EE 422G - Signals and Systems Laboratory

Lab 4 IIR Filters

written by
Kevin D. Donohue
Department of Electrical and Computer Engineering
University of Kentucky
Lexington, KY 40506
October 4, 2011

Objectives:

- Use filter design and analysis tools to create IIR filters based on general filter specification.
- Understand the impact of the placement of poles and zeros on the frequency response of the filter.
- Use Simulink to simulate filter performance using truncation effects of embedded processors.
- Understand the impact of different computational structures for filter implementations on filter performance.

1. Background

Infinite impulse response (IIR) filters are an alternative to finite impulse response (FIR) filters. An IIR implementation can meet filter specifications with less computation than an FIR implementation; however, IIR filters have nonlinear phase responses, are potentially unstable, and have increased sensitive to numerical errors.

Like FIR filters, IIR filters are linear time-invariant (LTI) systems that can recreate a wide range of frequency responses. IIR implementations with specified stop-band, pass-band, and transition-band properties typically require fewer filter taps (coefficients) than an FIR filter meeting similar specifications. This leads to a significant reduction in computational complexity and signal delay through the filter. However, IIR filter TFs have poles resulting in feedback, which cause instability if poles are not inside the unit circle. Feedback can increase the sensitivity to errors introduced from finite arithmetic computations (especially for fixed-point processors). In addition, IIR filters result in nonlinear phase distortion (frequency components delayed by different amounts, especially near the transition bands).

Up to this point, filters used in the laboratory assignments have been implemented with the ***Direct Form***, which is suggested by converting the transfer function (TF) directly to a difference equation, using the TF in the following form:

$$\hat{H}(z)=\frac{\sum_{m=0}^{M} b_m z^{-m}}{1+\sum_{n=1}^{N} a_n z^{-n}}=\left(\frac{\sum_{m=0}^{M} b_m z^{-m}}{1}\right)\left(\frac{1}{1+\sum_{n=1}^{N} a_n z^{-n}}\right) \quad (1)$$

The rational polynomial in z for Eq. (1) is factored into an all-zero filter (FIR) followed by an all-pole filter. Recall the product of TFs corresponds to connecting the systems in series in the time-domain, where the output of one filter becomes the input to the next. The factoring of Eq. (1) suggests the *direct form I* implementation shown in Fig. 1.

