

Lab 7: IIR Filter and its Comparison with the FIR Implementation

EE 352: Digital Signal Processing Laboratory

Lab Goals

In this lab, we will study about implementation of infinite impulse response (IIR) filters, their advantages and drawbacks. Also, we will understand how this compares to FIR implementations of a system.

- Direct form I implementation of an IIR filter
- FIR vs IIR implementation of a system

1 Introduction

1.1 Difference equation implementation of a linear time-invariant (LTI) system

Any discrete-time LTI system can be represented in terms of a relation between its past, current and future inputs as well as outputs, as follows.

$$\sum_{i=-\infty}^{\infty} a_i y[n-i] = \sum_{j=-\infty}^{\infty} b_j x[n-j] \quad (1)$$

where a_i and b_j are constants, characteristic of the particular system, $x[n]$ is the input signal and $y[n]$ is the filtered output signal. We shall consider only *causal* systems here, which means both the above summations will operate over the range 0 to ∞ rather than $-\infty$ to ∞ .

1.2 IIR filter

Consider a special case of (1).

1. $a_i = 0$ for $i > M$
2. $b_j = 0$ for $j > N$
3. $a_0 = 1$, and $a_i \neq 0$ for at least one i , $i = 1, \dots, M$

where M and N are natural numbers.

Here, we require at least one of the past outputs to calculate the current output. The transfer equation $H(z)$ then becomes,

$$H(z) = \frac{\sum_{j=0}^N b_j z^{-j}}{\sum_{i=0}^M a_i z^{-i}} \quad (2)$$

If we try to write this fraction in terms of a series in z^{-1} , it becomes infinitely long and the corresponding impulse response has infinite support. Such a filter is called an Infinite Impulse Response (IIR) filter.

The techniques studied in the previous labs to implement a FIR digital filter are not applicable to IIR filter for obvious reasons. Thus, we need to explore other representations of such an LTI system.

The difference equation representation lends itself to satisfy the purpose. If two buffers of lengths $M+1$ and $N+1$, for holding past values of input and output, respectively, are maintained throughout the data acquisition process, then performing finite multiplications and additions, an IIR filter can be readily realized. The two

buffers can be initialized appropriately. This leads us to the first method of implementing an IIR filter using difference equation.

2 Direct-form I IIR implementation

One algorithm for implementation of an IIR filter, defined by (3), on a DSP referred to as direct-form I can be written as,

$$\sum_{i=0}^N a_i y[n-i] = \sum_{j=0}^M b_j x[n-j] \quad (3)$$

1. Allocate memory space to two buffers of sizes $M + 1$ and $N + 1$ and initialize them with zeros or initial conditions, if known.
2. Start the acquisition of data.
3. Go on calculating the values of the output according to (3) and storing them in the buffer, one after the other, until the buffer saturates.
4. Simultaneously, keep storing the values of input till its buffer saturates. (all elements filled)
5. Feed the output display unit (the DSP codec combined with the DSO in our case) with every output sample, as it is calculated.
6. If any of the buffers runs out of memory space, start overwriting the oldest values with the latest values.
7. Continue steps 3 to 5 till the input data-flow continues.

