Management and analysis of physics datasets, Part. 1

Seventh Laboratory

Antonio Bergnoli bergnoli@pd.infn.it

13/1/2020

Laboratory Introduction

Goals

- Some arithmetic operations in VHDL.
- FIR (Finite Impulse Response) filter in VHDL.

VHDL naming convention

Signals/components	Name
Clock	*clk*
Reset	*rst*
Input Port	*port_in*
Output Port	*port_out*
VHDL file name	*entityname.vhd*
Test bench file name	*tb_entityname.vhd*
Signal between 2 comps	*sign_cmp1_cmp2*
ila signal	*ila_signal*
vio signal	*vio_signal*
...	...

Arithmetic operations

Arithmetic operations in VHDL (1)

Or rather, just the arithmetic operations preparatory for this laboratory.

- Numbers are represented as arrays.
- The numbers, in this laboratory, must be signed.
- The arithmetic operations exploited are sum and multiplication.
- The function to convert a *std_logic_vector* signal to a *signed* signal and vice versa are compulsory.

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
```

Arithmetic operations in VHDL (2)

Declaration

```
signal x : std_logic_vector(N-1 downto 0);
signal s : signed(N-1 downto 0);
```

Conversion

```
x <= std_logic_vector(s);
s <= signed(x);
```

Arithmetic operations in VHDL (3)

std_logic_vector Assignment

