UNIVERSITÀ DI PISA

MSc in Computer Engineering
Electronic systems

FIR low pass filter

Project discussion and VHDL implementation

Gioele Carignani

2017/2018

Index

1 Introduction

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response, or response to any other finite length input, is of finite duration, because it settles to zero in finite time. The impulse response 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. 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:

$$y[n] = \sum_{i=0}^{N} c_i \cdot x[n-i]$$

1.1 Filter Design

When a particular frequency response is desired, several different design methods are possible, for example:

- Window design method: we first design an ideal IIR filter and then truncate the result by multiplying it with a finite length window function.
- Frequency Sampling method: this technique is the most direct technique imaginable when a desired frequency response has been specified. It consists simply of uniformly sampling the desired frequency response, and performing an inverse DFT to obtain the corresponding (finite) impulse response.
- Parks-McClellan method: The Remez exchange algorithm is commonly used to find an optimal equiripple set of coefficients. Here the user specifies a desired frequency response, a weighting function for errors from this response, and a filter order N. The algorithm then finds the set of N+1 coefficients that minimize the maximum deviation from the ideal.

1.2 Applications

Finite-impulse response (FIR) digital filter is widely used in several digital signal processing applications, such as speech processing, loud speaker equalization, echo cancellation, adaptive noise cancellation, and various communication applications, including software-defined radio (SDR) and so on. Many of these applications require FIR filters of large order to meet the stringent frequency specifications. Very often these filters need to support high sampling rate for high-speed digital communication. The number of multiplications and additions required for each filter output, however, increases linearly with the filter order.

2 Architectures

We can have different type of architectures depending on the speed and power constraints. To fulfill the constraints we can:

- Reduce total number of operations, in particular multiplications that are heavier
- Fixed-point arithmetic is cheaper and faster
- The area of a fixed point parallel multiplier is proportional to the product of the coefficient and data word lengths one could try to reduce their word length

Suppose we want to realize a filter of order N.

2.1 Direct form

This structure requires N memory locations for storing previous inputs and in terms of complexity it does N+1 multiplications and N additions. For building this structure we simply use the definition:

$$y[n] = \sum_{i=0}^{N} c_i \cdot x[n-i]$$

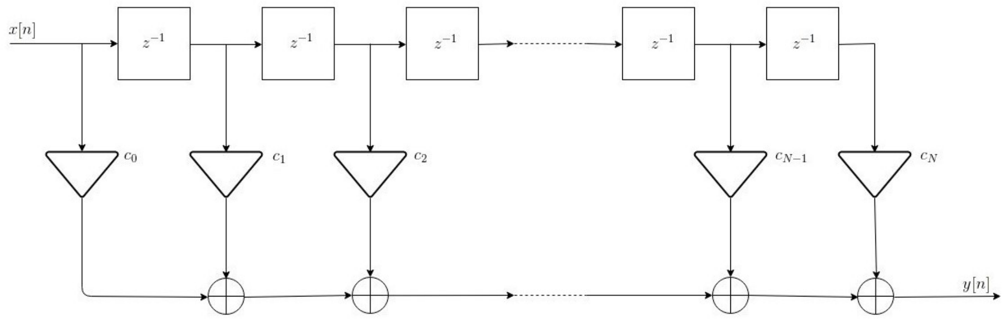

Figure 1: scheme of direct form

2.2 Transposed form

This structure is very similar at the first look w.r.t. the previous one but in the former there was a big addition in the end, here instead there is a set of small additions separated by delay elements.

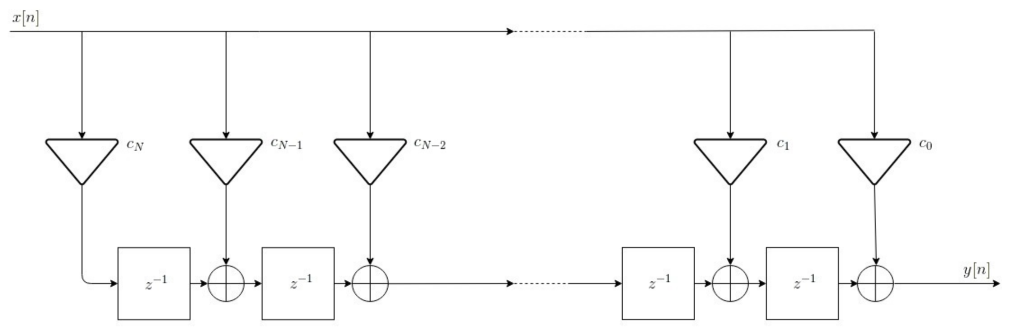

Figure 2: scheme of transposed form

2.3 Symmetric taps (linear phase)

FIR filters are often designed to have symmetry in the filter taps so we can exploit this symmetry in order to reduce the number of multiplications.

