Lab 3 Part 1: FIR Filter Design

1 Overview

1.1 Introduction

The FIR Filter Design Lab is split into two lab sessions:

I. In the first session, we will focus on the design of FIR filters in Matlab, understand how noise can be removed using a designed filter and examine the effects of quantisation on filter performance.

II. Next week, in the second part of the FIR Filter lab, we will implement our designed filters on hardware using the PYNQ-Z2 board.

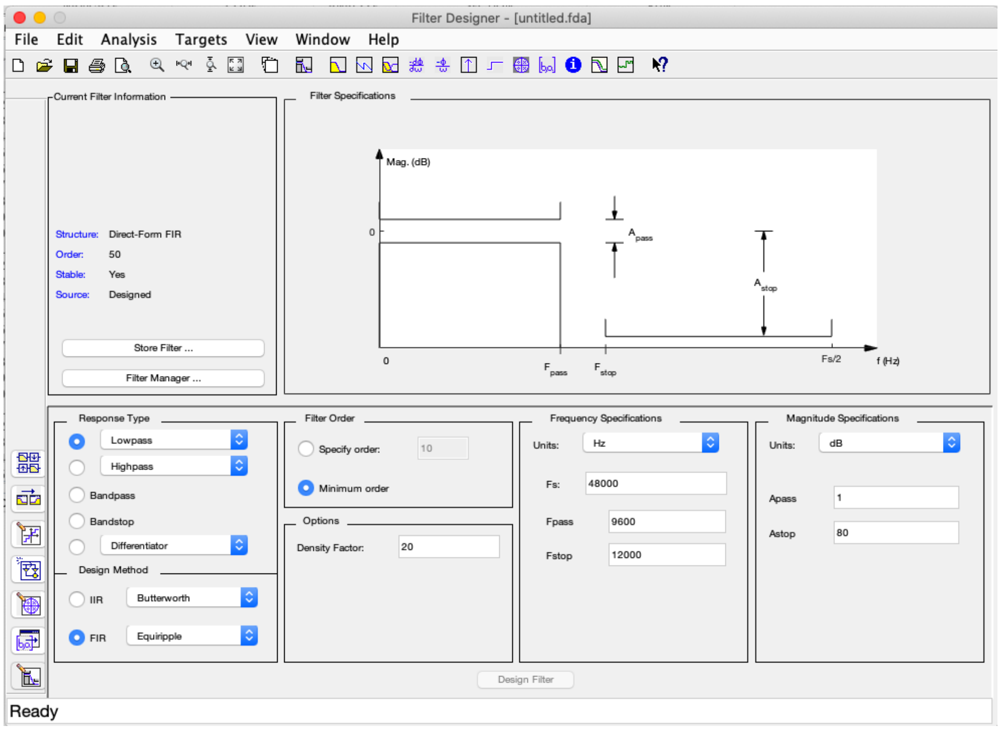
1.2 Learning Outcomes

On completing this lab, you will be able to:

- Become familiar with using the filterDesigner (previously fdatool) tool in Matlab for designing FIR filters
- Obtain filter coefficients for different FIR designs specifications
- Analyse a corrupted audio file and design a suitable filter to remove the noise
- Analyse the effects of quantising the filter coefficients

2 Using Matlab's Filter Designer Tool

The Matlab filterDesigner is used for designing and analysing filters. To open the GUI, type *filterDesigner* into the command window. Figure 1 shows a screenshot of the interface that will appear on your screen. To design a filter, you can enter the desired specifications into the GUI and click the *design filter* button to implement it. The magnitude response of the filter will then be displayed in the window.



Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: Lowpass
Design Method: FIR Equiripple
Filter Order: Minimum order
Density Factor: 20

Units: Hz
Fs: 48000
Fpass: 9600
Fstop: 12000

Units: dB
Apass: 1
Astop: 80

Figure 1: The Filter Designer GUI

- **Storing Filters:** You can store filters by clicking the *store filter* button and access stored filters using the *filter manager* button.

- **Generating Matlab Code:** The filter can also be exported as Matlab code by clicking File → Generate Matlab Code.

3 Filter Design Exercise

3.1 Problem Description

Design a minimum order lowpass FIR filter using the filterDesigner tool to meet the following specifications:

- Stopband attenuation > 90dB
- Passband ripple < 0.02 dB
- Passband edge frequency 3.375 kHz
- Stopband edge frequency 5.625 kHz
- Sampling frequency 20 kHz

3.2 Lab Report Questions

I. What is the order of the filter you designed? What does this mean?
II. What are the effects of altering the passband and stopband attenuation?

