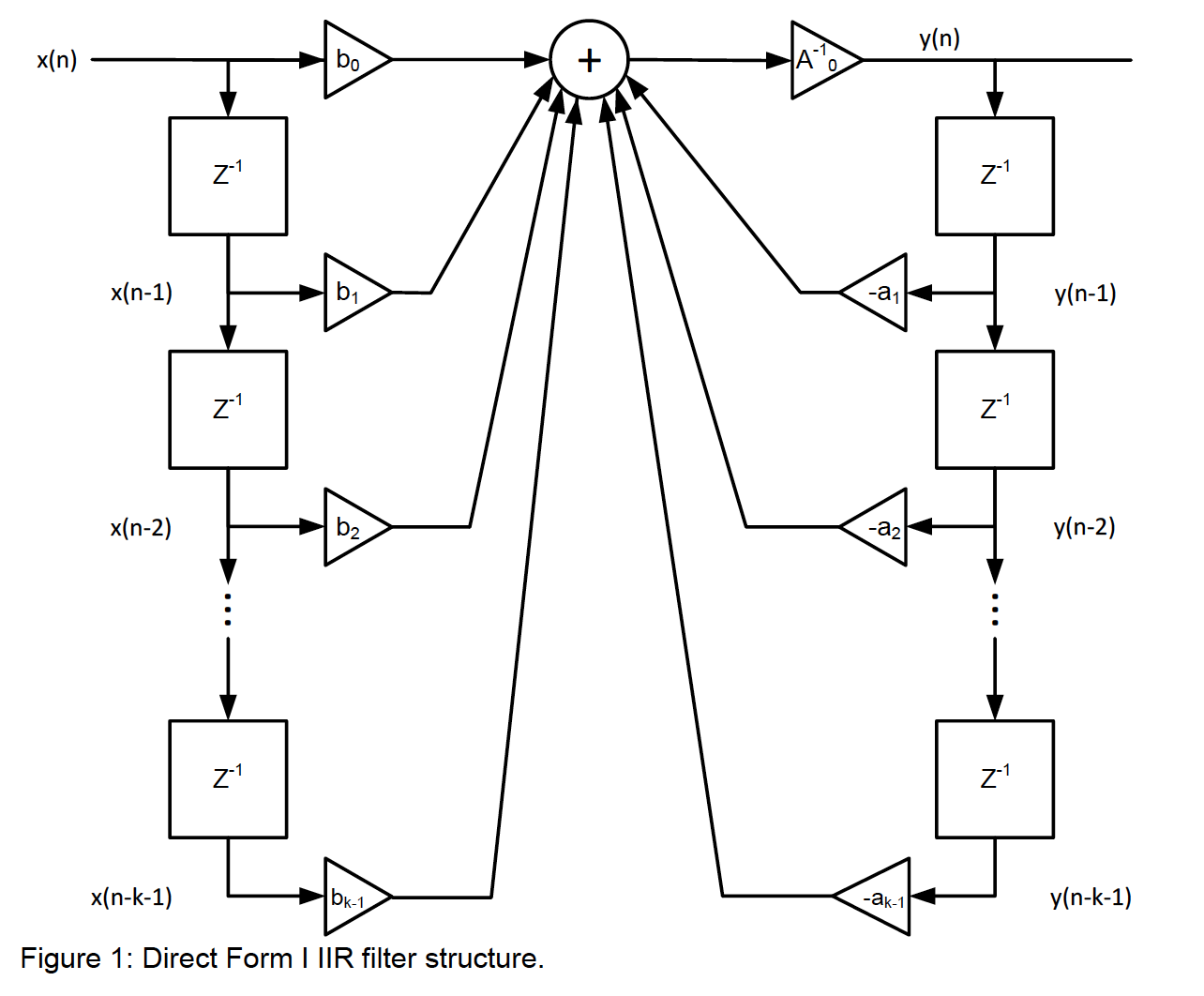
**Lab 3 and 4 IIR Filters**

**1. watch the video, link is as below:**

<https://ocw.mit.edu/resources/res-6-008-digital-signal-processing-spring-2011/video-lectures/lecture-12-network-structures-for-infinite-impulse-response-iir-systems/>

**2. Read sections 4.1-4.3 in the textbook before going any further.**

3. As with the FIR filter you have an input, **x** that yields an output, **y**. It is implementing the same structure as an FIR filter twice. Once with **x** as the input to the structure and secondly with **y** as the input to the structure. The following block diagram shows the IIR filter structure using inputs **x**, outputs **y** and delay elements. It is the standard Direct Form I diagram. Notice that the left half of the diagram is an FIR filter structure.

Input comes in from the left and feeds the delay line giving us as needed.

