Politecnico di Torino

III Facoltà di Ingegneria

Exercises and Homeworks for the course Integrated Systems Architecture

Master degree in Electronic Engineering

Group 07

Campagnoli Rafael, Fraval Adrien, Galasso Luigi

November 15, 2019

Contents

CHAPTER 1

Lab 1: Design and Implementation of a Digital Filter

1.1 Introduction

1.1.1 Objectives

The purpose of this laboratory was to develop a digital filter with a cut-off frequency of 2 kHz and with a sampling frequency set to 10kHz using different implementation techniques and comparing the results. In the case of this laboratory activity, the filter to be designed was a Finite Response Filter (FIR) with an order of $N = 8$ and number of bits $n_b = 14$. Starting from a Matlab implementation of the filter a C-code fixed point implementation was developed. After that, a VHDL description at RTL level of the basic filter has been realized. The architecture has been tested, synthesized and finally physically implemented. As last step, optimization techniques such as unfolding and pipelining were applied. Improvements in terms of critical path delay were obtained after the repetition of the previous mentioned design phases.

1.1.2 FIR filter

A FIR filter direct form can be represented as it can be seen in Figure 1.1. One input x and one output y are the main ports and it is basically composed by N adders, N+1 multipliers and N registers. Furthermore, the equation 1.1 reports the time domain behavior of the FIR filter, where $x(n-k)$ are the input samples and b_k are the impulse response factor in time domain.

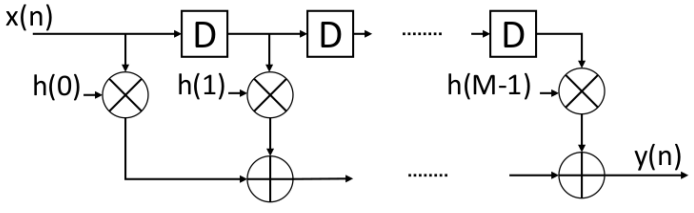

Figure 1.1: FIR filter

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) \tag{1.1}$$

1.2 Reference model development

1.2.1 Matlab

As a first step, two given Matlab scripts were executed for the generation of the samples and the output files. The main script, *"my_fir_filter.m"*, calls the function script *"myfir_design.m"* to generate a FIR filter with the assigned parameters. The matlab script generates the coefficients to be used in the calculations of the filter. Then, the main script generates two sinusoidal waves, one in and one out of the cut-off band, apply the filter for both waves and display the output. At the end, the script generates a "samples.txt" file, i.e. the file containing the input samples, and a "mresults.txt" file, i.e. the file containing the filtered values. The frequency response and time domain response of Matlab simulation are reported in Figure 1.2 and Figure 1.3.