Include screenshots of the magnitude response of your filter design to support your answers where appropriate.

4 Filtering a Noisy Audio File

4.1 Problem Description

You will now design an FIR filter to filter out noise from a speech .wav file. You should be able to hear the noise when you listen to the sample (if not, please contact me). Either use the filterDesign matlab tool or write a script to design an appropriate filter.

Everyone has been assigned a speech file with a slightly different tone. Do not assume that the quality of the filtered speech file for everyone will be the same. It will depend on exactly where your interfering tone is. Some people's filter output will sound worse than others; you will be graded on your demonstrated understanding and application of filter design principles. You will need to decide what is reasonable for **your** speech file.

Please refer to the Appendix to see your assigned audio file.

4.2 Instructions

I. **Identify the Noise:** First, you need to identify the frequency range you want to filter out. The Discrete Fourier Transform (DFT) allows you to examine the frequency content of the wavefile. The fft() function in Matlab is a fast implementation of the DFT. A pure tone has been added to each file at a different frequency which you will need to remove. You should read in and analyse the frequency content of the file in Matlab.

   Sample Code:

```
[x,Fs] = audioread('speech_0.wav'); % Read in audio file

% Plot to find out which frequency to remove from signal
nfft = 2^10;
X = fft(x, nfft);
fstep = Fs/nfft;
fvec = fstep*(0: nfft/2-1);
```

```
fresp = 2*abs(X(1:nfft/2));
plot(fvec,fresp)
title('Single-Sided Amplitude Spectrum of x(t)')
xlabel('Frequency (Hz)')
ylabel('|X(f)|')
```

This code will generate a figure showing the frequency content of the speech file. There should be a strong peak at a certain frequency. This is the noisy pure tone that was added. Write Matlab code to automatically capture the frequency of the tone and record this plot for your write-up (hint: max and argmax functions). Include this code in your Matlab script for submission with each line commented to demonstrate you understand its purpose.

II. **Design the FIR:** Use the filterDesigner tool in Matlab to design an FIR filter to remove the noise. You will need to define your passband(s) and stopband(s) etc. appropriately. Outline your decision process in your write-up.

Pass your audio file through the filter as demonstrated in class, then listen to the output. Is the noise gone? How clear or intelligible is the speech? If it is not clear, is there a problem with your filter?

Redo the frequency analysis on the filtered signal to verify this. Record the filter magnitude response for your write-up.

III. **Quantise the FIR Filter:**

Once you are satisfied that the filter operates correctly, experiment with quantising the coefficients. Compare the magnitude response for full precision and different levels of coefficient quantisation. What do you notice? Listen to the output. What do you notice?

At this point you may wish to reassess the order of the filter you originally designed, to account for quantisation. Discuss the trade-offs considered with your parameter choices.

Record the magnitude response at 3 different levels of quantisation for your write-up, these levels should be 'under-quantised', 'appropriately quantised' and 'over-quantised'. Indicate the final number of bits you will use for the filter coefficients. Include the steps taken to produce these in your submitted Matlab code.

4 Submission

Please submit a **brief** lab report containing:

- The lab report questions and screen captures from the first filter design exercise (refer to Part 3).
- Details of your decision process to design the noise-removing FIR filter.
- Graphs of the frequency response before and after the audio is filtered.
- Your method for quantising the filter coefficients and graphs of the frequency response after applying the 'under-quantised', 'appropriately quantised' and 'over-quantised' filters.
- Discuss the trade-offs you considered for selecting the number of bits for your final filter.

Please submit the following in a zipped folder using your name and lab3_p1 as the file name (e.g. AWalsh_lab3_p1):

- Your **brief** lab report titled in the format AWalsh_lab3_p1.pdf
- Your Matlab code used to generate filter coefficients, quantise them, estimate hardware cost, generate frequency responses and filter the data.
- Attach your filtered speech file (.wav) filtered with the full-precision FIR.
- Attach your filtered speech file (.wav) filtered with the final/chosen quantised FIR.

5 Appendix: Audio File Assignment

If your name does not appear in the table below please contact me to be assigned a speech file.