```
x <= "01010110";
x <= x"a6";
```

signed Assignment

```
s <= to_signed(10, N);
s <= to_signed(-10,N);
```

Arithmetic operations in VHDL (3)

- $13 \times 3 \;= 39$;
- $1101_2 \times 11_2 \;= 100111_2$;
- (4 elements) $\times$ (2 elements) $=$ 6 elements;
- $\Rightarrow$
- (N1 - 1 `downto` 0) $\times$ (N2 -1 `downto` 0) $=$ (N1 $+$ N2 - 1 `downto` 0);

Arithmetic operations in VHDL (4)

If a number is less than zero:

- $20 \times 0.75 \ = 15$;
- $10100_2 \times 0.11_2 \ = 1111_2$;
- **How realize this operation in VHDL?**
- Scale-up the number less than zero of certain quantity Q. For example $Q \ = \ 3$.
- $0.11_2 \ << \ Q \ = \ 110_2$. That is $0.75 * 2^3 = 6$.
- Then: $10100_2 \times 110_2 \ = 1111000_2$
- Finally scale-down of the same quantity Q, that is $1111000_2 \ >> \ Q \ = \ 1111_2$. That is $120 : 2^3 = 15$.

Arithmetic operations in VHDL (5)

If a number is less than zero:

- $20 \times 0.057 \ = 1.11$;
- But 0.057×2^{Q} for each Q chosen is never integer. Therefore it is necessary a trade-off between the number of bits and the approximation wished.

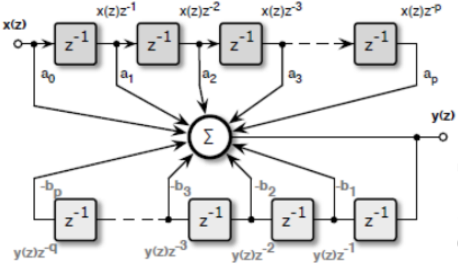
Arithmetic operations in VHDL (6)

- In order to do this kind of arithmetic operations is necessary change the dimension of the arrays and scale (up or down) the numbers.
- The numbers in this laboratory are represented as **signed** type.
- You can find a complete list of the operation at this . Except the sum $(+)$ and the multiplication $(*)$, may be useful two functions:
  1. RESIZE;
  2. SHIFT_RIGHT.

Finite Impulse Response filter

Digital Sampled-data Filter implementation

using
- addition
- multiplication
- delay



using
→ 2 digital delay lines
(one for input
one for output)

$$\frac{y(z)}{x(z)} = H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_p z^{-p}}{1 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_q z^{-q}},$$

