              Cell Name: DFFRQX2
       No. of Terminals: 4
      Instance Location: 69.16, 49.56 um
        Instance Length: 0.011 um
         Instance Width: 6.16 um
        Instance Height: 2.52 um
   Instance Orientation: MX
```

Figure 15: Instance report

In the last picture is recorded the gate count report, obtained with a tool command named in the same way (Fig. 16):

```
Gate area 2.1168 um^2
Level 0 Module FIR
Gates=  7003
Cells=  1439
Area=   14825.4 um^2
```

Figure 16: Gate count report

Finally, in the post place and route phase, we took a new simulation with the goal of estimate the switching activity and so the power consumption onto the final result, achieved after the previous steps. Thanks to **Modelsim** simulation we could record the switching activity in a file *design.vcd*, but just after having simplified the verilog test-bench. Using again **Encounter**, by restoring the design previously saved, we could do another extraction of RC and evaluate the power consumption at the end(Fig. 17).

```
##########################################
# The Power Analysis Report for VDD net      #
##########################################
power supply: 1.1 volt
average power between 0.0000e+00 S and 1.0951e-06 S
Total id in vcd file: 1894
    In module tb_fir/UUT valid id: 1894
         redundant id: 0
    In module tb_fir/UUT invalid id: 0
        redundant id: 0
Total activity in vcd file: 110240
    In module tb_fir/UUT valid activity: 110240
    In module tb_fir/UUT invalid activity: 0
average power(default): 1.8108e+00 mw
    average switching power(default): 5.6761e-01 mw
    average internal power(default): 1.1955e+00 mw
    average leakage power(default): 4.7671e-02 mw
    user specified power(default): 0.0000e+00 mw
average power by cell category:
    core:  1.8108e+00 mw
    block: 0.0000e+00 mw
    io:    0.0000e+00 mw
biggest toggled net: RST_n
    no. of terminal: 279
    total cap: 1.1295e+03 ff
```

Figure 17: Power analysis

From a comparison led between pre- and post- Place and Route results in term of power, it was possible to notice that switching, total and internal power do not change their values, instead of the contribution of the leakage that goes form 0.0000e+00 mW to 4.767085e-05 W. The reason why we see this increase is that

A VHDL

A.1 Adder

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

entity adder is
  generic ( n : integer := 8);
  port(in_A, in_B: in signed(n-1 downto 0);
       sum: out signed(n-1 downto 0));
  end adder;

architecture Behavior of adder is

begin

  sum <= in_A + in_B;

end Behavior;
```

A.2 Multiplier

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

entity mult is
  generic ( n : integer := 8);
  port (data_in_1,data_in_2 : in signed (n-1 downto 0);
    dout : out signed (2*n-1 downto 0));
end mult;

architecture behavior of mult is

  begin
     dout <= data_in_1 * data_in_2;

end behavior;
```

A.3 Multiplexer

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

entity mux is
  port (in1: in std_logic;
    in2: in std_logic;
    sel : in std_logic;
    output: out std_logic);
end mux;

architecture behavior of mux is
begin
  with sel select
    output <= in1 when '0',
          in2 when others;
end behavior;
```