Student Name:		Assigned speech file
		speech_1.wav
		speech_2.wav
		speech_3.wav
		speech_4.wav
		speech_5.wav
		speech_6.wav
		speech_7.wav
		speech_8.wav

		speech_9.wav
		speech_10.wav
		speech_11.wav
		speech_12.wav
		speech_13.wav
		speech_14.wav
		speech_15.wav
		speech_16.wav
		speech_17.wav
		speech_18.wav
		speech_19.wav
		speech_20.wav
		speech_21.wav
		speech_22.wav
		speech_23.wav
		speech_24.wav
		speech_25.wav

Lab 3 Part 2: FIR Filter Hardware Implementation

1 Overview

1.1 Introduction

Last week we learned how to design FIR filters using the filterDesigner tool in Matlab. In this lab we will now implement an FIR filter on hardware using the PYNQ-Z2 boards.

Why do we want to implement the filter on hardware? Software implementations use a computer's memory, ALU (arithmetic logic unit) and controller and the resulting implementation is slow and not suitable for real-time processing. Hardware implementations using FPGA, however, can achieve faster processing speeds due to hardware parallelism and pipelining.

Recall that PYNQ is an open-source project from Xilinx which integrates software and hardware components for faster development using Zynq devices. PYNQ combines the Python language with FPGA-based Programmable Logic (PL) and an Arm-based Processing System (PS) for building electronic systems.

PYNQ uses Jupyter Notebooks as an interactive environment for writing code and running it on the target PYNQ board. The Notebook server runs on the ARM® processor of the board. In this lab, we will make use of Jupyter Notebooks for running our designed FIR filters on the PYNQ-Z2 boards.

1.2 Learning Outcomes

On completing this lab, you will be able to:

- Obtain the filter coefficients from a designed FIR filter.
- Edit an overlay's FIR filter IP provide the desired filtering functionality.
- Load an overlay onto the PYNQ-Z2 board and assess its performance.
- Compare the effects of different filters on the audio output.
- Understand the trade-offs of implementing audio processing on hardware vs software.

1.3 Definitions

- Overlay: Overlays are hardware system designs on PYNQ. It's a configurable and reusable class of Programmable Logic Design and can be downloaded into the Programmable

Logic at runtime to provide functionality required by the software application. The PYNQ overlay has a Python interface, meaning it can be used like a Python package.

2 Design the Filter

I. **Design:** You have been provided with an audio file (audio.wav). Design a FIR filter for this audio file following the steps from last week's lab.

II. **Export Coefficients:** When you have designed your filter, you will need to export the coefficients as a .COE file. To do this, first generate the a Matlab file from the filter design within the filterDesigner tool (File → Generate MATLAB Code → Filter Design Function). Then use the the *coewrite* function on the filter output:

```
Hd.arithmatic = 'fixed', % Requires Fixed Point Designer
Coewrite(Hd)
```

3 Edit the PYNQ Overlay

You have been provided with an overlay for the PYNQ-Z2 as a zipped Vivado project. Extract the project to your Local Drive (C:).

3.1 Open the PYNQ Overlay

To open the project you will need to run some Tcl commands. Open the Vivado Application and go to the Tcl Console at the bottom of the GUI.

I. Navigate to the directory where you saved the project (hint: cd command for changing directory and pwd to print current directory). Then cd to PYNQ-image_v2.4/boards/Pynq-Z2/base/

II. Build the IP files by running source *build_base_ip.tcl*

III. Open the project by running *base.tcl*

This will open the Vivado project and you will be able to see the overlay as shown in figure 1.

