Management and analysis of physics datasets, Part. 1

Eleventh Laboratory

Antonio Bergnoli bergnoli@pd.infn.it, Filippo Marini filippo.marini@pd.infn.it

19/01/2021

Laboratory Introduction

Goals

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

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 one:

- $20 \times 0.75 \ = 15$;
- $10100_2 \times 0.11_2 \ = 1111_2$;
- **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 one:

- $20 \times 0.057 \ = 1.11$;
- 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

FIR filter(1)

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]

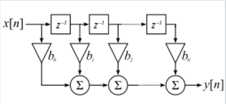
Definition [edit]

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:

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

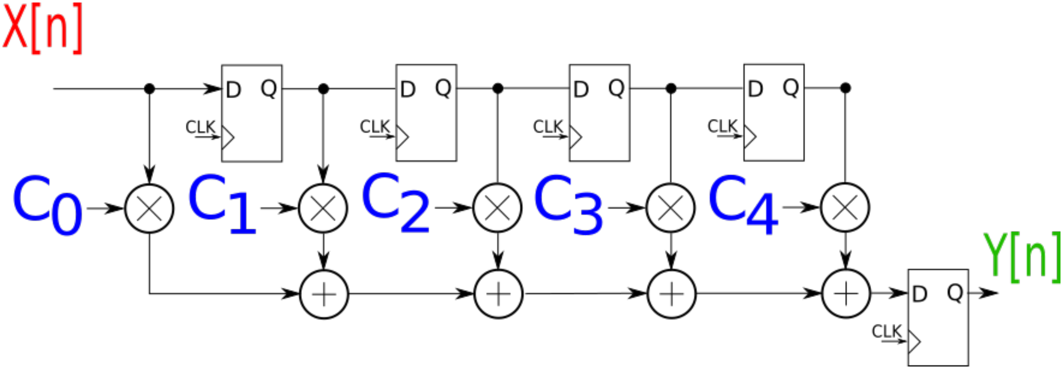
where:

- $x[n]$ 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



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(2)



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

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

FIR filter(3)

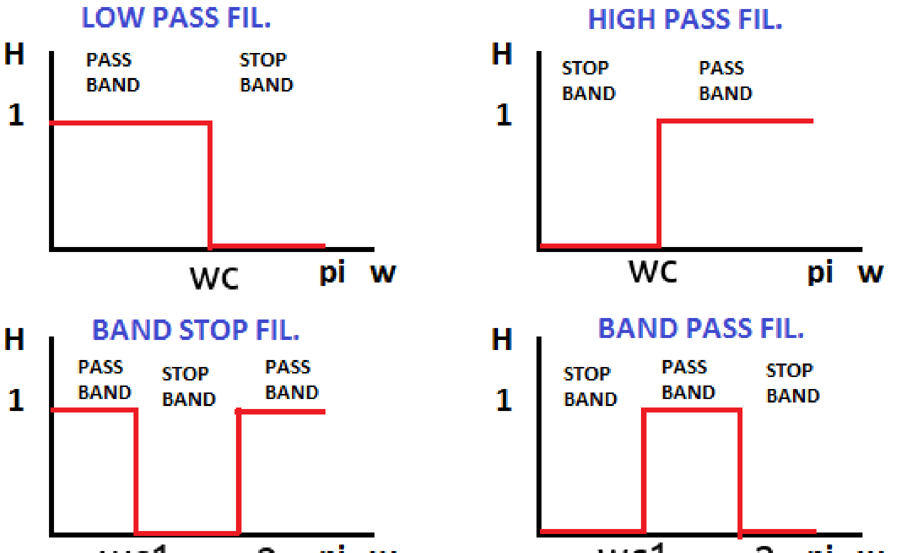
A numerical example. Data:

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

Then:

- $y[0] = 0$;
- $y[1] = x[0] * C_0 = 1$;
- $y[2] = x[1] * C_0 + x[0] * C_1 = 3$;
- $y[3] = x[2] * C_0 + x[1] * C_1 + x[0] * C_2 = 6$
- $y[4] = x[3] * C_0 + x[2] * C_1 + x[1] * C_2 + x[0] * C_3 = 10$
- $y[5] = x[4] * C_0 + x[3] * C_1 + x[2] * C_2 + x[1] * C_3 + x[0] * C_4 = 15$
- $y[n] = x[n-1] * C_0 + x[n-2] * C_1 + x[n-3] * C_2 + x[n-4] * C_3 + x[n-5] * C_4 \; \forall \; n \geq 5$

FIR filter type



FIR filter coefficients (1)

You can use the python function `firwin` to generate the FIR filter cœfficients.

- It is set to compute the cœfficients of a N tap digital low-pass filter.
- N tap means N cœfficients.
- 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 cœfficients: $C_0 = C_4 = 0.19335315$, $C_1 = C_3 = 0.20330353$ and $C_2 = 0.20668665$.

FIR filter coefficients (2)