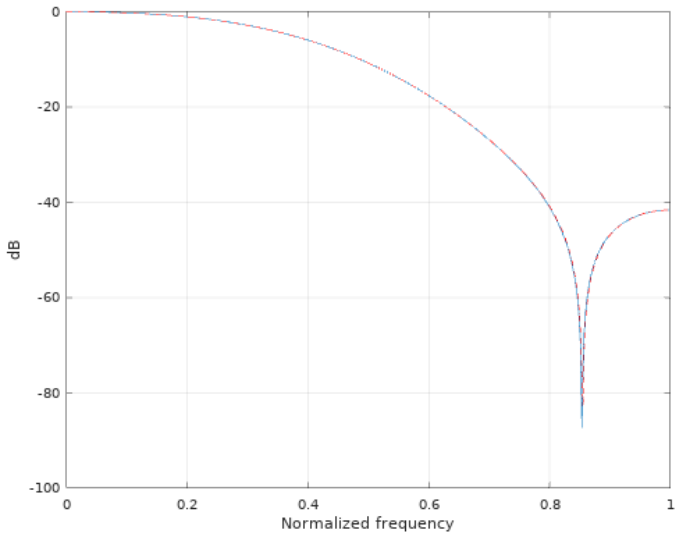

Figure 1.2: Frequency domain response

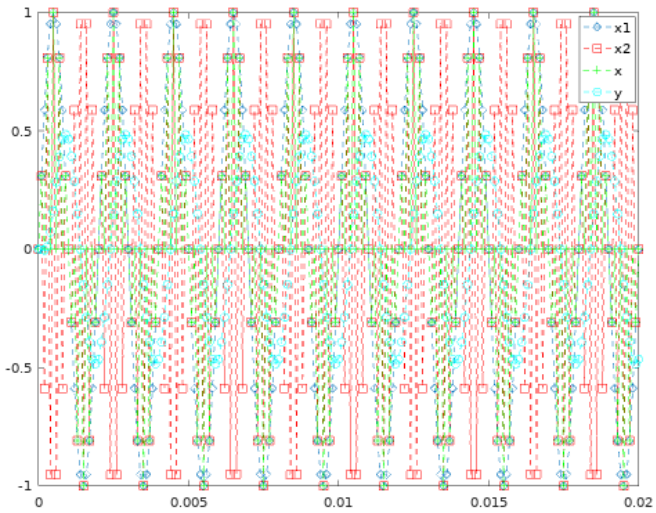

Figure 1.3: Time domain response

1.2.2 C Program

In the successive step, the FIR filter was implemented using a fixed point approach in C language. The "samples.txt" file and the coefficients generated by the Matlab script were used by the program in order to generate another output file, "outputc.txt" containing the filtered results. The output files was used for further comparison.

1.2.3 Comparison

In the Table 1.1 it has been reported a comparison between the first output results in the two implementation, i.e. Matlab and C. As it can be observed by the third column the two outputs differ in many cases, probably due to the fact that Matlab script does not exploit a fixed point approach with respect to the C program.

Results Comparison		
MATLAB	C	Difference
0	0	0
-15	-16	1
-34	-35	1
89	87	2
582	581	1
1316	1313	3
2321	2320	1
3197	3193	4
3811	3808	3
3970	3965	5
3810	3807	3
3208	3206	2
2351	2351	0
1225	1222	3
-1	-5	4
-1226	-1232	6

Table 1.1: Matlab and C results

1.3 VLSI implementation

Referring to the given schema reported in Figure 1.4 a VHDL description of the filter was implemented and simulated, with a further logical synthesis finishing with placing and routing. All steps generated different logical netlists and were simulated. Timing and area reports were also written. As it can be seen by the Figure, besides the n_b(n-bits) input (DIN) , output(DOUT), impulse response factors(b_j) an enable signal for input data(VIN) and output data(VOUT) have been added. It is here pointed out that the Filter's registers are positive edge triggered and reset low-level wise.

1.3.1 VHDL Model

The behavioral FIR filter was descibred in VHDL following the structure described in Subsection 1.1.2. The code can be observed by the Appendix A. It exploits two processes clock and reset-sensitive, i.e. one to obtain the correct delayed input and the other for the correct output synchronization. In

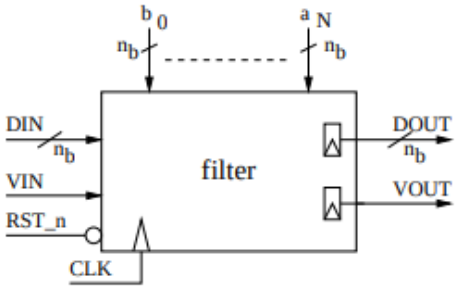

Figure 1.4: General Schema

addition it can be noticed that the operations of addition and multiplication are behavioral and asynchronous.

For the simulation, a given testbench was modified and used to read the samples.txt as an input of the filter, with a following output file generation. It could be observed that the VHDL described filter generates an output exactly equal to the fixed point C program, showing a correct behaviour of the VHDL filter.

1.3.2 Synthesis

For the circuit Synthesis, the **Synopsys** design compiler was used. The VHDL file of the filter was elaborated and compiled using a clock constraint of 0 ns, in order to verify which was the minimum clock period the design compiler could achieve. After that, the clock constraint was set to 4 times the minimum period found and the circuit was synthesised again. In both phases, the timing and area reports were generated. To finish the synthesis script, three files were written, a verilog file containing the netlist, a .sdf file used to be used for the switching activity analysis and a .sdc file to be used in the physical implementation phase.

The verilog netlist file was then simulated using the same testbench used before in order to check if the synthesised circuit works correctly. In the simulation script, it was also included the generation of a .vcd file containing the switching activity information.