$$y[n] = c_0(x[n] + x[0]) + c_1(x[n-1] + x[1]) + ... + c_{N/2}(x[N/2])$$

The last term of the previous equation is needed only if N is even. With this type of implementation the number of multiplications is reduced to $\lfloor N/2 + 1 \rfloor$

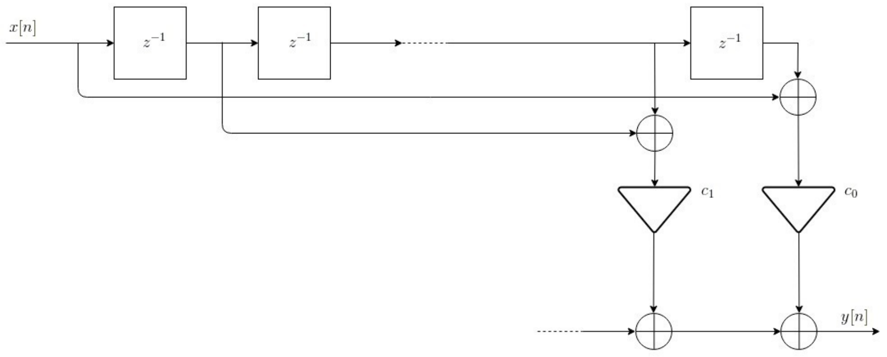

Figure 3: direct form with symmetric taps

2.4 Cascade form

We express the transfer function as products of second-order polynomial system functions via factorization:

$$H(z) = \frac{y[n]}{x[n]} = \sum_{i=0}^{N} c_i z^{-i} = \prod_{i=1}^{M_c} (\beta_{0i} + \beta_{1i} z^{-1} + \beta_{2i} z^{-2}) \tag{1}$$

Where $M_c = \lfloor (N+1)/2 \rfloor$. Assuming that N is even, this implementation needs N storage elements, $\frac{3N}{2}$ multiplications and N additions for each output value.

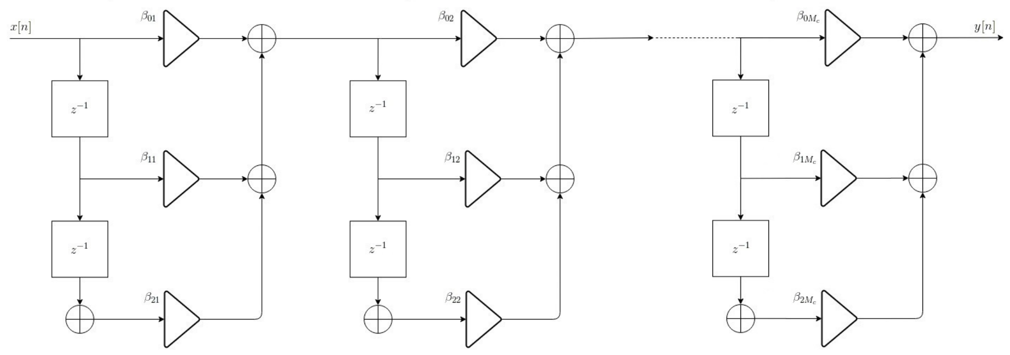

Figure 4: cascade form

To save computational complexity, we express (1) as:

$$H(z) = G\prod_{i=1}^{M_c}(1 + \beta'_{1i}z^{-1} + \beta'_{2i}z^{-2}) \tag{2}$$

where $G = \beta_{01}\beta_{02}\cdots\beta_{0M_c}$,$\beta'_{1i} = \beta_{1i}/\beta_{0i}$, $i = 1, 2, \cdots, M_c$ in this way all β_{0i} are normalized to 1 as one can see in (2). Assuming that N is even we need N delay elements, (N+1) multiplications and N additions, for computing each output value.

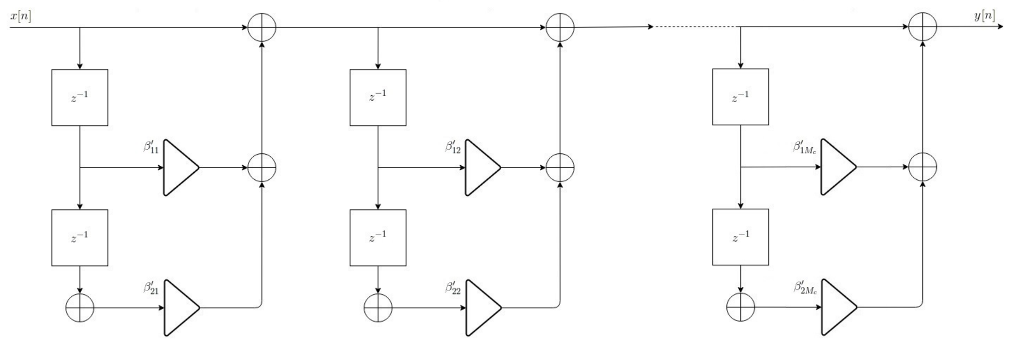

Figure 5: cascade form with lower complexity