A.4 Register

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

entity reg_pipe is
generic (n : integer := 8);
port( Ck,rstn: in std_logic;
    d : in signed( n-1 downto 0);
    q: out signed(n-1 downto 0));
end reg_pipe;
architecture behavior of reg_pipe is
begin
    process (Ck)
    begin
      if (rstn = '0') then
        q<=(others =>'0');
      elsif (Ck'event and Ck ='1') then
          q <= d;
      end if;
    end process;
end behavior;
```

A.5 Shift Register

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

library work;
use work.package_vett_reg.all;

entity Shift is
port( rstn,en,Ck: in std_logic;
      Dato_in_bit: in std_logic;
      dato_out_bit: out std_logic;
      Dato_in: in signed (7 downto 0);
      dato_out: out vett_reg_vector(0 to 6));
end Shift;

architecture A of Shift is

component ShiftIN is
port( rstn,en,Ck: in std_logic;
      Dato_in: in signed (7 downto 0);
      dato_out: out vett_reg_vector(0 to 6));
end component ShiftIN;

component ShiftIN_bit is
port( rstn,en,Ck: in std_logic;
      Dato_in: in std_logic;
      dato_out: out std_logic);
end component ShiftIN_bit;

begin

SRI: ShiftIN port map(rstn => rstn,en => en,Ck => Ck,Dato_in => Dato_in ,
      dato_out => dato_out);
SRI_bit: ShiftIN_bit port map(rstn => rstn, en => en, Ck=> Ck,
      Dato_in => Dato_in_bit,dato_out=> dato_out_bit);

end A;
-------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.package_vett_reg.all;

entity ShiftIN is
port( rstn,en,Ck: in std_logic;
      Dato_in: in signed (7 downto 0);
```

```
49      dato_out: out vett_reg_vector(0 to 6));
    end ShiftIN;
51  architecture A of ShiftIN is
    component reg is
53  port( Ck,rstn, EN: in std_logic;
        d : in signed( 7 downto 0);
55      q: out signed(7 downto 0));
    end component;
57
    signal temp_out: vett_reg_vector(0 to 6);
59
      begin
61    reg0: reg port map(d => dato_in, ck => ck, rstn => rstn, en => en,
          q => temp_out(0));
63     u1: for i in 1 to 6 generate
        regi: reg port map (d => temp_out(i-1), CK => CK, Q => temp_out(i),
65          rstn => rstn, en => en);
      end generate;
67
      dato_out <= temp_out;
69  end A;

71  library ieee;
    use ieee.std_logic_1164.all;
73  use ieee.numeric_std.all;

75  entity reg is
    port( Ck,rstn, EN: in std_logic;
77      d : in signed( 7 downto 0);
        q: out signed(7 downto 0));
79  end reg;
    architecture behavior of reg is
81  begin
        process (Ck)
83      begin
          if (rstn = '0') then
85          q<=(others =>'0');
          elsif (Ck'event and Ck ='1') then
87          if (EN = '1') then
              q <= d;
89          end if;
          end if;
91      end process;
    end behavior;
93
    ----------------------------------------------------------------------

95
    library ieee;
97  use ieee.std_logic_1164.all;
```

```
99  entity ShiftIN_bit is
    port( rstn,en,Ck: in std_logic;
101     Dato_in: in std_logic;
        dato_out: out std_logic);
103 end ShiftIN_bit;
    architecture A of ShiftIN_bit is
105 component reg_bit is
    port( Ck,rstn, EN: in std_logic;
107     d : in std_logic;
        q: out std_logic);
109 end component;

111 signal temp_out: std_logic_vector(0 to 6);

113  begin
     reg0: reg_bit port map(d => dato_in, ck => ck, rstn => rstn, en => en,
115         q => temp_out(0));
        u1: for i in 0 to 5 generate
117     regi: reg_bit port map (d => temp_out(i), CK => CK, Q => temp_out(i+1),
            rstn => rstn, en => en);
119  end generate;
     dato_out <= temp_out(6);
121
    end A;
123
    library ieee;
125 use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
127
    entity reg_bit is
129 port( Ck,rstn, EN: in std_logic;
        d : in std_logic;
131     q: out std_logic);
    end reg_bit;
133 architecture behavior of reg_bit is
    begin
135     process (Ck)
        begin
137       if (rstn = '0') then
            q<='0';
139       elsif (Ck'event and Ck ='1') then
            if (EN = '1') then
141           q <= d;
            end if;
143       end if;
        end process;
145 end behavior;
```

A.6 FIR

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

library work;
use work.package_vett_reg.all;

entity FIR is
port( DIN : in signed(7 downto 0);
      H0, H1, H2, H3, H4, H5, H6: in signed(7 downto 0);
      VIN, RST_n, CLK : in std_logic;
      VOUT : out std_logic;
      DOUT : out signed(7 downto 0)
      );
end FIR;

architecture A of FIR is

component Shift is
port( rstn,en,Ck: in std_logic;
      Dato_in_bit: in std_logic;
      dato_out_bit: out std_logic;
      Dato_in: in signed (7 downto 0);
      dato_out: out vett_reg_vector(0 to 6));
end component Shift;

component reg_pipe is
generic (n : integer := 8);
port( Ck,rstn: in std_logic;
      d : in signed( n-1 downto 0);
      q: out signed(n-1 downto 0));
end component reg_pipe;

component mux is
  port (in1: in std_logic;
      in2: in std_logic;
      sel : in std_logic;
      output: out std_logic);
end component mux;

component mult is
  generic ( n : integer := 8);
  port (data_in_1,data_in_2 : in signed (n-1 downto 0);
      dout : out signed (2*n-1 downto 0));
end component mult;

component adder is
  generic ( n : integer := 8);
```

```
49    port(in_A, in_B: in signed(n-1 downto 0);
          sum: out signed(n-1 downto 0));
51  end component adder;