To finalize the sythesis analysis, a switching-activity-based power consumption estimation was performed. To do that, the .vcd file generated in the simulation phase was converted to a .saif file using **Synopsys's** *vcd2saif* command and then the generated file was read by the design compiler. A power report could then be generated by the design compiler. The results of the final synthesised timing-optimized FIR filter can be observed in Table 1.2 and 1.3. The critical path delay obtained allow to achieve a maximum clock frequency of 400 MHz.

Critical Path Delay	Cell Area
2.49 ns	7620.102 μm^2

Table 1.2: Simple FIR filter reports

Internal Power	Switching Power	Leakage Power	Total Power
1.2541e+03 μW	1.0740e+03 uW μW	1.5332e+05 nW	2.4814e+03 μW

Table 1.3: Simple FIR filter Power reports

1.3.3 Place and Route

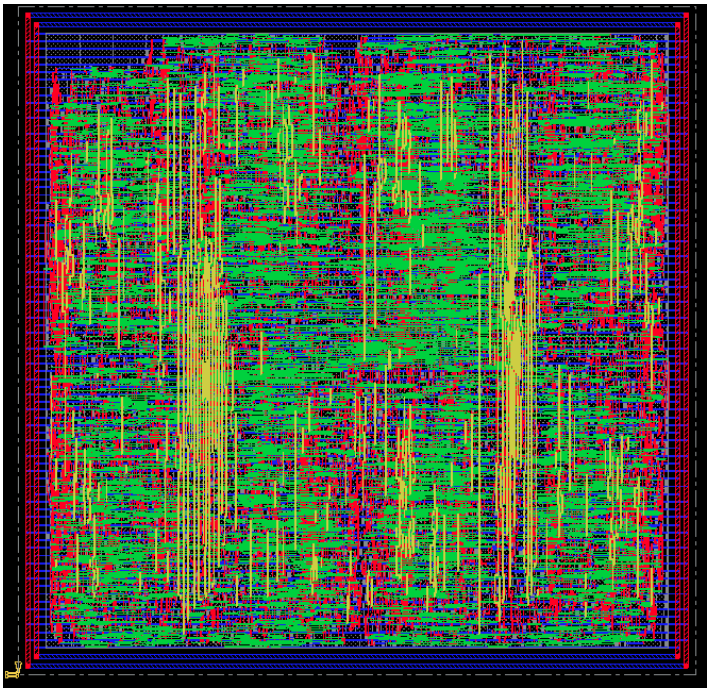

Figure 1.5: Screen Dump of the physical implementation of the basic FIR filter

For the physical design, a standard procedure was followed. The verilog file generated in the synthesis part, i.e. the one synthesized and tested at the maximum frequency divided by 4, was used for place and route using the software **Innovus**, through an optimization for routing. The results of the placing and routing can be observed in the Figure 1.5 showing the screen dump generated. The post Routing analysis results are reported in the following Table 1.4.

Gate Count	Area
9555	7625.4 μm^2

Table 1.4: Gate Count and Area obtained in the physical design.

As previously done after the synthesis, in this case too, a further simulation post Place and Route was performed in order to confirm the correct behaviour of the system. A .vcd file was also generated in order to estimate the power consumption on the routed system. The power analysis results can be observed in Table 1.5.

Internal Power	Switching Power	Total Power	Leakage Power
1.241 mW	0.997 mW	2.391 mW	0.1532 mW

Table 1.5: Power analysis of the basic FIR filter in the physical design.

1.4 Advanced architecture development

In this section two techniques were explored to perform some improvements in terms of critical path delay so that it could be achieved an higher clock frequency for the FIR filter. First of all the Unfolding was applied and then a further modification of the architecture through the Pipelining was done. For these two implementations the project flow phases were followed and some comparison were performed.

1.4.1 Unfolding

The Unfolding technique consists of a procedure which can be exploited to move from a pure sequential computation into a parallel computation. For this laboratory activity the L-level of Unfolding chosen was 3. A formal method was applied and the final architecture obtained is reported in Figure 1.6. It can be observed by the Figure that the critical path remains, exactly the same of the previous simple implementation, however the amount of resources used is higher. In particular the sequential blocks(registers) are the same number as the simple FIR filter and the combinational blocks(adders and multipliers) are three times more. Therefore the same performance in terms of timing have been obtained through the usage of more resources.

