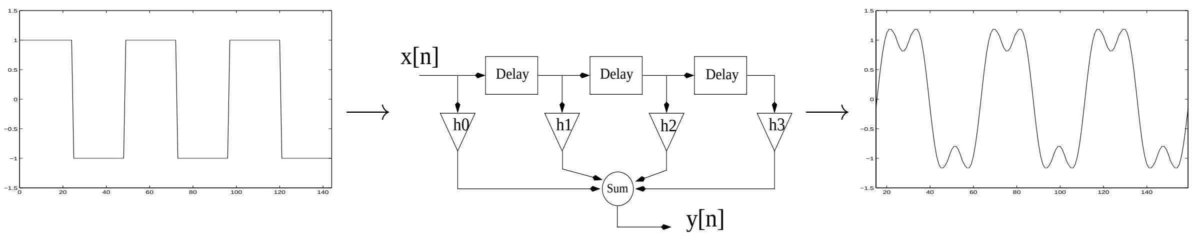
Lab # 4

Digital Filters



1 Purpose

Digital Filters are among the most common DSP applications, being found in a large variety of embedded systems. The purpose of this lab is to provide you with the basic understanding of the design and implementation of digital filters. You will have a chance to design, simulate and implement a finite impulse response (FIR) filter. The design of your digital filter will be accomplished with the use of the `Filter Design and Analysis Tool`, or `FDA Tool`, provided by `MATLAB`. The simulation will be done with `Simulink`, and the implementation will be done in C within `Code Composer Studio`.

After this lab, you will be able to do this:

- The design of a digital filter with the assistance of a software tool;
- The testing, through simulation, of this filter;
- The implementation of a digital filter on a DSP platform.

2 Reviewing the Theory

The theoretical background needed for this lab is well documented, and it can be found in both basic and advanced texts in signals and systems and Digital Signal Processing. The course textbook ([1]) provides a solid theoretical review. For practical implementations, you can look at [3] and [4], among many others.

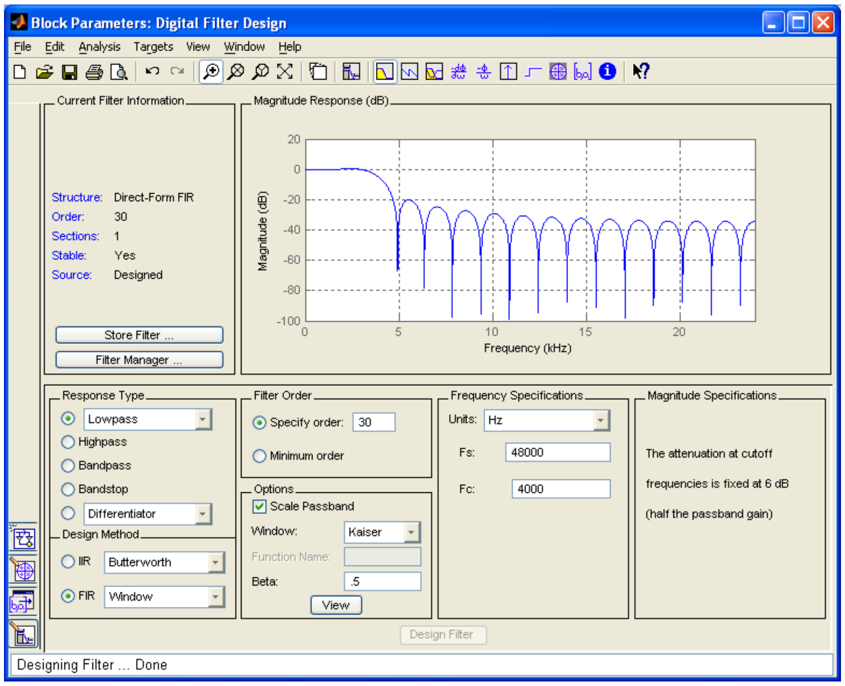
The digital filtering process can be described, in a very simplified manner, as the result of a convolution between the input (sampled) signal and the coefficients (also called "parameters," or "taps") of the digital filter. These coefficients are real-valued parameters of the polynomial which describes the filter in the z domain. By "polynomial" we mean "polynomial in the variable z^{-1}." If the topology of such filter has feedback, its transfer function will have polynomials for both numerator and denominator. When no feedback is present, the denominator will equal 1, and the coefficients of the numerator polynomial will represent the impulse response of the filter. This no-feedback topology describes a *finite impulse response* (FIR) filter. The one with feedback is called *infinite impulse response* (IIR) filter.

3 Lab

This lab encompasses the design, simulation and implementation in realtime of an FIR filter. It should take two hours, if you work efficiently.