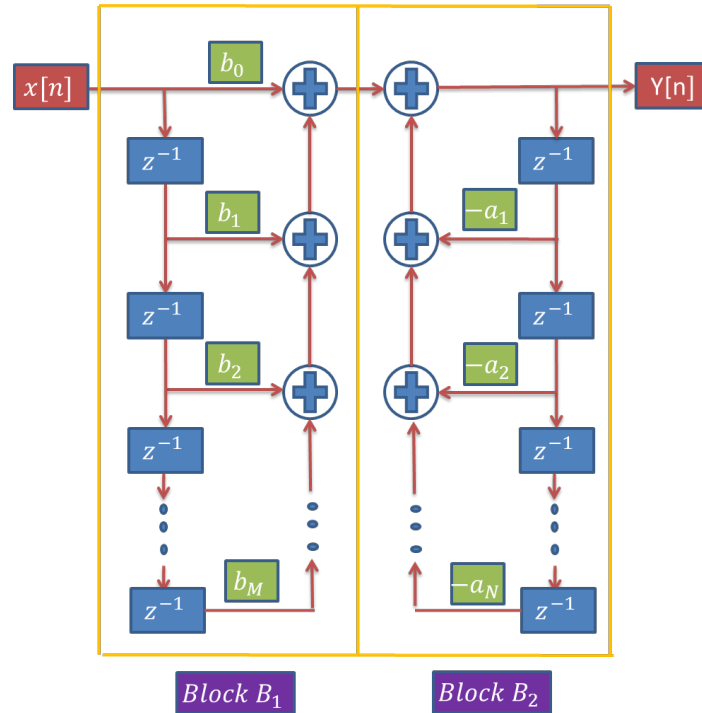


Figure 1: Block diagram for direct-form I structure for the IIR filter given by (3)

2.1 Learning checkpoint

All the following tasks need to be shown to your TA to get full credit for this lab session.

- **Direct form I IIR filter:** Consider the IIR filter given by equation (4).

$$H_1(z) = \frac{1 - 1.2728z^{-1} + 0.81z^{-2}}{1 + 0.81z^{-2}} \quad (4)$$

1. Draw the pole-zero plot of this filter and get it verified by your TA/RA.
2. Using the `freqz` function provided by Matlab, plot the frequency response of the filter and identify the type of the filter (Lowpass / Highpass / Bandpass / Notch).
3. Import the `c_iir` project into the CCS. Open the file `main.c`. It follows the format of the skeleton codes that were provided to you during previous lab sessions. You have to write the function `direct_form_1` at the place indicated to realize the direct form I implementation of the filter from equation (4). You may make use of the arrays declared just above the function as buffers and filter coefficients. Note that the filter numerator and denominator coefficients have been provided suitable gains. This is done to be able to “watch” the output conveniently. **Do not** change these gains nor the scaling of 500 applied to the **output**. Keep in mind that since the first denominator coefficient is not unity, you may have to scale down the output by the same. Also, to compress the output to accommodate it in 16 bits, you may have to further scale down the output by 32768.
4. Declare a synthetic input `input_signal = {1, 2, -1, 0, 3, 4, 0}` in the file and test your code against it by first manually calculating the output signal and then comparing it with the output of the actual run of the program. Use the coefficients declared in the code during manual calculations and not those from equation (4).
5. Now, verify the frequency response of the filter. Take help from your TA/RA to understand how to do this.

3 Direct form II IIR implementation

In direct form I structure, the total memory space required is of length $M + N + 2$ (combining that needed by input and the output buffers). To reduce it, we turn to the direct-form II structure which we shall describe now. We will not implement it in this lab but interested readers can try it out.

- If, while implementing direct form I structure, the processing (of multiplication and addition) is performed on the previous inputs first and then on the previous outputs to calculate the current output, then it can be depicted in block-diagram format as shown in Figure 1.

Note: All block diagrams are drawn assuming that $a_0 = 1$, i.e., all coefficients are normalized by a_0 . But, during implementation if it is not, you will have to scale down the output by a_0 .

- Let's change the order of operations so that, the two blocks B_1 and B_2 exchange their positions. As a result, we get the block diagram in Figure 2.