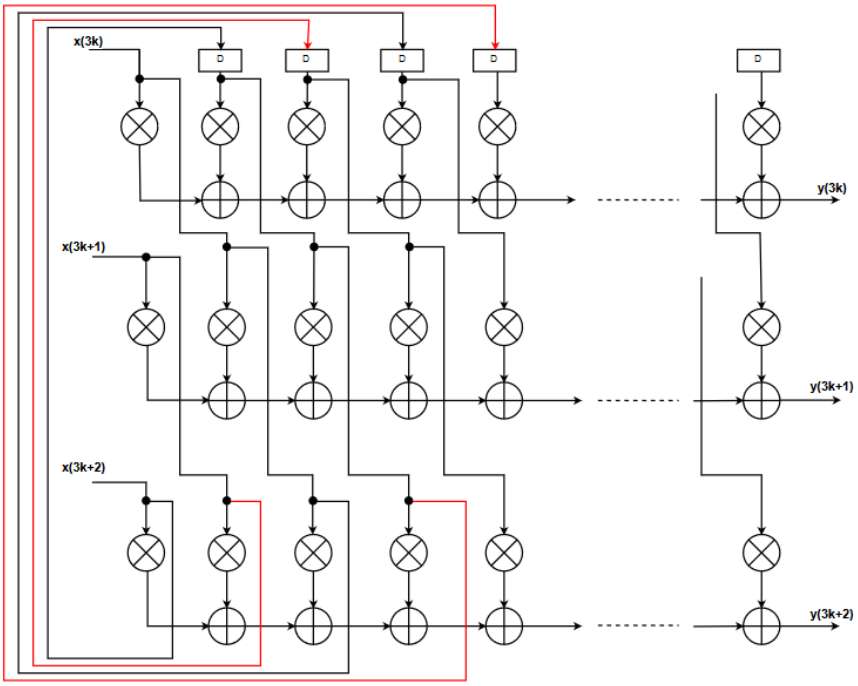

Figure 1.6: FIR filter L=3 Unfolding

1.4.2 Pipelining

The Pipelining technique consists of cutting the critical path(CP) of the architecture, which is the bottleneck of the timing performance. This can be achieved inserting pipeline registers, in order to reduce the delay increasing the latency. The critical path is the longest path from a register output port to an other register input port. In that case, it goes from the input x to the output y, which are supposed to be hosted by registers. In particular the CP is here composed by the sequence of 1 multiplier and the following N adders. In that case a feed-forward cut set is found as reported in

Figure 1.7 and along it in each edge cut, apart from the jumper wire through the levels, one register has been added. The architecture is reported in Figure 1.8.