53  component reg_bit is
    port( Ck,rstn, EN: in std_logic;
55        d : in std_logic;
          q: out std_logic);
57  end component;


59
    signal H : vett_reg_vector(0 to 6);
61  signal SR_to_MUX, MUX_to_PIPE1, PIPE1_to_PIPE2, PIPE2_to_REGOUT:std_logic;
    signal SR_to_MUL, TEMP_H : vett_reg_vector(0 to 6);
63  signal MUL_to_PIPE1 : vett_reg_vector_16(0 to 6);
    signal PIPE1_to_ADD : vett_reg_vector_16(0 to 6);
65  signal ADD_esteso : vett_reg_vector_19(0 to 14);


67
    begin
69
    H(0) <= H0;
71  H(1) <= H1;
    H(2) <= H2;
73  H(3) <= H3;
    H(4) <= H4;
75  H(5) <= H5;
    H(6) <= H6;
77
    SR: Shift port map (rstn => RST_n ,en => VIN ,Ck => CLK ,
79      Dato_in_bit => VIN ,dato_out_bit => SR_to_MUX ,Dato_in => DIN ,
        dato_out => SR_to_MUL );
81
    MX_VIN: mux port map (in1 => VIN, in2 => SR_to_MUX, sel => VIN,
83      output => MUX_to_PIPE1);

85  PIPE1_VIN: reg_bit port map ( Ck => CLK, rstn => RST_n, EN => '1',
        d => MUX_to_PIPE1, q => PIPE1_to_PIPE2);
87
    PIPE2_VIN: reg_bit port map ( Ck => CLK, rstn => RST_n, EN => '1',
89      d => PIPE1_to_PIPE2, q => PIPE2_to_REGOUT);

91  REG_OUT_VIN: reg_bit port map ( Ck => CLK, rstn => RST_n, EN => '1',
        d => PIPE2_to_REGOUT, q => VOUT);
93
    pippo: for i in 0 to 6 generate
95    REG_IN_H_i: reg_pipe generic map (8)
                  port map( Ck => CLK, rstn => RST_n, d => H(i), q => TEMP_H(i));
97  end generate;
```

```
99
101 u1: for i in 0 to 6 generate
      MULi: mult  generic map(8)
103           port map (data_in_1 => SR_to_MUL(i) , data_in_2 => TEMP_H(i),
                  dout=> MUL_to_PIPE1(i) );
105 end generate;

107  u2: for i in 0 to 6 generate
      PIPE1_i: reg_pipe  generic map(16)
109               port map( Ck => CLK, rstn => RST_n, d => MUL_to_PIPE1(i),
                      q => PIPE1_to_ADD(i));
111 end generate;


113
    ADD_esteso(0)<=PIPE1_to_ADD(0)(15) & PIPE1_to_ADD(0)(15) & PIPE1_to_ADD(0);
115 ADD_esteso(1)<=PIPE1_to_ADD(1)(15) & PIPE1_to_ADD(1)(15) & PIPE1_to_ADD(1);
    ADD_esteso(2)<=PIPE1_to_ADD(2)(15) & PIPE1_to_ADD(2)(15) & PIPE1_to_ADD(2);
117 ADD_esteso(3)<=PIPE1_to_ADD(3)(15) & PIPE1_to_ADD(3)(15) & PIPE1_to_ADD(3);
    ADD_esteso(4)<=PIPE1_to_ADD(4)(15) & PIPE1_to_ADD(4)(15) & PIPE1_to_ADD(4);
119 ADD_esteso(5)<=PIPE1_to_ADD(5)(15) & PIPE1_to_ADD(5)(15) & PIPE1_to_ADD(5);
    ADD_esteso(6)<=PIPE1_to_ADD(6)(15) & PIPE1_to_ADD(6)(15) & PIPE1_to_ADD(6);
121