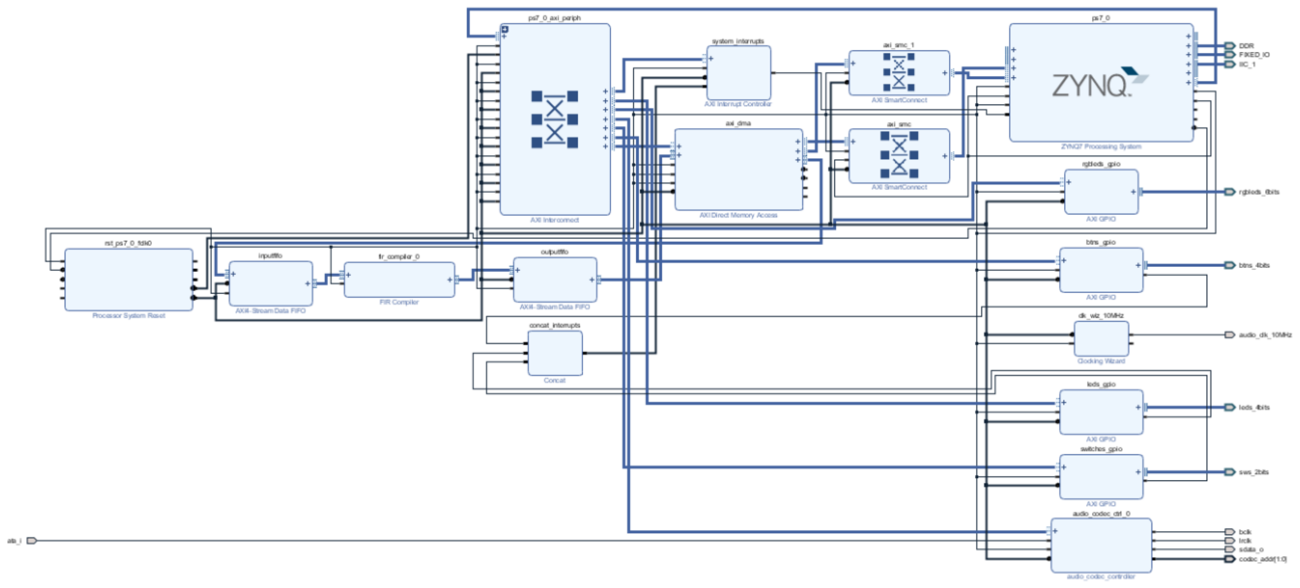


Figure 1: PYNQ FIR Filter Overlay

3.2 Edit the FIR Compiler IP

Next you will need to edit the FIR filter IP to provide the functionality of your filter design.

I. Locate the FIR Compiler IP in the overlay and double click to customise it.

II. Under the Filter Options tab, select COE file in the drop down menu for the coefficient source.

III. Browse to select your COE file of the FIR filter you designed in Matlab.

IV. Under the Channel Specification tab you will need to edit the Hardware Oversampling Specification section:

   a. Decide the appropriate sampling frequency for the audio file.
   b. Set the clock frequency to 100 MHz which is the clock frequency of the PYNQ-Z2 board.

V. Under the Implementation tab:

   a. Configure the Coefficient Options section to set the coefficient type as signed, integer coefficients with a 16 bit width.
   b. Set the input data as signed 32 bit with 0 fractional bits.
   c. Set the output as 32 bit and select the Non Symmetric Rounding Up rounding mode.

VI. You can view an overview of your designed filter under the Summary tab. Select ok to finish customising the filter.

3.3 Export the .bit and .hwh Files

Next we need to export the .bit and .hwh files so we can load the overlay onto the PYNQ-Z2. The .bit file instantiates the overlay class, which loads PYNQ overlays to the Programmable Logic (PL). The .hwh file is parsed and the bitstream is downloaded to the PL.

I. Export the hardware by clicking File → Export →Export Hardware and tick the box to include bitstream.
II. Run design Synthesis and Implementation.
III. Close the project the type source *build_bitstream.tcl* in the Vivado Tcl Console to build the project's bitstream. Note: Make sure you cd back to the correct directory first.
IV. You will find the .bit and .hwh files in the PYNQ-image_v2.4/boards/Pynq-Z2/base/ directory.

4 Run the Filter on the PYNQ-Z2

You will need to insert a micro-SD card with the PYNQ-Z2 image into your board and connect to the local network in order to access Jupyter Notebooks and load the overlay to the board.