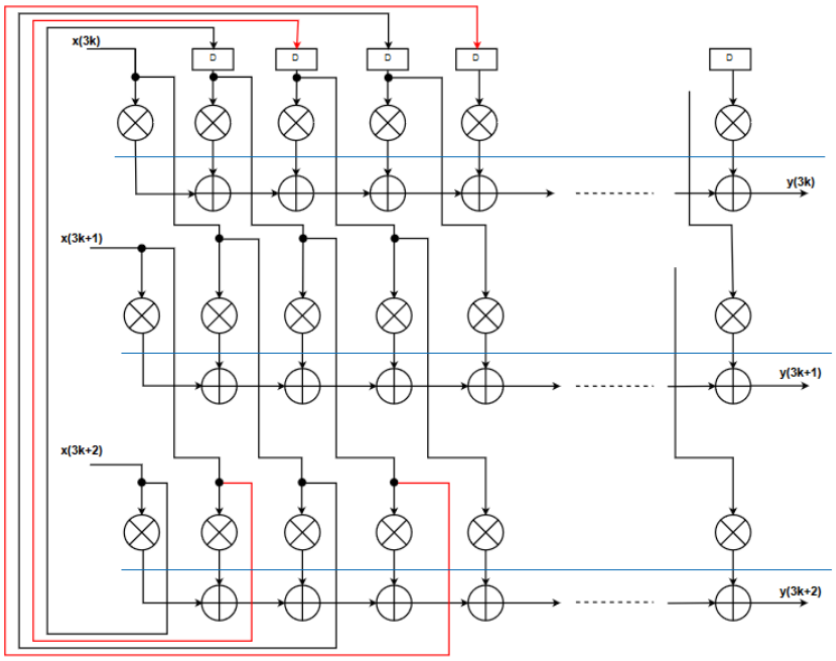

Figure 1.7: FIR filter Feed-Forward Cut Set highlighted

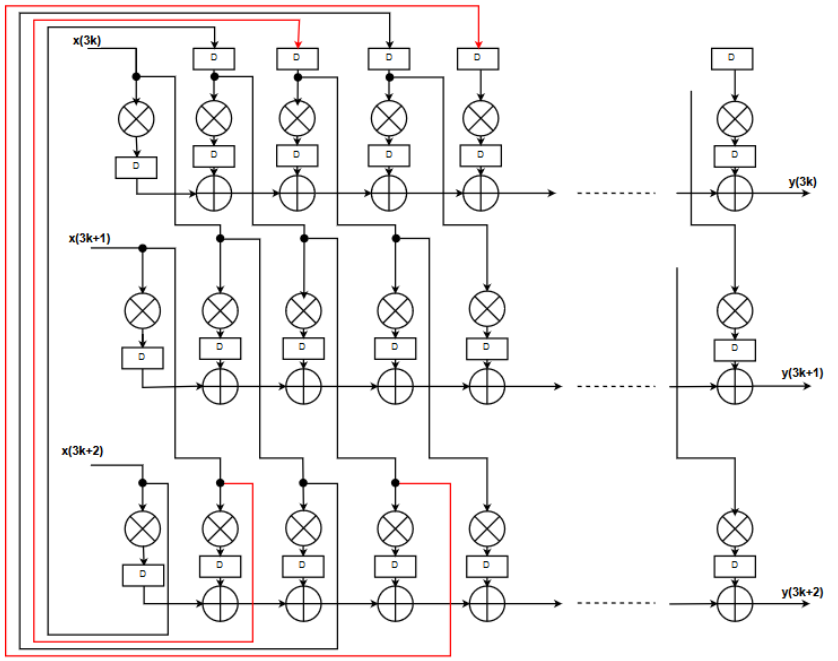

Figure 1.8: FIR filter L=3 Unfolded and Pipelined

1.4.3 VHDL Model

Using a behavioral approach the advanced architecture design was implemented following the functional schema previously proposed. The complete VHDL code is reported in the Appendix. In that case, a simple N bit register was exploited to simplify the code. The general structure is similar to the simple filter; more signals have been exploited in that case for the greater amount of components and connections. In that case too, a simulation was performed exploiting a modified TestBench with the respect to the simple implementation one.

1.4.4 Synthesis

The synthesis of the timing optimized FIR filter followed the same procedure of the previous case, described in Section 1.3.2. The same reports were generated and its results can be better visualized in the following Table 1.6 and 1.7 .

Critical Path Delay	Cell Area
1.64 ns	32433.37 μm^2

Table 1.6: Advanced time-optimized FIR filter reports.

Internal Power	Switching Power	Leakage Power	Total Power
2.3575e+03 μW	1.9400e+03 μW	7.2903e+05 nW	5.0265e+03 μW

Table 1.7: Advanced time-optimized FIR filter Power reports.

1.4.5 Place and Route

The Place and Routing procedure was followed just as before, and the same reports were generated and can be observed in the Tables 1.8 and 1.9. A screen Capture is also reported in Figure 1.9. In this case, too the correct behavior of the filter synthesized and physically designed at a reduced frequency(4 times smaller than the maximum achievable) was confirmed by the simulations.

Gate Count	Area
11450	26451.0 μm^2

Table 1.8: Gate Count and Area obtained in the physical design of the optimized FIR filter.

Internal Power	Switching Power	Total Power	Leakage Power
6.079 mW	3.903 mW	10.5 mW	0.5195 mW

Table 1.9: Power analysis of the optimized FIR filter in the physical design.