$$y(z) = -(b_1 z^{-1} + b_2 z^{-2} + \ldots + b_q z^{-q}) y(z) +$$

$$+(a_0 + a_1 z^{-1} + \ldots + a_p z^{-p}) x(z).$$

Warning
several sum over variables limited to n-bit values → $\boxed{\text{approximation !}}$

FIR filter(2)

Finite impulse response

From Wikipedia, the free encyclopedia

In signal processing, a **finite impulse response** (**FIR**) **filter** is a filter whose impulse response (or response to any finite length input) is of *finite* duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response (that is, the output in response to a Kronecker delta input) of an Nth-order discrete-time FIR filter lasts exactly $N + 1$ samples (from first nonzero element through last nonzero element) before it then settles to zero.

FIR filters can be discrete-time or continuous-time, and digital or analog.

Contents [hide]

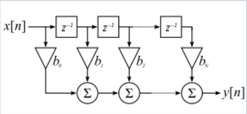
Definition [edit]

For a causal discrete-time FIR filter of order N, each value of the output sequence is a weighted sum of the most recent input values:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \\ &= \sum_{i=0}^{N} b_i \cdot x[n-i], \end{aligned}$$

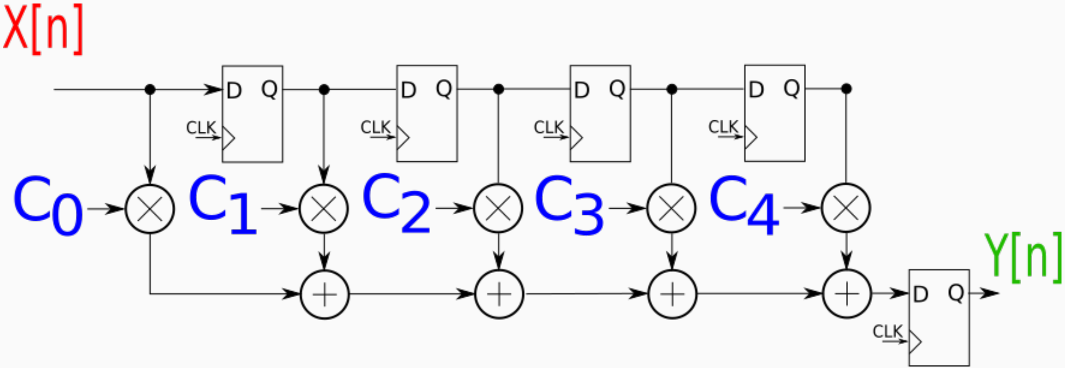
where:

- $x[n]$ is the input signal,
- $y[n]$ is the output signal,
- N is the filter order; an Nth-order filter has $(N+1)$ terms on the right-hand side
- b_i is the value of the impulse response at the i'th instant for $0 \le i \le N$ of an Nth-order FIR filter. If the filter is a direct form FIR filter then b_i is also a coefficient of the filter.



A direct form discrete-time FIR filter of order N. The top part is an N-stage delay line with $N + 1$ taps. Each unit delay is a z^{-1} operator in Z-transform notation.

FIR filter(3)



This FIR filter circuit is described by the equation: $y[n+1] = \sum_{i=0}^{4} x[n-i] * C_i$

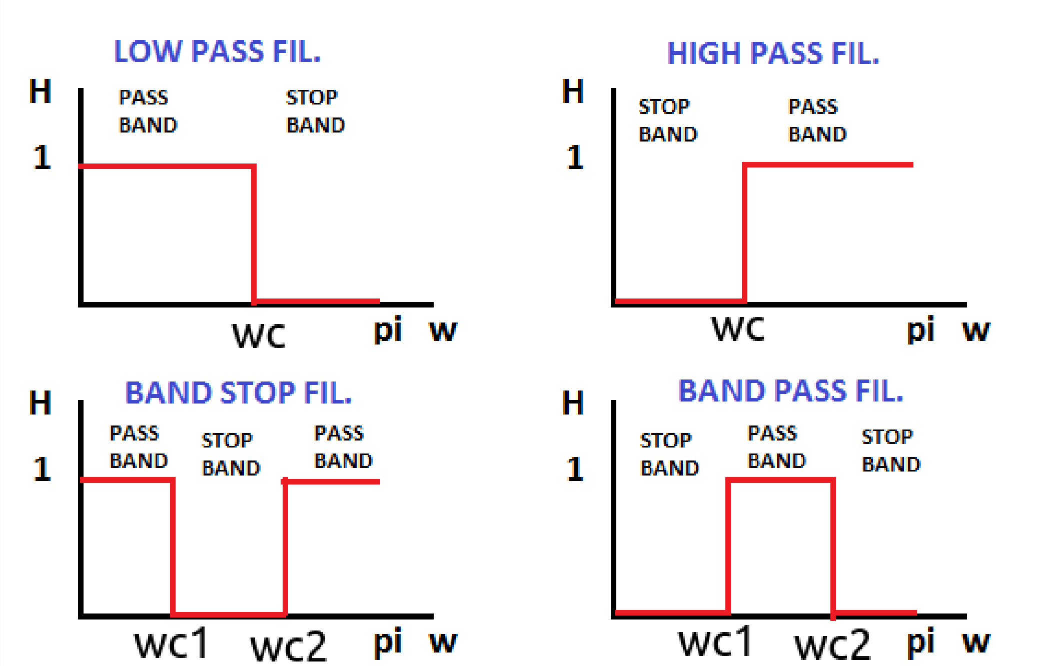