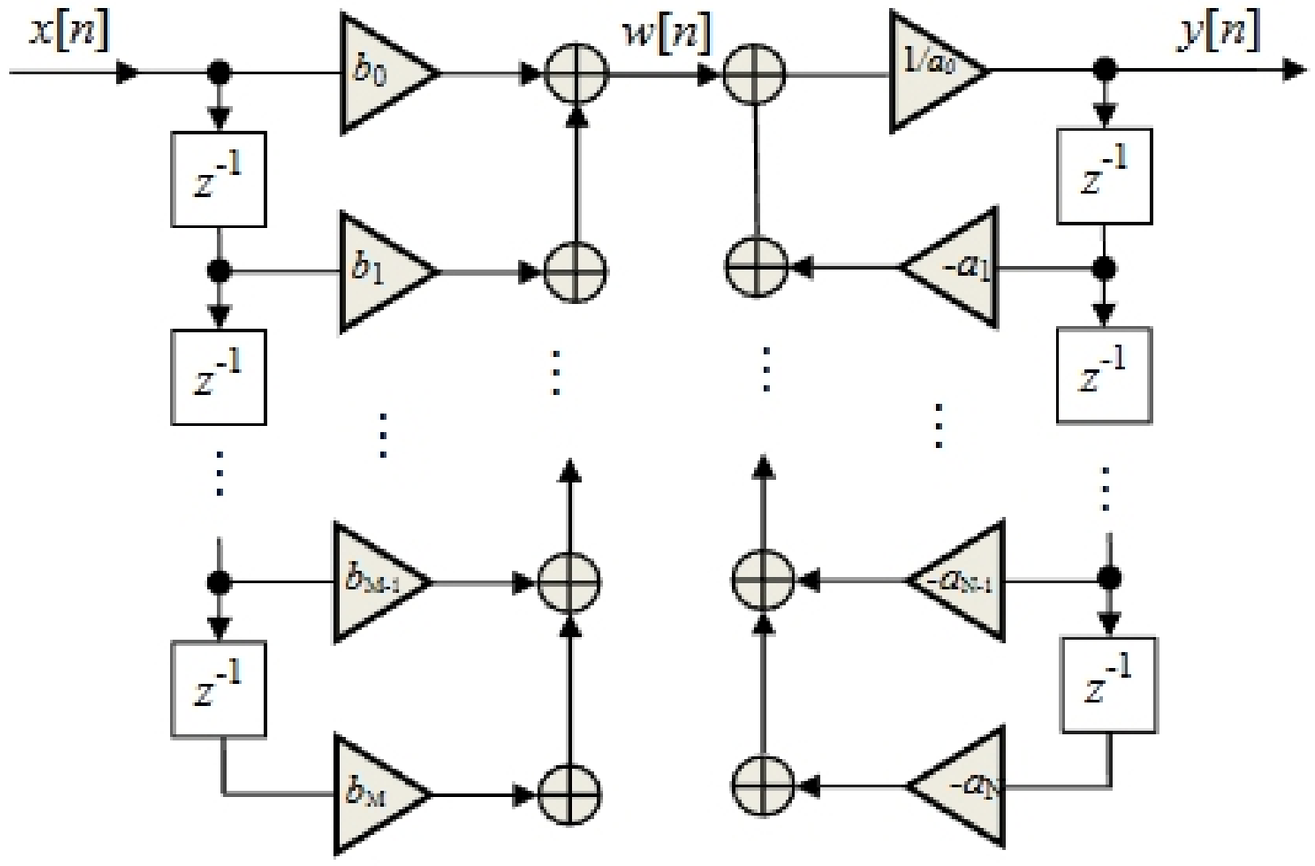


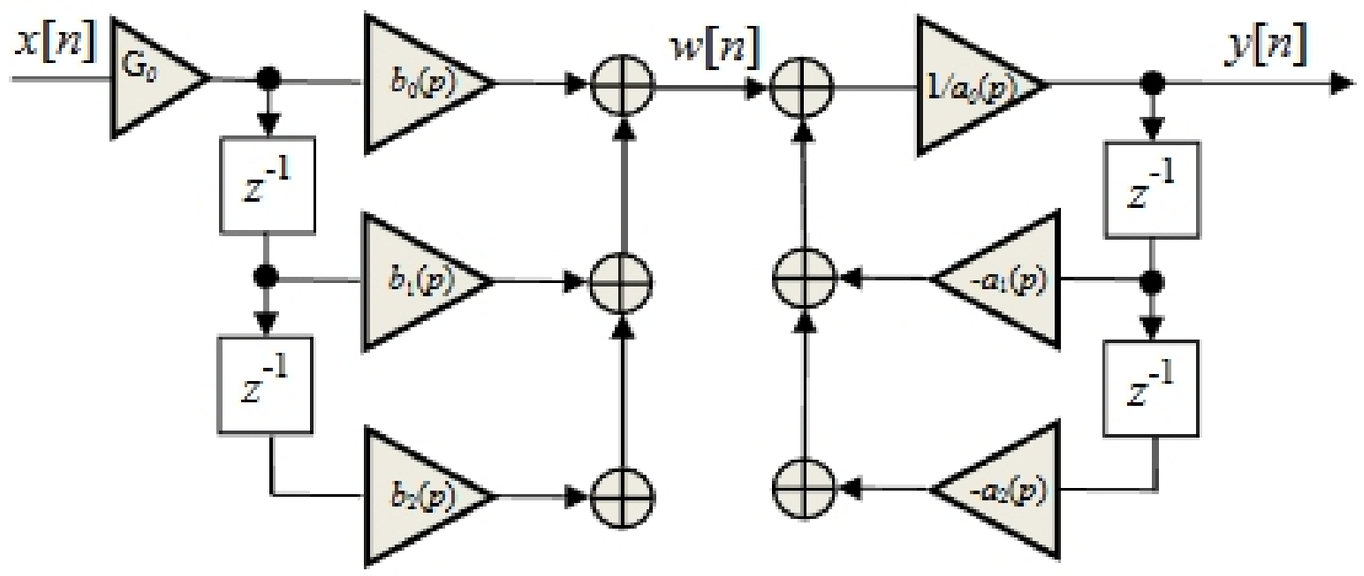
Figure 1. Direct form I implementation of an IIR filter. The square blocks represent unit delays, the triangles represent multiplies, and the circles represent accumulators. The variable $w[n]$ is an intermediate value being the output of the all-zero component and the input to the all-pole component of the filter, and a_0=1 for corresponding to the IIR filter of Eq. (1).