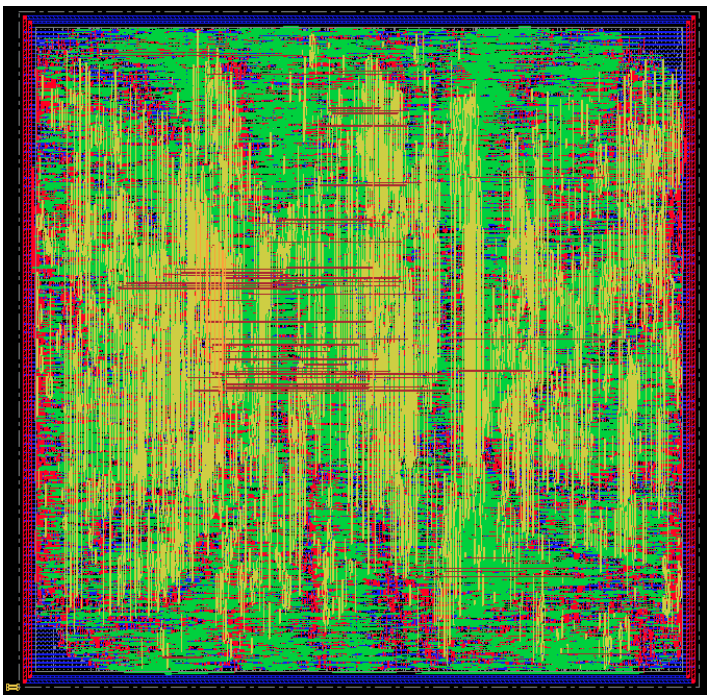

Figure 1.9: Screen Dump of the physical implementation of the advanced FIR filter

1.5 Results and Conclusion

The results obtained through the process of optimization have revealed that the simple L=3 Unfolding does not allow to get better timing performance, in fact as previously illustrated the critical path associated to the architecture was the same of the not unfolded one. In addition it's easy to understand that the combinational blocks required are more than the simple case, and so the area occupation would be higher. Whereas the pipelining technique, applied to the highlighted feed-forward cut-set slicing the critical path, improves the the timing performance at the cost of a greater area and an higher power consumption.

In particular, observing the post Synthesis report results it can be concluded that:

- **Critical Path Delay**: it's reduced in the optimized architecture from 2.49ns to 1.64ns. The maximum frequency achievable is therefore 590 MHz;
- **Area**: the area is bigger in the optimized architecture, in fact it grows from 7620.102 μm^2 to 32433.37 μm^2: the combinational blocks are tripled and the registers are quadrupled;
- **Total Power Consumption**: it's almost doubled from 2.4814mW to 5.0265mW.

For what concern the post Place and Route reports, remembering that it was performed in both the simple and advanced FIR filter exploiting the generated netlists at the reduced frequency, instead it can be here noted that:

- **Area**: the area is bigger in the optimized architecture, according with the higher resources usage, from 7625.4 μm^2 to 26451.0 μm^2;
- **Power Consumption**: all the power components strongly increase from the simple to the advanced architecture, in particular the Internal Power becomes five times bigger and the Switching Power 4 times.

APPENDIX A

VHDL Code

A.1 Constants package

```
package CONSTANTS is
   constant NB : integer := 14;
   constant NT : integer := 9;
end CONSTANTS;
```

A.2 FIR filter

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;
use WORK.constants.all;
use WORK.all;

entity myfir is
port (
        CLK:            IN std_logic;
        RST:            IN std_logic;
        VIN:            IN std_logic;

        DIN:            IN signed(NB - 1 downto 0);

        B0:         IN signed(NB - 1 downto 0);  --coefficients
        B1:         IN signed(NB - 1 downto 0);
        B2:         IN signed(NB - 1 downto 0);
        B3:         IN signed(NB - 1 downto 0);
        B4:         IN signed(NB - 1 downto 0);
        B5:         IN signed(NB - 1 downto 0);
        B6:         IN signed(NB - 1 downto 0);
        B7:         IN signed(NB - 1 downto 0);
        B8:         IN signed(NB - 1 downto 0);
```

```
        DOUT:           OUT signed(NB - 1 downto 0);

        VOUT:           OUT std_logic );
end myfir;

architecture rtl of myfir is

type data     is array (0 to NT-1) of signed(NB -1   downto 0);--type for
    data in
type operand is array (0 to NT-1) of signed(2*NB -1   downto 0);--type
    for sum and products
type c        is array (0 to NT-1) of signed(NB -1   downto 0);--type for
    coefficients

signal prod:                    operand :=(others=>(others=>'0'));
signal sum:                             operand :=(others=>(others=>'0'))
    ;
signal Q:                               data :=(others=>(others=>'0'));

signal array_coeff:     c       :=(others=>(others=>'0'));

signal sig_vout                 : std_logic;