FIR filter(4) $y[n+1] = \sum_{i=0}^{N} x[n-i] * C_i$

A numerical example. Data:

- $x[n] = 1 \; \forall \; n \geq 0$;
- $C_0 = 1, C_1 = 2, \; C_2 = 3, \; C_3 = 4, \; C_4 = 5,$

Then:

FIR filter type

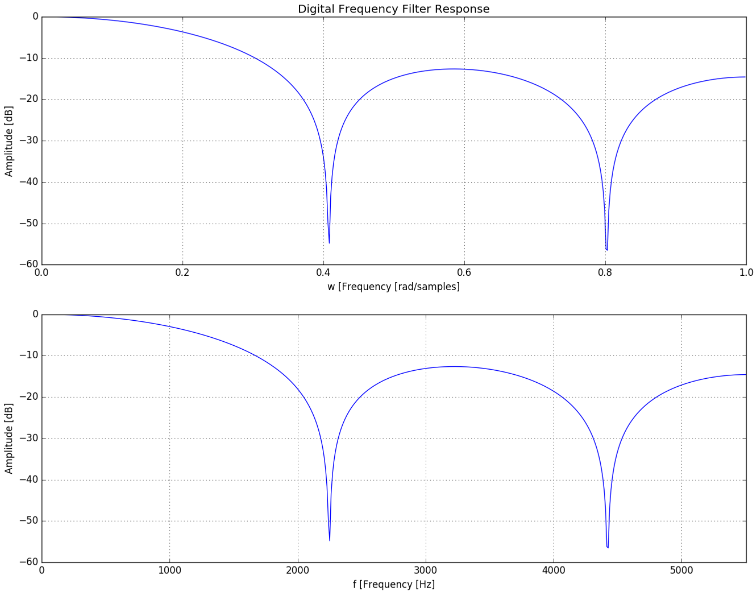


FIR filter coefficients (1)

You can use the python script **coeff.py**. (It is in the folder "utilities"). It exploits the python function `firwin`.

- It is set to compute the coefficients of a 5 tap digital low-pass filter.
- 5 tap means 5 coefficients.
- 0.1 is the cutt-off frequency, that is $w_c = 0.1 * \pi$. For example a typical sampling frequency of an audio wav file is $f_s = 11025$ Hz. So 0.1 means $f_c = 0.1 * f_s/2 \approx 550$ Hz.
- The script gives the coefficients: $C_0 = C_4 = 0.19335315$, $C_1 = C_3 = 0.20330353$ and $C_2 = 0.20668665$.

FIR filter coefficients (2)



Homework

Homework

- The frame of the *fir* entity and the frame of the testbench are given. You have to fill them.
- You can implement which type of filter you prefer, with a number of tap greater or equal to 5.
- A behavioral simulation is enough.
- Write a brief report (max. 3 pages), where you describe at least which filter you chose, the value of the coefficients and their conversion in VHDL. Report also a screenshot of the schematic and a screenshot of the testbench simulation result.

Hints

1. Write the code following the circuit represented in the silde 17.
2. Regarding on Flip-Flop remember how to re-use the code with the *component*.
3. Do the first testbenches with trivial value of the input and of the coefficients.
4. When you have done the point 1, you can set the actual value of the coefficients and re-do the simulation.
5. In order to test the correctness of the filter, with the actual coefficients, is better choose values of the input X greater, in absolute value, than 2^{10}.

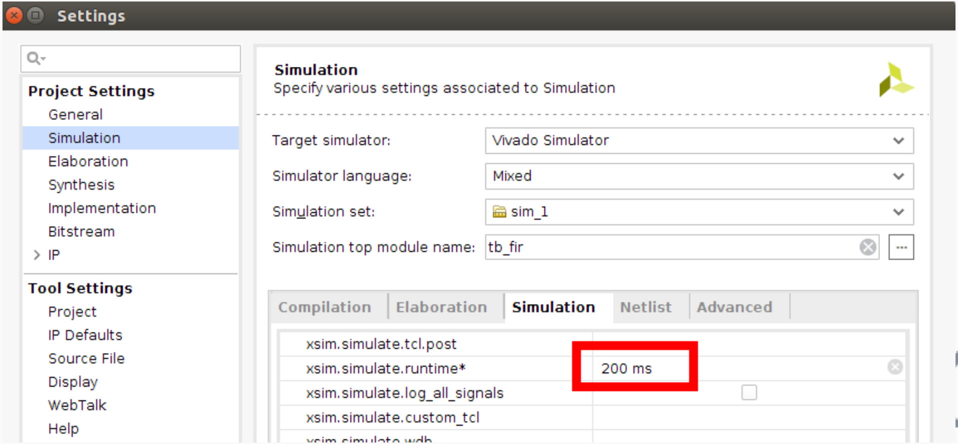