123 u3: for i in 0 to 2 generate
      ADD1_i : adder generic map(18)
125             port map(in_A => ADD_esteso(i+i), in_B => ADD_esteso(i+i+1),
                    sum => ADD_esteso(i+7));
127 end generate;

129 ADD2_1: adder  generic map(18)
             port map(in_A => ADD_esteso(7), in_B => ADD_esteso(8),
131             sum => ADD_esteso(10));

133 ADD2_2: adder generic map(18)
           port map(in_A => ADD_esteso(9), in_B => ADD_esteso(6),
135             sum => ADD_esteso(11));

137 PIPE2_1: reg_pipe  generic map(18)
               port map( Ck => CLK, rstn => RST_n, d => ADD_esteso(10),
139                 q => ADD_esteso(12));

141 PIPE2_2: reg_pipe  generic map(18)
               port map( Ck => CLK, rstn => RST_n, d => ADD_esteso(11),
143                 q => ADD_esteso(13));

145 ADD3_1: adder generic map(18)
           port map(in_A => ADD_esteso(12), in_B => ADD_esteso(13),
147             sum => ADD_esteso(14));
```

```
149 REG_OUT: reg_pipe  generic map(8)
                port map( Ck => CLK, rstn => RST_n,
151                     d => ADD_esteso(14)(17 downto 10), q => DOUT);
end A;
```

B Verilog

B.1 Testbench

```
module tb_fir ();
2
    wire CLK_i;
4   wire RST_n_i;
    wire [7:0] DIN_i;
6   wire VIN_i;
    wire [7:0] H0_i;
8   wire [7:0] H1_i;
    wire [7:0] H2_i;
10  wire [7:0] H3_i;
    wire [7:0] H4_i;
12  wire [7:0] H5_i;
    wire [7:0] H6_i;
14  wire [7:0] H7_i;
    wire [7:0] DOUT_i;
16  wire VOUT_i;
    wire END_SIM_i;
18
    clk_gen CG(.END_SIM(END_SIM_i),
20          .CLK(CLK_i),
          .RST_n(RST_n_i));
22
    data_maker SM(.CLK(CLK_i),
24          .RST_n(RST_n_i),
      .VOUT(VIN_i),
26    .DOUT(DIN_i),
      .H0(H0_i),
28    .H1(H1_i),
      .H2(H2_i),
30    .H3(H3_i),
      .H4(H4_i),
32    .H5(H5_i),
      .H6(H6_i),
34    .END_SIM(END_SIM_i));

36  FIR UUT(.CLK(CLK_i),
        .RST_n(RST_n_i),
38      .DIN(DIN_i),
              .VIN(VIN_i),
```

```
        .H0(H0_i),
        .H1(H1_i),
        .H2(H2_i),
        .H3(H3_i),
        .H4(H4_i),
        .H5(H5_i),
        .H6(H6_i),
              .DOUT(DOUT_i),
              .VOUT(VOUT_i));

   data_sink DS(.CLK(CLK_i),
    .RST_n(RST_n_i),
    .VIN(VOUT_i),
    .DIN(DOUT_i));

endmodule

always @ (END_SIM_i) begin
if (END_SIM_i) begin
$toggle_stop;
$toggle_report("../saif/myfir_back.saif", 1.0e-9, "tb_fir.UUT");
end
end
```

C Matlab

C.1 Function: from complement to integer

```
function [ c2 ] = complement2int( a )
%Function take as input a string which represent a number in two's complement
%and provides outsite its value converted in integer
%                       [ c2 ] = complement2int( a )
  if a(1) == '1'
    c2 = -2^((length(a))-1) + bin2dec(a(2:length(a)));
  else
    c2 = bin2dec(a);
  end
end
```

C.2 FIR: Modelsim version

```
x=load('Dati_in.txt');
H = [ 121 ; 128 ; 77 ; 177 ; 51 ; 199 ; 31];
H_bin_1 = dec2bin(H(1),8);
H_bin_2 = dec2bin(H(2),8);
H_bin_3 = dec2bin(H(3),8);
H_bin_4 = dec2bin(H(4),8);
```