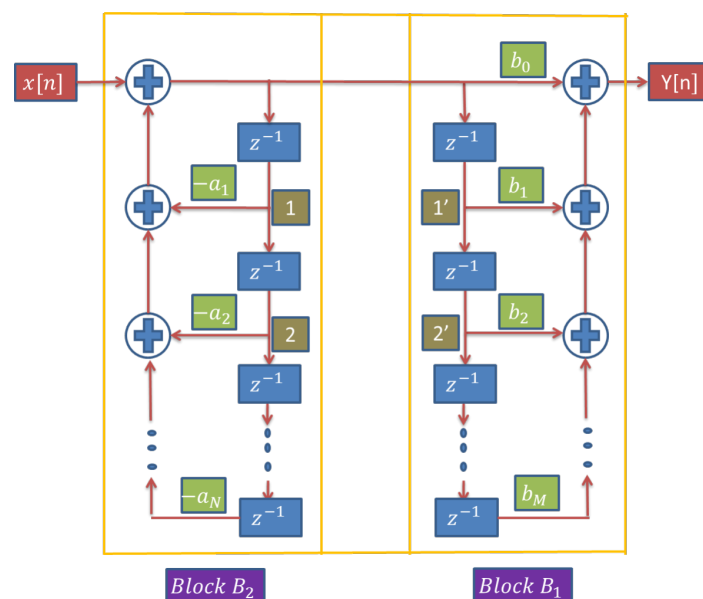


Figure 2: Block diagram obtained by exchanging the positions of blocks B_1 and B_2

- At this point, look at the block diagram in Fig. 2 and think on how you can reduce the total buffer space required to implement the same system. Take help from your TA/RA, if required. Once you have figured it out, read further.
- We observe that, the signals at the points 1 and 1', 2 and 2' and so on, are identical. So, rather than using separate delay elements (note that each delay element in the structure is equivalent to an extra memory space in the buffer) to generate these pairs of signals, we combine them to get a more efficient structure, in terms of memory usage, as is shown in the block diagram in Figure 3.

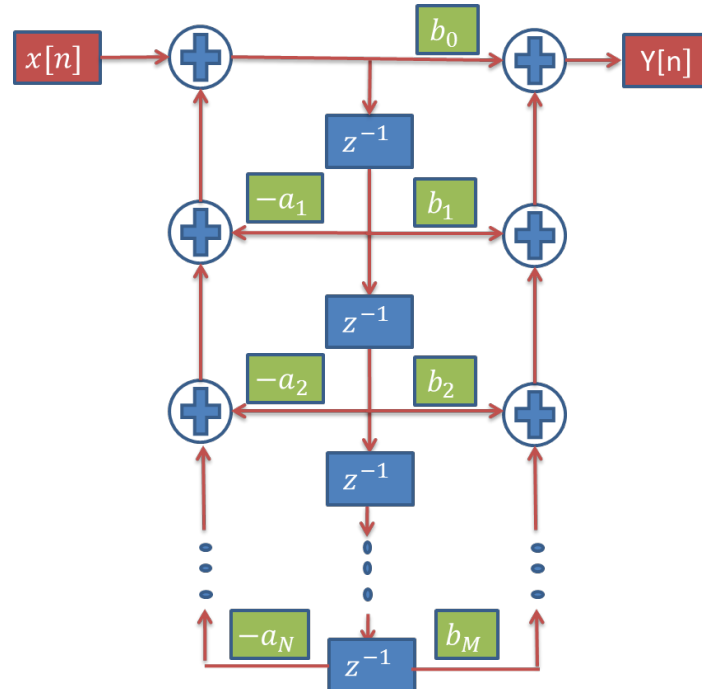


Figure 3: Block diagram for direct-form II structure for the IIR filter given by (3)

- This implementation is called the direct form II structure. Here, only one buffer has to be maintained, for the *intermediate signal*, which we shall denote by $w[n]$.

The algorithm for implementation of an IIR filter, defined by (3), on a DSP by direct-form II can be written as follows.

1. Allocate memory space to just one buffer of size $\max\{M + 1, N + 1\}$ (why?) and initialize it with zeros. Let's call this as *Intermediate signal buffer (ISB)*.
2. Start the acquisition of data.
3. Go on calculating the values of the intermediate signal according to (3) and storing them in the ISB, one after the other, until it saturates.
4. Feed the output display unit (the DSP codec combined with the DSO in our case) with every output sample, as it is calculated.
5. After the buffer runs out of memory space, start overwriting the oldest values with the latest values.
6. Continue steps 3 to 5 till the input data-flow continues.