begin
  array_coeff(0) <= B0;
  array_coeff(1) <=     B1;
  array_coeff(2) <=     B2;
  array_coeff(3) <= B3;
  array_coeff(4) <= B4;
  array_coeff(5) <=     B5;
  array_coeff(6) <= B6;
  array_coeff(7) <= B7;
  array_coeff(8) <= B8;


                prod(0)<=array_coeff(0)*DIN;
                products: for j in 1 to NT - 1 generate
                        prod(j)<=Q(j-1)*array_coeff(j-1);
                end generate;
                sums:     for j in 1 to NT - 1 generate
                        sum(j)<=prod(j) + sum(j-1);
                end generate;



in_reg: process(CLK,RST)
begin
            if RST ='0' then
```

```
                                Q(0)<=(others=>'0');
            elsif CLK'event and CLK='1' then
                        if VIN = '1' then
                                Q(0)<=DIN;
                        end if;
                         temp: for j in 0 to NT - 2 loop
                                        Q(j+1)<= Q(j);
                         end loop;
            end if;
end process;

out_reg: process(CLK,RST)
begin
            if RST ='0' then
                        VOUT <= '0';
                        DOUT <=(others=>'0');
            elsif CLK'event and CLK='1' then
                        if VIN = '1' then
                                sig_vout <= '1';
                                VOUT <= sig_vout;
                                DOUT <= resize(sum(NT-1)(2*NB -2  downto
                                    NB-1),NB);
                        end if;
                end if;
end process;

end rtl;