The right half side is the second FIR like filter structure. The input to the delay line is however the output, **y** of the filter and the coefficients are the s associated with the denominator portion of the transfer function. Note there are minus signs in front of the to coefficients because they need to be the negative of the coefficients generated by the MATLAB IIR filter design tools. You can negate the coefficients and add them to the sum of products or keep the coefficients as they are and subtract them from the sum of products. Make sure you do one of these corresponding to the code implementing the filter. If you don’t the filter will blow up because these coefficients are in the feedback path of the IIR filter.

That leads to the major difference between FIR and IIR filters. The FIR filter is very difficult to make unstable. The IIR filter on the other hand can be unstable just by having a sign wrong on one of the coefficients. This is because the IIR filter output feeds back into itself. In other word, it is recursive which gives it the quality of having an Infinite Impulse Response hence IIR.

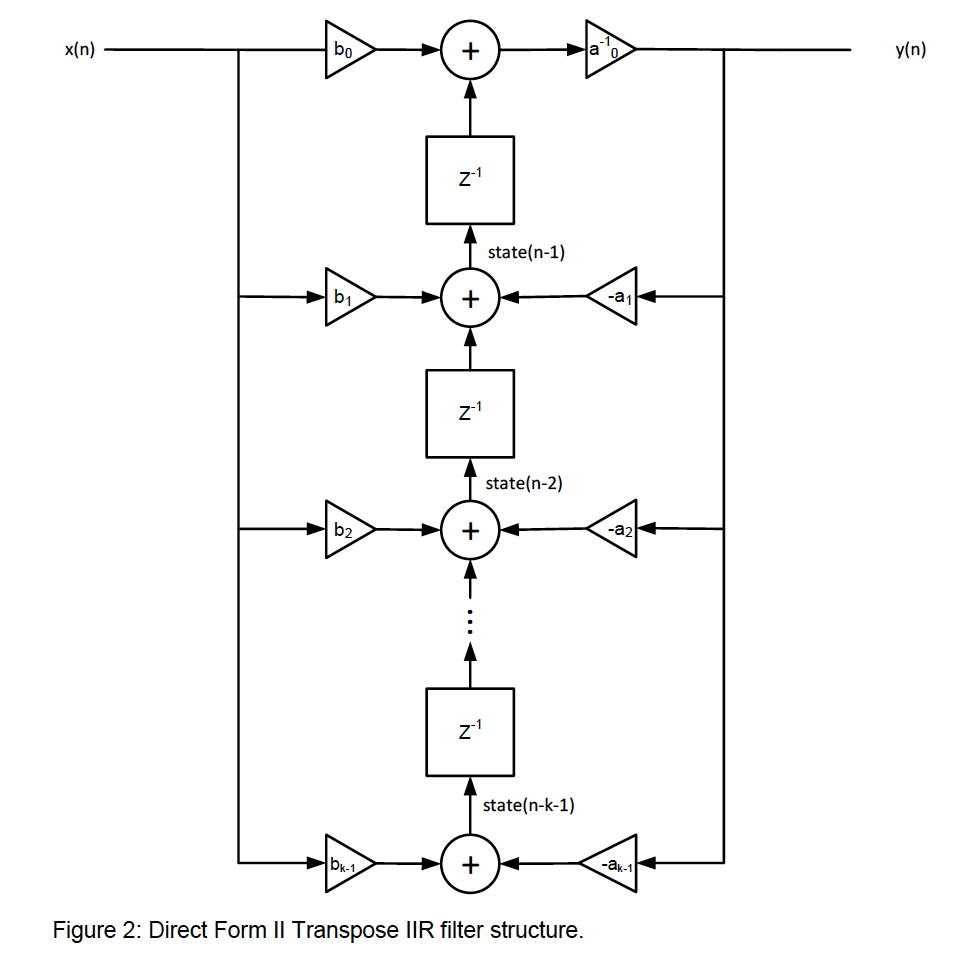
The reason that Direst Form I of the IIR filter is presented first is that it is fairly easy to see that this block diagram implements the difference equation for the IIR filter.

Re-written so the output is on one side depending on the inputs and delayed outputs:

Normally so:

Note MATLAB gives the coefficients normalized to . Also most implementations in C assume too. So equation **(1)** is most often used.

The classic forms of the IIR filter structure are Direct Form I, Direct Form I Transpose, Direct Form II and Direct Form II Transpose. All four Forms are equivalent. They all use the same coefficients and with an input and output that are the same. Direct Form I and Direct Form II Transpose are most often used in practice. **Figure 1** shows DFI and **Figure 2** shows DFIIT.



The reason why DFIIT is popular is the efficient use of memory. There is only one delay line used to implement this filter structure.