4 IIR vs FIR filter implementations


One can observe that the desired frequency response for an LTI system can be obtained through both, FIR and IIR systems. Let us explore the major advantages and disadvantages of both the approaches through working examples in the form of checkpoints.

4.1 Learning checkpoint/exercise

All the following tasks need to be shown to your TA to get full credit for this lab session.


1. Go back to the project `c_iir` project used in checkpoint 2.1. Change the denominator coefficients (stored in array `den_coeff`) to $\{100, -160, 64\}$ in the `main.c` file. In other words, the filter that you are implementing now is,

$$H_2(z) = \frac{1 - 1.2728z^{-1} + 0.81z^{-2}}{1 - 1.6z^{-1} + 0.64z^{-2}}. \quad (5)$$

2. List down the poles and zeros of the system and get them verified from your TA/RA.
3. Repeat step 2 from checkpoint 2.1. Now, let us call the frequencies where the gains fall by 3dB and 20dB w.r.t. the pass-band as **Fpass** and **Fstop**, respectively. Note down **Fpass** and **Fstop** for $H_2(z)$.
4. Verify the frequency response on DSO. Follow the guidelines mentioned in step 3 of checkpoint 2.1.. 
5. Give an estimate of the number of computations required to calculate one sample of the output using the filter $H_2(z)$.
6. Finally, is the phase of the filter $H_2(z)$ linearly varying with the frequency **f**? Refer to the plot output by the function `freqz`.
7. Consider the situation where we are constrained to use only 1 digit after decimal point to specify all the coefficients. Now, this is equivalent to introducing round-off error while quantizing the coefficients. Let, $H_3(z)$ be the resulting transfer function. You can use the coefficients `num_coeff` = $\{10000, -12000, 8100\}$, `den_coeff` = $\{100, -160, 60\}$ in your IIR filter implementation code. Now, repeat steps 2 to 4 of checkpoint 4.1 for $H_3(z)$. Report the observations to your TA/RA. Is the change in the frequency response from that of $H_2(z)$ to that of $H_3(z)$ significant? Also, comment about the stability of $H_3(z)$.

4.2 Learning checkpoint/exercise

All the following tasks need to be shown to your TA to get full credit for this lab session.

1. Here we will obtain a FIR filter whose frequency response is similar to the IIR and understand effect of quantization. For this checkpoint, you will have to use the **FDatool** facility provided by Matlab. Type `fdatool` in the Matlab command window to open the same. It allows you to design an FIR or an IIR filter given its desired frequency response. Explore the GUI and try to understand the functionality of the tool. You may also take help from the `help` documentation in Matlab about the tool.
2. Now, considering the frequency response of the filter $H_2(z)$ as the desired one, design an FIR filter using **FDatool**. Use the **Fpass** and **Fstop** entries as the **Fpass** and **Fstop** values you calculated in step 3 in checkpoint 4.1. **Apass** should be 3dB whereas **Astop** should be 25dB. Click on the **Design Filter** button at the bottom. Click on **File** menu and then on **Export** sub-menu. Click on the **Export** button. You can see the designed FIR filter coefficients in the `Num` variable in the Matlab workspace. Note down the length of the resulting filter. Let's call this filter $H_4(z)$.
3. Estimate the number of computations required to generate 1 output sample using $H_4(z)$.
4. Does $H_4(z)$ have a linear phase? 
5. Now, we want to assess the performance of $H_4(z)$ in the scenario of step 7 of checkpoint 4.1. So, first plot the frequency response of this filter using the function `freqz`.
6. Next, normalize the first coefficient to 1 using the command `Num=Num*(Num(1)^(-1))`.
7. Finally, run the command `Num_single_digit=round((Num*10))/10`. The vector `Num_single_digit` contains the filter $H_4(z)$ with all coefficients reduced to the precision of 1 digit after the decimal point. Let's call this filter $H_5(z)$.
8. Plot the frequency response of $H_5(z)$. Is it much different from that of $H_4(z)$?