configuration CFG_myfir of myfir is
  for rtl
  end for;
end CFG_myfir;
```

A.3 FIR filter L=3 Unfolded and Pipelined

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;
use WORK.constants.all;
use WORK.all;

entity myfir is
port (
        CLK:            IN std_logic;
        RST:            IN std_logic;
        VIN:            IN std_logic;

        DIN0:       IN signed(NB - 1 downto 0); --multiple inputs
```

```
        DIN1:       IN signed(NB - 1 downto 0);
        DIN2:       IN signed(NB - 1 downto 0);

        B0:         IN signed(NB - 1 downto 0);  --coefficients
        B1:         IN signed(NB - 1 downto 0);
        B2:         IN signed(NB - 1 downto 0);
        B3:         IN signed(NB - 1 downto 0);
        B4:         IN signed(NB - 1 downto 0);
        B5:         IN signed(NB - 1 downto 0);
        B6:         IN signed(NB - 1 downto 0);
        B7:         IN signed(NB - 1 downto 0);
        B8:         IN signed(NB - 1 downto 0);

        DOUT0:      OUT signed(NB - 1 downto 0); --multiple outputs
        DOUT1:      OUT signed(NB - 1 downto 0);
        DOUT2:      OUT signed(NB - 1 downto 0);

        VOUT:           OUT std_logic );
end myfir;

architecture rtl of myfir is

component reg is
        port(
                clock,reset,load : in std_logic;
                i : in std_logic_vector(31 downto 0);
                o : out std_logic_vector(31 downto 0)
        );
end component;

type data     is array (0 to NT-1) of signed(NB -1   downto 0);--type for
    data in
type operand is array (0 to NT-1) of signed(2*NB -1   downto 0);--type
    for sum and products
type c        is array (0 to NT-1) of signed(NB -1   downto 0);--type for
    coefficients
type dataout     is array (0 to 2) of signed(NB -1   downto 0);--type for
    data out




signal prod0,prod1,prod2:                       operand :=(others=>(
    others=>'0'));
signal sum0,sum1,sum2:                          operand :=(others=>(
    others=>'0'));
signal Q:                                 data     :=(others=>(others=>'0'))
    ;
signal array_coeff:                             c        :=(others=>(
```

```
    others=>'0'));
signal sig_vout                    : std_logic;
signal sig_dout:                                    dataout     :=(others=>(
    others=>'0'));
signal mult0,mult1,mult2:                                       operand :=(others
    =>(others=>'0'));

begin

                array_coeff(0) <= B0;
                array_coeff(1) <= B1;
                array_coeff(2) <= B2;
                array_coeff(3) <= B3;
                array_coeff(4) <= B4;
                array_coeff(5) <= B5;
                array_coeff(6) <= B6;
                array_coeff(7) <= B7;
                array_coeff(8) <= B8;


                        prod0(0)<=array_coeff(0)*DIN0;    0reg0:reg port
                            map(clock,reset,prod0(0),mult0(0));
                        prod1(0)<=array_coeff(0)*DIN1;    1reg0:reg port
                            map(clock,reset,prod1(0),mult1(0));
                        prod2(0)<=array_coeff(0)*DIN2;    2reg0:reg port
                            map(clock,reset,prod2(0),mult2(0);
                        prod1(1)<=array_coeff(1)*DIN0;    1reg1:reg port
                            map(clock,reset,prod1(1),mult1(1));
                        prod2(1)<=array_coeff(1)*DIN1;    2reg1:reg port
                            map(clock,reset,prod2(1),mult2(1));
                        prod2(2)<=array_coeff(2)*DIN0;    2reg2:reg port
                            map(clock,reset,prod2(2),mult2(2));

                products: for j in 1 to NT - 1 generate
                        prod0(j)<=Q(j-1)*array_coeff(j);
                            reg0:reg port map(clock,reset,prod0(j),mult0(j
                            ));

                        secondrow:if(j+1 < NT-1) generate
                                prod1(j+1)<=Q(j-1)*array_coeff(j+1); reg1
                                    :reg port map(clock,reset,prod1(j+1),
                                    mult1(j+1));
                        end generate;
                        thirdrow:if(j+2 < NT-1) generate
                                prod2(j+2)<=Q(j-1)*array_coeff(j+2); reg2
                                    :reg port map(clock,reset,prod2(j+2),
                                    mult2(j+2));
                        end generate;
                end generate;
```

```
                    sum0(0)<=mult0(0);
                    sum1(0)<=mult1(0);
                    sum2(0)<=mult2(0);


            sums:     for j in 1 to NT - 1 generate
                    sum0(j)<=mult0(j) + sum0(j-1);
                    sum1(j)<=mult1(j) + sum1(j-1);
                    sum2(j)<=mult2(j) + sum2(j-1);
            end generate;


in_reg: process(CLK,RST)
begin
        if RST ='0' then
                            Q<=(others=>(others=>'0'));
        elsif CLK'event and CLK='1' then
                    if VIN = '1' then
                            Q(0)<=DIN2;
                            Q(1)<=DIN1;
                            Q(2)<=DIN0;
                            Q(3)<=Q(0);
                            Q(4)<=Q(1);
                            Q(5)<=Q(2);
                            Q(6)<=Q(3);
                            Q(7)<=Q(4);
                            Q(8)<=Q(5);

                    end if;
        end if;
end process;

out_reg: process(CLK,RST)
begin
        if RST ='0' then
                    VOUT <= '0';
                    DOUT0 <=(others=>'0');
                    DOUT1 <=(others=>'0');
                    DOUT2 <=(others=>'0');
        elsif CLK'event and CLK='1' then
                    if VIN = '1' then
                            sig_vout <= '1';
                            VOUT <= sig_vout;
                            sig_dout(0) <= resize(sum0(NT-1)(2*NB -2
                                downto NB-1),NB);
                            sig_dout(1) <= resize(sum1(NT-1)(2*NB -2
                                downto NB-1),NB);
                            sig_dout(2) <= resize(sum2(NT-1)(2*NB -2
```

```
                                        downto NB-1),NB);
                                    DOUT0 <= sig_dout(0);
                                    DOUT1 <= sig_dout(1);
                                    DOUT2 <= sig_dout(2);
                            end if;
                    end if;
end process;

end rtl;

configuration CFG_myfir of myfir is
  for rtl
  end for;
end CFG_myfir;

configuration CFG_myfir of myfir is
  for rtl
      for all : reg
       use configuration WORK.CFG_reg;
      end for;
   end for;
end CFG_myfir;
```

A.4 Simple register

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;
use WORK.constants.all;
use WORK.all;
----simple register----------
entity reg is
        port(
                clock,reset : in std_logic;
                i : in signed(2*NB-1 downto 0);
                o : out signed(2*NB-1 downto 0)
        );
end reg;

architecture behavioral of reg is
        signal temp : signed(2*NB-1 downto 0);

begin
        process (clock)
        begin
                if(clock = '1' and clock'event)then
                        if(reset = '0')then
                                temp <= (others=>'0');
```

```
                    elsif(reset = '1' )then
                        temp <= i;
                    end if;
                end if;
        end process;

        o <= temp;

end behavioral;


configuration CFG_reg of reg is
  for behavioral
  end for;
end CFG_reg;
```