Practical example (1)

Not mandatory.

- FIR filter are widely used with audio files.
- In the folder "utilities" there is a .wav file. It's a test file with a frequency sweep from 100 Hz to 3.5 KHz. It was downloaded from youtube and modified. If you want download other wav file, for this laboratory, it is better choose a sampling frequency (f_s) of 11025 Hz.
- In the folder "utilities", there is the script python **write_input_txt.py**, to convert a mono wav file into a txt file.
- Put the txt file in the vivado project folder ".../proj.sim/sim_1/behav/xsim/".

Practical example (2)

- You need the testbench file *tb_with_file* (it is in the folder "utilities").
- In vivado project settings you have to change the simulation settings, changing the duration of the simulation. 20 ms are enough.

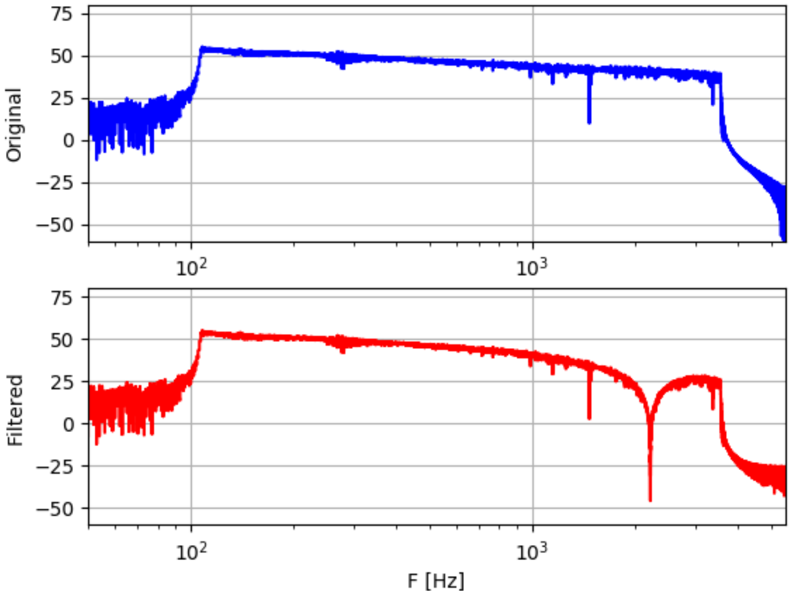


Practical example (3)

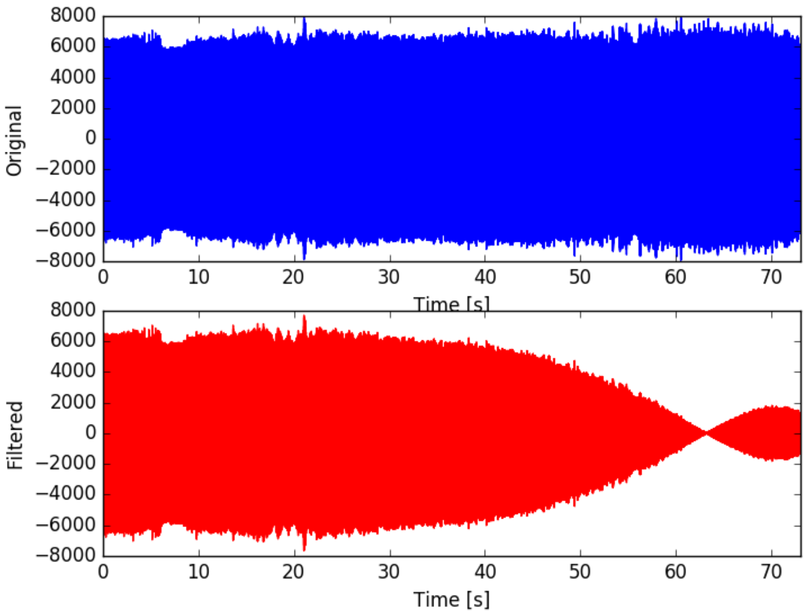
- Run the Behavioral simulation.
- Copy the output txt file ("output_file.txt") produced by the testbench in the folder "utilities".
- Run the python script **write_filtered_wav.py**
- A wav file is created.

Listen the two audio files. Are they different?
Now, if you run the script file **spectrum.py** the frequency spectrum of the audio files are plotted. Meanwhile if you run the file **time_domain.py** the amplitudes of the signals in time domain are plotted.

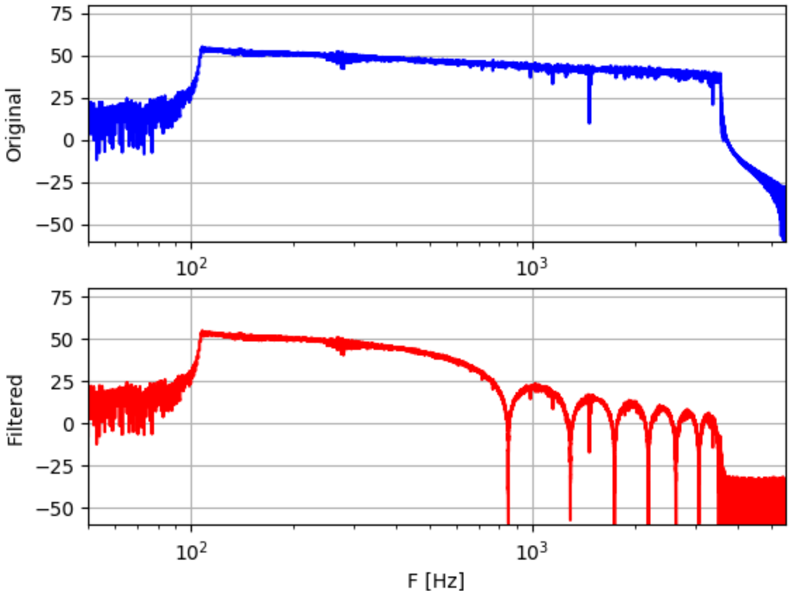
5-tap LP FIR filter (1)



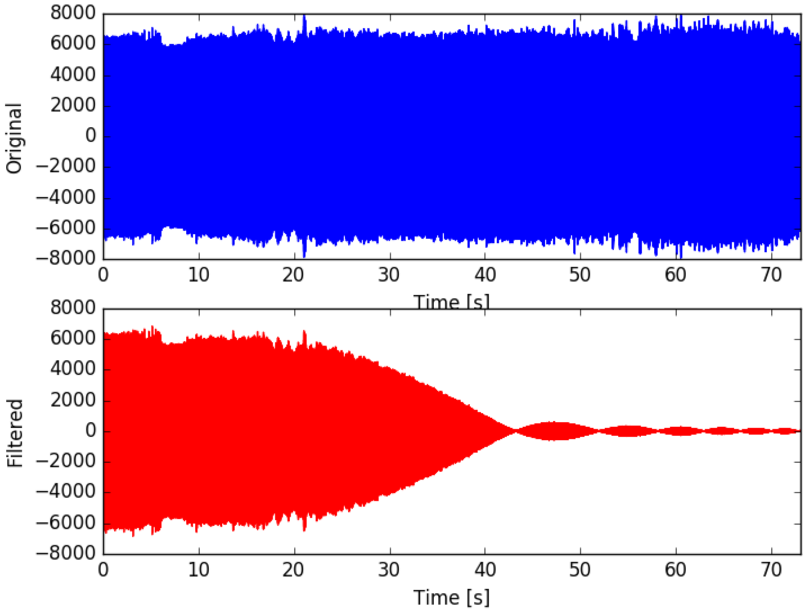
5-tap LP FIR filter (2)



25-tap LP FIR filter (1)



25-tap LP FIR filter (2)



Suggested works

1. rewrite the filter architecture from scratch
2. change the input data type from unsigned to signed
3. play with signals: try to generate some interesting waveforms and feed the filter with them
4. change the coefficents of the filters and observe how the outputs changes