4.1 Setting up the PYNQ-Z2

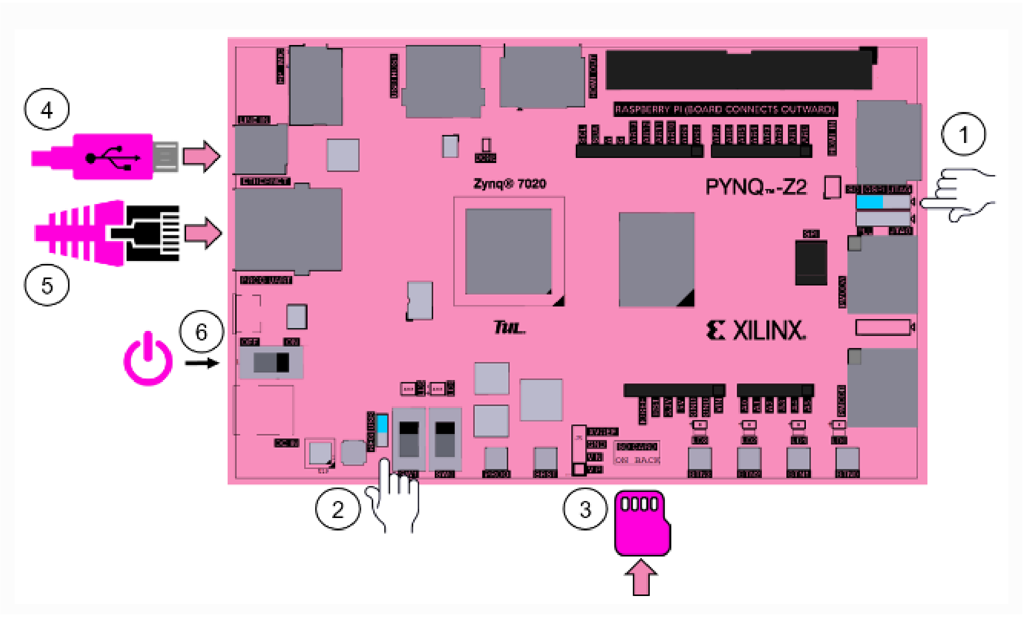
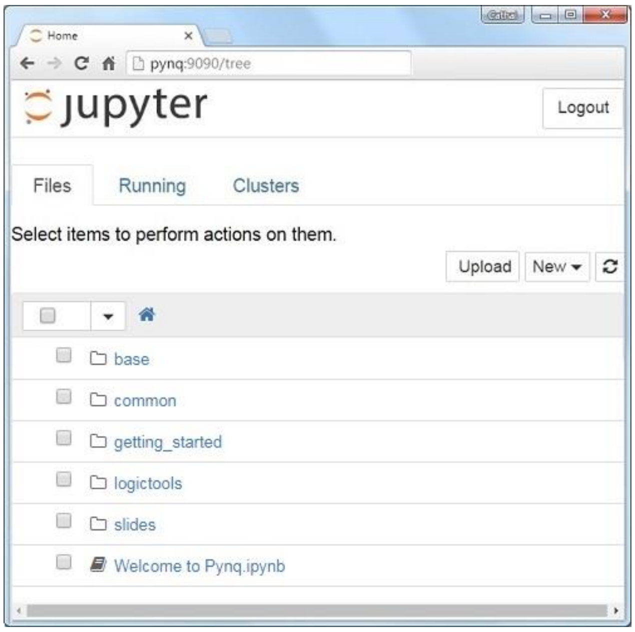


Figure 2: Setting up the PYNQ board

I. Set to board to boot from the micro-SD card by setting the Boot jumper to the SD position.
II. To power the PYNQ-Z2 from the micro USB cable, set the Power jumper to the USB position.
III. Insert the Micro SD card loaded with the PYNQ-Z2 image into the Micro SD card slot underneath the board.

IV. Connect the USB cable to your PC, and to the PROG - UART MicroUSB port on the board
V. You have been provided with a Ethernet-USB dongle and ethernet cable to connect your PYNQ board to the same local network as your PC. Plug the ethernet cable into the PYNQ-Z2 board and use the dongle to connect your board to a USB port on your PC.
VI. Turn on the PYNQ-Z2.
VII. The PYNQ board is configured to assign a default static IP address of 192.168.2.99 and the hostname *pynq.* Open your web browser and browse to http://192.168.2.99 to connect to Jupyter Notebooks.
VIII. You will be prompted to login. Enter xilinx as the username and password and you should see the following screen:



Select items to perform actions on them.

Figure 3: Jupyter Notebooks

4.2 Load the Overlay onto the Board

I. Create a folder for your project then drag and drop your .bit and .hwh files, as well as the provided fir_test.ipynb Jupyter Notebook into the folder.
II. Open the fir_test.ipynb notebook.
III. Edit the path to the overlay to match the location of your .bit and .hwh files.
IV. Follow the steps in the notebook to play the output of your filter and plot the filter output. To run the code in a cell press ctrl + enter.
V. After you have implemented your FIR filter in hardware, implement it in software using the notebook. To do this, you will need to copy your filter coefficients into the notebook.
VI. Note the difference in execution time and compare the filter outputs on a graph.

5 Submission

Please submit a **brief** lab report containing:

- A brief description of the designed FIR filter and its desired effect.
- A brief description of the role of each IP in the overlay.
- Graphs of the audio before and after applying the FIR filter.
- A graph comparing the hardware filter output to the software filter output.
- Compare the trade-offs between implementing the filter in hardware verses software.

Please submit the following in a zipped folder using your name and lab3_p2 as the file name (e.g. AWalsh_lab3_p2):

- Your Jupyter Notebook saved with the name format AWalsh_lab3_p2.ipynb
- A pdf of your lab report
- Your .bit and .hwh files