3.1 FIR: Design

There are numerous software packages commercially available that will provide you with the necessary tools to design digital filters, both FIR and IIR. In this lab, you will design your filters using the Filter Design and Analysis Tool from `MATLAB`. The most straightforward way to start the tool is to type `fdatool` on the command line in the main `MATLAB` window. You can alternatively open a blank model from `Simulink` and drag in the `FDA Tool` block. When you double-click on the `FDA Tool` block, the GUI presented in Figure 1 will open. This block will then invoke `MATLAB` to generate the desired filter coefficients according to your specifications.



Structure: Direct-Form FIR
Order: 30
Sections: 1
Stable: Yes
Source: Designed

Response Type: Lowpass
Design Method: FIR Window
Specify order: 30
Scale Passband ✓
Window: Kaiser
Beta: .5
Units: Hz
Fs: 48000
Fc: 4000

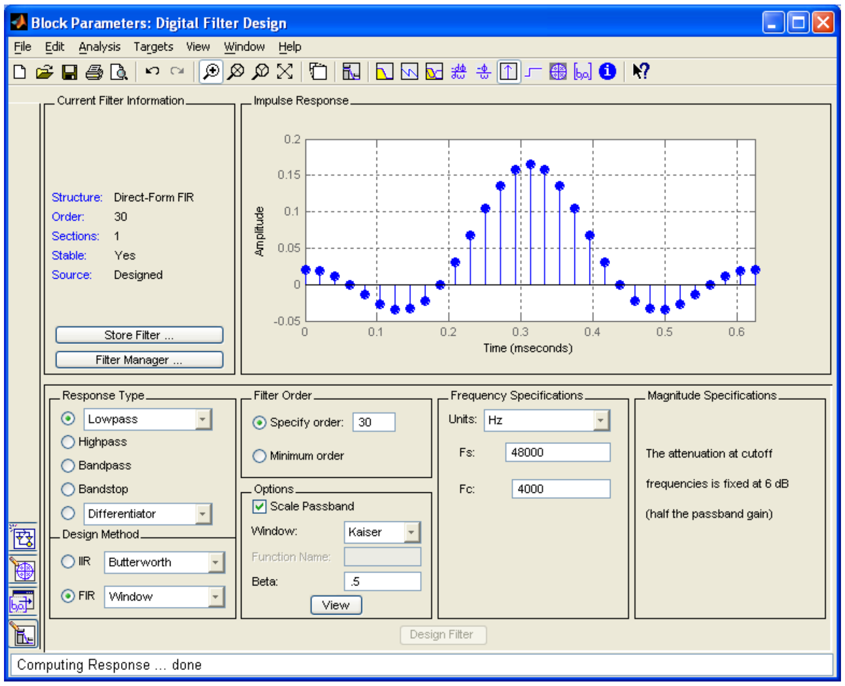
The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

Designing Filter ... Done

Figure 1: `FDA Tool` Graphic User Interface - Frequency Response Mode

The user interface provides you with options and spaces to set all parameters relative to the filter you want to design. In general, the most utilized display modes are the **frequency response display** (as in the picture above) and the **impulse response display**, shown in Figure 2.

You should be familiar at this point with moving back and forth between the time domain and frequency domain, particularly relating to filters. On the `FDA Tool` the impulse response plot is obtained by clicking on the upward-pointing arrow from the top menu.



Structure: Direct-Form FIR
Order: 30
Sections: 1
Stable: Yes
Source: Designed

[✓ Lowpass]
Highpass
Bandpass
Bandstop
Differentiator

IIR Butterworth
[✓ FIR] Window

[✓ Specify order:] 30
Minimum order

[✓ Scale Passband]
Window: Kaiser
Function Name:
Beta: .5

Units: Hz
Fs: 48000
Fc: 4000

The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

Computing Response ... done

Figure 2: `FDA Tool` Graphic User Interface - Impulse Response Mode

A block diagram example of an FIR topology could be obtained from the `FDA Tool` by clicking on the "realize" button on the bottom left corner of the tool. A new window should appear with a direct form implementation of your filter (reference Section 6.5.1 in the text), utilizing unit delay blocks, gains and adders. If you look into the gain blocks you will notice that the values there are the ones displayed in the coefficients display of your FDA Tool. The coefficients display can be obtained by clicking on the icon labelled **[b,a]**, located next to the **i** on the top button menu. This notation for coefficients is widely adopted, not only by `MATLAB`: usually `b` is the vector holding the numerator coefficients, whereas `a` is the vector holding the denominator coefficients of the filter transfer function in z.