For higher order filters, the direct-form implementation involves long delays (M samples for the input and N samples for the output). This may lead to numerical instability from rounding the filter coefficients. For higher-order polynomials the roots may shift widely for small changes (i.e. rounding errors) in the filter coefficients. Therefore, it is hard to predict how small coefficient errors will impact the pole positions of a high order direct-form implementation. Different implementations (computational structures) can be realized by factoring and representing the polynomials of Eq. (1) in different ways.

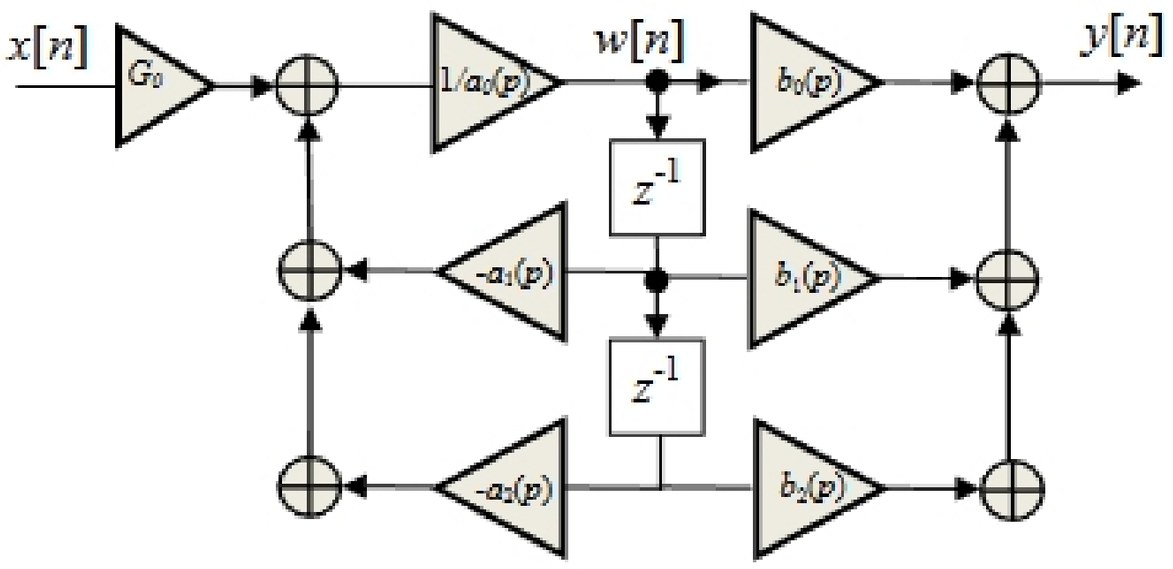
Consider factoring numerator and denominator polynomials into second-order stages as follows:

$$\hat{H}(z) = \frac{\sum_{m=0}^{M} b_m z^{-m}}{1 + \sum_{n=1}^{N} a_n z^{-n}} = G_0 \prod_{p=1}^{P} \frac{b_0(p) + b_1(p) z^{-1} + b_2(p) z^{-2}}{1 + a_1(p) z^{-1} + a_2(p) z^{-2}} \qquad (2)$$

where coefficient argument p denotes each filter stage, and is derived from the original direct-form single stage filter coefficients by factoring. When factoring into second order stages, pairs of real poles or complex conjugate poles are combined to ensure real coefficients. G_0 is the overall gain, which can be factored out of all the b coefficients. The form of Eq. (2) suggests cascading second-order IIR filters in series. This implementation is referred to as the ***Cascade*** form. One way to improve numerical stability over a higher-order direct form implementation is to implement the IIR filter as a cascade of second-order *direct form* sections. The data flow for a second-order, *direct-form* implementation is shown in Figure 2. Note that in the *direct form II* implementation, as shown Fig. 2b, the delayed samples are neither input nor output samples, but are instead the intermediate values $w[n]$. This implementation can be derived by first applying the all-pole component of the TF and then making that the input of the all-zero component (just the opposite multiplication order of the *direct form I* implementation shown in Eq. (1).



(a)



(b)

Figure 2. (a) Single stage *direct form I cascade* implementation of an IIR filter. (b) Single stage *direct form II cascade* implementation of an IIR filter. G_0 is multiplier affecting the overall gain of the filter (does not change relative spectrum or waveform values). This is only applied at one stage. And a_0=1 in order to correspond to the IIR filter of Eq. (2).