It is important to note that this filter structure is more susceptible to numeric precision than the DFI. This means that the filter will go unstable more easily and will have a noisier stop band as the precision of the states gets lower.

The advantage of the DFI is it is less susceptible to numeric precision at the cost of

using more memory.

This is a simple 2nd order IIR DFI filter.

* It should run as written.
* Use a musical input.
* Listen to the filter output.
* Add a talk thru on the other channel and listen again.

**You don’t have to turn in anything for this part 3.**

**4. Generate coefficient files for a 4th order DFIIT IIR filter (.c and .h file) in MATLAB**

Parameters:

* Response Type: Low pass
* Design method: IIR Butterworth
* Filter Order: 4th
* Frequency specifications: Sample frequency: 48kHz Cutoff frequency: 750Hz

(1) Use command: fdatool

a. Launch the fdatool in MATLAB command, and correct all the parameters, then Design Filter

b. Go to Edit-Convert to Single Section

c. Go to Edit-Convert Structure, choose Direct Form II Transpose, click OK

d. Go to File-Export, Export to a workspace, Export as coefficients, Set the Numerator name as Num, Set Denominator name as Den, click Export. Then you get two workspaces as Num and Den.

e. Go to Targets-Generate C Header: Set NUM as the Numerator, Set NL as the Numerator length, Set DEN as the Denominator, Set DL as the Denominator length, then Generate the .h flie

**\* This .h file contains the contents of both the regular .c and regular .h files. This fdacoef.h file is not what you are going to use in your CCS project. You have to split it into your own .c and .h files.**

(2) Use the IIR\_dump2C.m (Path: Book3rdEdition/code/chaptor04/matlab/matlabExport/ IIR\_dump2C.M, please change .M to .m in this folder)

a. Use two workspaces you just created.

b. Run this IIR\_dump2C.m to generate your own .c and .h file

Remember the correct signs on a1, a2, a3 and a4 are critical. Figure out if MATLAB assumes the terms are added or subtracted. Make sure the signs in your C code are correct.

Note there is usually an exponent associated with the B coefficients. In MATLAB execute the format long command to display the coefficients with double floating point accuracy if needed.

(3) Create a new CCS project. The example ISR code and startup.c you should start with is in the folder of chapter\_04 in the Book3rdEdition. Use the code in the IIRrevA folder to start with.

(4) Add all necessary files to your CCS project with your coefficient files. Build the project, and run it into the board, see if your filter is working.

**\* Turn in the whole CCS project you created for implementing the 4th order DFIIT IIR filter.**

**5. Create an 8th order DFII SOS Butterworth IIR Low-pass Filter**

(1) Generate your .c and .h file

a. As mentioned in Part 4, launch the fdatool, design the Filter.

Parameters are:

Response Type: Low-pass

Design Method: IIR Butterworth

Filter Order: 8th

Sample Frequency: 48000Hz

Cutoff Frequency: 750Hz

Then design the filter, and go to Edit-Convert Structure, and choose DFII SOS

Next, Go to File-Export, Export to the workspace, Export as coefficients, Set the SOS Matrix as SOS, Set the Scale Values as G. Then Export.

b. Go to the course Moodle website, download the MATLAB code: SOS2C.m, and copy this MATLAB code to your own folder.

Open this code, go to the MATLAB command and run this SOS2C.m function. It will generate the .c and .h files for you.

(2) Create your CCS project

Please follow the instruction of how to create a new CCS project, add the correct files are needed in your project. Besides the general files, you have to add

a. Your own .c and .h file

b. IIRmono\_ISRs.c and StartUp.c from Book3rdEdition/code/chaptor04/ccs/IIRrevA

Don’t forget to choose eabi(EFL) as your output format.

**Let the left channel go through the filter and leave the right channel talk through.**

**\* Turn in the whole project for the 8th order DFII SOS Butterworth IIR Low-pass Filter.**

**6. Compare the 4th order DFIIT to the 8th order DFII using music as a source to filter.**

(1) Put the 8th order DFII IIR LPF on the left channel and the 4th order DFIIT IIR LPF on the right channel

**\* Turn in the whole project.**

(2) Comment on the performance of these filters.

* Discuss the advantages of using the DFIIT filter structure.
* Discuss the advantages of using the SOS filter structure for higher order filters. Why didn’t we use SOS for the FIR filters?

**＊Turn in the two answers to these questions in a text or a word or a PDF file.**

**\* Please include:**

**a. Whole CCS project for part 4**

**b. Whole CCS project for part 5**

**c. Whole CCS project for part 6**

**d. Your answers to the questions in Part 6 in a txt/word/PDF file**

**Please Zip all of them up and turn in the zip file for the Assignment. Put both of your members’ names as the name of your zip file.**