Now you will explore the `FDA Tool` by trying different parameters and different design methods. The hardware you will utilize in the lab samples analog signals at 48kHz. This is your F_s. In this section you will use 6kHz as your cutoff frequency, or F_c. The initial [1] order of your filter is 20. Also, select `Low Pass` as a `Response Type` and `FIR: Window` as a `Design Method`. The idea here is to give you a relative picture between different windows used to design lowpass FIR filters. When creating the tables, feel free to use the zoom controls provided by the `FDA Tool`, or you will have

[1] it will change later

to estimate the values. Answer the question found on Section 1 of the answer booklet.

3.2 FIR: Simulation

From now on, keep in mind that you should be simulating a system which operates at a sampling frequency of 48kHz (sampling period of 1/48000 seconds), as dictated by the DSP hardware platform that will be utilized in the implementation part. In order to answer the questions, you can create a model using the `FDA Tool` block, by connecting an input (a sine wave generator) and outputs (time domain and frequency domain scope) to it. At this point you should be able to create your frequency domain scope. This model is shown on Figure 3.

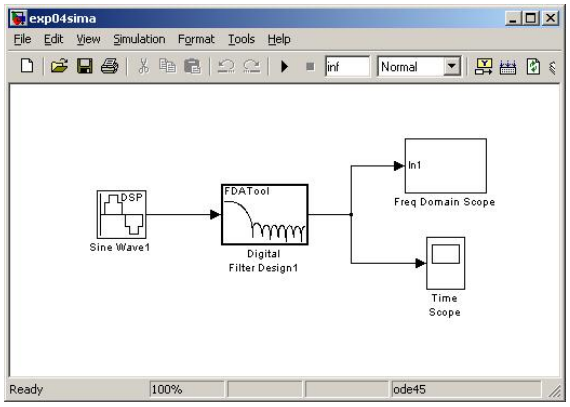
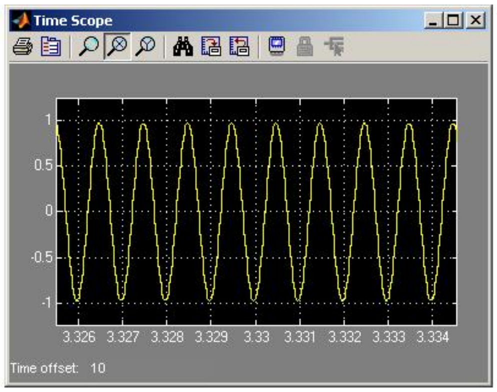
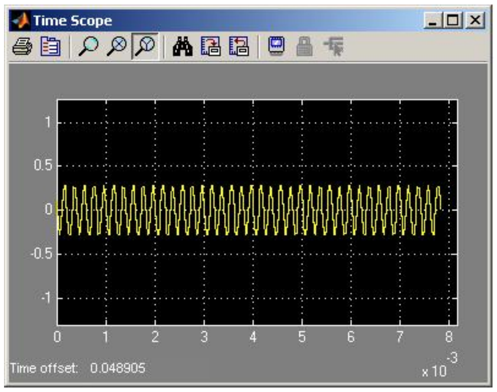


Figure 3: Simulation Model with `FDA Tool`

You will need to run this model in order to answer the questions below. Design an order 20 FIR filter using a Hanning window, with a 4kHz cutoff frequency. At this frequency, the output amplitude should be half of the input. Just to ensure things are running as they should, you can try the system for two input frequencies: 1kHz and 5kHz. You should expect to see what is displayed in Figure 4(a) and Figure 4(b).



(a) Time Domain, 1kHz



(b) Time Domain, 5kHz

Figure 4: Simulation Of A lowpass Filter

Answer the questions on Section 2 of the answer booklet.

3.3 FIR: Implementation

Now you will run an FIR filter of order 20, designed with a Hamming window for a 4kHz cutoff frequency. As an engineer, you are to choose the filter compatible with the needs of a project, and select the parameters according to the resources available to you. This is to say that if you can only spare so many processor cycles and you must add a filter to your project, you should stay away from long filters, or even from FIR filters. On the other hand, you may have to watch out for non-linear phase or instability if you decide to implement a lower order IIR instead. In this section you will run and test a filter which is designed with the `FDA Tool` and coded in C.

A **Direct Form** implementation of an FIR filter holds its impulse response in an array of coefficients (gains in a block diagram) and performs a weighted sum with the incoming samples of the input signal. Every time a sample (or a frame of samples) is ready to be delivered for processing, an interrupt is generated. This interrupt will cause the execution of the program to be directed to the interrupt service routine, in which the convolution is implemented. A sample of the code in C is shown below, and it is intended for a floating-point device. The filter coefficients are named in this sample code $h(0), \ldots, h(4)$. In reality they will be real numbers.

```
#define N  5 /* length of the filter */

/* filter taps: order = #taps - 1 */
float taps[N] = {h(0), h(1), h(2), h(3), h(4)};
short yn=0;
float delay_line[N]

interrupt int_12()
{
  short i;
  delay_line[0]= input_sample();  /* first sample at top of delay line */
  yn=0;
  for(i=0; i<N; i++)
     yn += (taps[i] * delay_line[i]);  /* WHAT IS THIS LOOP DOING? */

  for (i=N-1; i>0; i--)
    delay_line[i]=delay_line[i-1]; /* shift data in delay line by one */

  output_sample(yn);   /* put filtered result out */
  return;
}
```