```
H_bin_5 = dec2bin(H(5),8);
H_bin_6 = dec2bin(H(6),8);
H_bin_7 = dec2bin(H(7),8);

% convert vector H to signed
H(1) = complement2int(H_bin_1);
H(2) = complement2int(H_bin_2);
H(3) = complement2int(H_bin_3);
H(4) = complement2int(H_bin_4);
H(5) = complement2int(H_bin_5);
H(6) = complement2int(H_bin_6);
H(7) = complement2int(H_bin_7);

c=1
indice = 1;
i= length(x);
dato_elaborato=zeros(7,1); in ingresso
f = zeros(7,1);
vettore_uscita=zeros(fix(length(x)-7));
while i >= 7
    i = i -1;
    dato_elaborato = x(indice:indice+6);
    indice = indice +1;
    dato_elaborato = flipud(dato_elaborato); %invert element order in column vector
        %because first sample has to be multiplied with last coefficient
    f = dato_elaborato .* H;

    y = sum(f);
    y_bin = int2complement(y,18);
    y = complement2int(y_bin(1:8)); %truncation
    fprintf('Il valore dell''uscita e''%d \n',y);

    vettore_uscita(c) = y;
    c = c+1;
end

file_id= fopen('uscita.txt','w') ;
for i=1:c-1
fprintf(file_id,'%d\n',vettore_uscita(i));
end
fclose(file_id);
```

C.3 Function: from integer to fixed point

```
function [ w ] = int2fix1( a )
%Function take as input an integer type number and provides outsite its value
%converted in fixedpoint with just a bit reservet for the integer part
%     function [ w ] = int2fix1( a )
  b=int2complement(a,8);
```

```
    c=zeros(8,1);
    h=[-2^0;2^-1;2^-2;2^-3;2^-4;2^-5;2^-6;2^-7];
    for i = 1:8
        c(i) = str2num(b(i));
    end
    w=sum(c.*h);

end

function [ w ] = int2fix4( a )
%Function take as input an integer type number and provides outsite its value
%converted in fixedpoint with four bit reservet for the integer part
%     function [ w ] = int2fix4( a )
b=int2complement(a,8);
c=zeros(8,1);
h=[-2^3;2^2;2^1;2^0;2^-1;2^-2;2^-3;2^-4];
for i = 1:8
    c(i) = str2num(b(i));
end
w=sum(c.*h);

end
```

C.4 FIR: fixedpoint version

```
N=7;

h=[121,-128,77,-79,51,-57,31];
for y = 0:15
    f1=fopen('Dati_in.txt');
    f=fscanf(f1,'%g',y);
    f=fscanf(f1,'%g',N);
    fclose(f1);


    data_in=zeros(1,7);

    for i=1:7
        data_in(i)=int2fix1(f(i));
    end
    %data_in

    h_in=zeros(1,7);

    for i=1:7
        h_in(i)=int2fix1(h(i));
    end
    %h_in
```

```
25      data_in_inv= fliplr(data_in);

27      result=sum(data_in_inv.*h_in)
end
```

D C

D.1 Matching evaluator

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define N 4
6
int main ()
8 {
    FILE *fptr, *fptr2, *fptr_o;
10  char s1[N],s2[N];
    int riga=1, errore = 0;
12
    fptr = fopen("results.txt","r");
14  fptr2 = fopen("uscita.txt","r");
    fptr_o = fopen("esito.txt","w");
16
    if(fptr == NULL) {
18      fprintf(fptr_o, "Errore nell'apertura del file");
        exit(1);
20      }
    if(fptr2 == NULL){
22      fprintf(fptr_o, "Errore nell'apertura del file");
        exit(1);
24      }
    if(fptr_o == NULL){
26      fprintf(fptr_o, "Errore nell'apertura del file");
        exit(1);
28      }

30  while(fscanf(fptr,"%s",s1) != EOF){
        fscanf(fptr2,"%s",s2);
32      if (strcmp (s1,s2) != 0){
            fprintf(fptr_o, "Errore nella riga %d\n", riga);
34          errore = 1;
        }
36      riga++;
    }
38
    if(errore != 1){
```

```
40              fprintf(fptr_o, "Tutto_corretto");
        }
42
44
        fclose(fptr);
46      fclose(fptr2);
        fclose(fptr_o);
48      return 0;
}
```