It is worthwhile mentioning that the code shown above is a straightforward implementation of a convolution, and it is highly inefficient from a DSP programmer's point of view. A more efficient implementation would make use of circular buffers, reutilizing the buffers rather than performing shifting operations. Also, assembly language should be used to take advantage of other processor features such as pipelining. For this lab, it suffices that you verify the implementation to perform as well as the simulation. Nonetheless, the code you will run makes use of pointers and circular buffers, which avoid the shifts shown above. For more efficient pieces of code [2], refer to [3] and [4].

Load the project named `c:/ECE431/Exp04/Exp04.pjt`. Inspect the files in the project and try to identify where are the main parts, such as: the management of input and output samples, the interrupt vector and interrupt service routines, the memory map, the placement of FIR filter coefficients and the routine implementing a convolution. Note that the program given to you performs the filtering on both channels, i.e., every incoming 32-bit sample from the CODEC is split into two 16-bit samples (it is a stereo codec), then each of these is passed through the FIR filter.

Your task is to design a lowpass filter of order 20, cutff frequency of 4kHz, using a Hamming window as your design method (use the `FDA Tool!`). You will need to export your coefficients into a coefficient file (in ASCII), by going under `File/Export` on the tool and selecting ASCII. Alternatively, you can generate an `m file` and run it within `MATLAB`. After you do that, you will copy the coefficients into the proper place in the project (you find out where it is), make the proper syntax arrangements, compile and run your filter. If you have no errors and for some reason it does not run the first time, just load it again and run it.

You will test your filter with a $2V_{pp}$ sinusoid as input to both channels of your DSP platform. Make sure the output enabling button is set on your signal generator. Answer the final questions on your answer booklet.

4 Accomplishments

In this lab, you have designed, simulated and implemented an FIR filter. You have explored a digital filter design tool and used it to generate coefficients for a digital filter running in real-time on a DSP platform. You have also explored the C code used to run your filter, which was implemented using a straightforward convolution. If you are willing to further explore digital filters, the next step is to investigate more computationally efficient methods to realize digital filters in real-time.

[2] as well as methods to test their efficiency and limitations

Answer Booklet
Lab # 4 - Digital Filters

• *Name:*	*Lab Date:*	
• *Student No.:*	*Day of the week:*	*Time:*
• *Name:*		
• *Student No.:*	*Grade:*	*/ 10*

1 FIR: Design

1. *Having set all parameters, click "view" to select a normalized view, then create a table to compare the amplitude of the first sidelobe for the following windows: Rectangular, Hamming, Hanning and Blackman. Compare your table with the one found on page 471 (Table 7.1) in the textbook.*

2 FIR: Simulation

1. *With the model running, change the input frequency and draw the filter frequency response from the output values. In order to do that, you can set the running time to* `inf`*, as it is in Figure 3. Use 500Hz, 1KHz, 2KHz, 3KHz, 4KHz, 5KHz and 10KHz.*

2. *Change the order to 40 (i.e., redesign the filter) and repeat the procedure above for 500Hz, 1KHz, 2KHz, 3KHz, 4KHz, 5KHz and 10KHz.*

3. *Compare the results from the two steps above, and explain what changes in the response. At what expense would you opt towards a higher order filter?*

3 FIR: Implementation

1. *With the filter running on the DSP, change the input frequency and draw the frequency response magnitude. Use 500Hz, 1KHz, 2KHz, 3KHz, 4KHz, 5KHz and 10KHz. Compare these results with the ones obtained in the simulation.*

2. *Identify where the first sidelobe is by varying the frequency of your input signal. Show it to the TA.*

TA Check:

References

[1] A. V. Oppenheim, R. W. Schaefer with J. R. Buck, 2nd Ed. *Discrete-Time Signal Processing*, 2nd Ed., Prentice Hall, 1999

[2] E. Ifeachor and B. Jervis, *Digital Signal Processing - A Practical Approach*, Addison Wesley

[3] R. Chassaing, *Digital Signal Processing and Applications with the c6713 and c6416 DSK*, Wiley, 2004

[4] T. B. Welch, C. H. G. Wright and M. G. Morrow, *Real-Time Digital Signal Processing - From MATLAB to C with the TMS320C6X DSK*, CRC Press, Taylor & Francis, 2006