arm Education

L.EEC025 - Fundamentals of Signal Processing (FunSP)

2023/2024 – 1st semester

Week09, 13 Nov 2023

Objectives:

-modifying and measuring the frequency response of FIR filters running in real-time on the STM32F746G Discovery board:

- **designing an equiripple linear-phase FIR of order 80**
- **creating two modified versions of the designed FIR filter**
- **obtaining experimentally the frequency responses of all three filters and comparing them to the theoretical ones**

DSP Education Kit

LAB 8

Finite Impulse Response (FIR) Filters

Issue 1.0

Contents

1 Introduction

1.1 Lab overview

The examples in this exercise introduce some of the concepts of Finite Impulse Response (FIR) filtering. Also explored are various methods of modifying and estimating the magnitude frequency response of FIR filters implemented in real-time.

2 Requirements

To carry out this lab, you will need:

- An STM32F746G Discovery board
- A PC running Keil MDK-Arm
- MATLAB
- An oscilloscope
- 3.5 mm audio jack cables + BNC cables
- An audio frequency signal generator

3 The FIR filter to be tested and modified

The FIR filter that we test and modify experimentally in this lab is an equiripple linear-phase filter of order 80 (this means that the length of its impulse response is 81 samples). The filter has a peculiar frequency response magnitude so as to facilitate its experimental assessment. The filter is designed in Matlab using the Parks MacClellan optimization algorithm that is called by the Matlab command `firpm()`:

```
order=80;
f=[0 0.465 0.535 1.0];
m=[1 1 0 1];
w=[1 1];
h=firpm(order,f,m,w);

frange=pi*[-1:1/256:1-1/256];
[H W]=freqz(h,1,frange);
plot(W/pi, abs(H), 'LineWidth', 2)
xlabel('\omega/\pi')
ylabel('GAIN')
```

The corresponding frequency response magnitude is as illustrated in Figure 1.

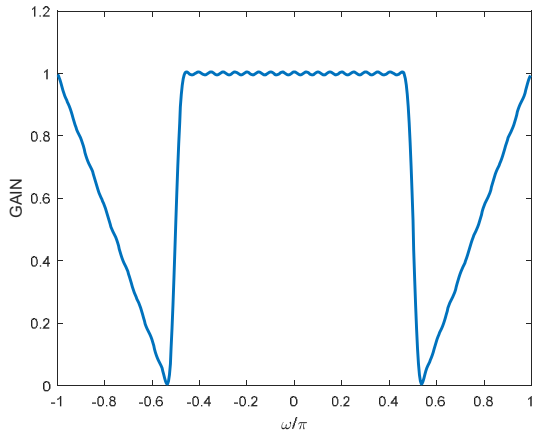
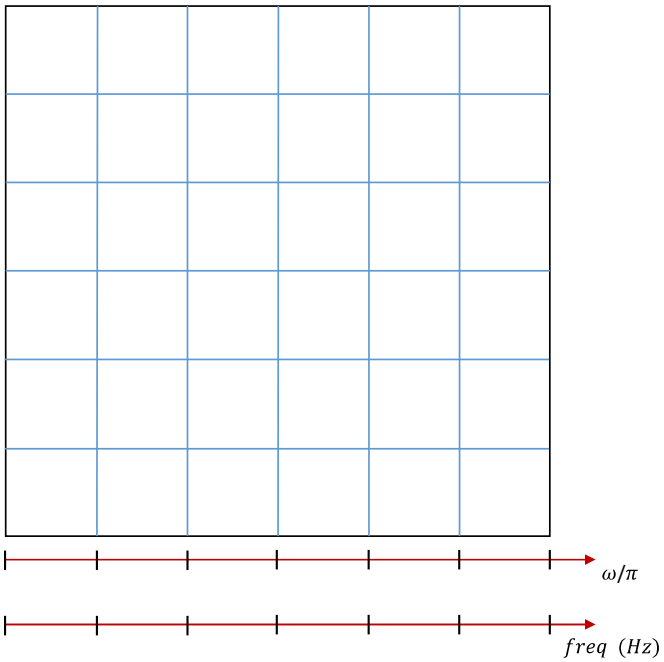


Figure 1: Frequency response magnitude of the FIR filter to be tested in the lab

Question 1 [1 pt / 10 **]**: Assuming that the sampling frequency is 8 kHz, represent graphically the frequency response magnitude that you expect to observe in the lab. Represent frequency using two different but equivalent frequency axes: ω/π (in the range [0, 1]) and `freq` (in Hz, in the range [0, F_N] where F_N is the Nyquist frequency).



In particular, specify the exact frequencies, in Hertz, that correspond to the upper limit of the lower band ($\omega/\pi = 0.465$), and the lower limit of the upper band ($\omega/\pi = 0.535$) where, very approximately, $H(e^{j\omega}) \cong 0$.

$\frac{\omega}{\pi} = 0.465 \quad \rightarrow$ (Hz)

$\frac{\omega}{\pi} = 0.535 \quad \rightarrow$ (Hz)

In this lab, we will also perform and assess experimentally the impact of two types of filter modification that have been studied in Problem 2 of the week05 TP class (video is available on Moodle). In fact, if the impulse response of our linear-phase FIR is $h[n]$, and if its Z-Transform is $H(z)$, we will create two modified versions of $h[n]$ such that their Z-Transforms are $H(z^2)$, and $H(-z)$. The impact of these transformations are as illustrated in Figure 2 (not necessarily according to this order).

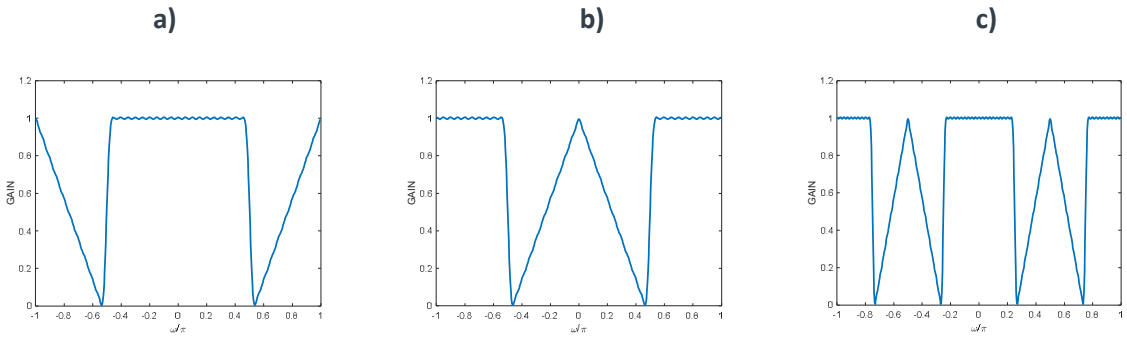


*Figure 2: Frequency response transformations of an original FIR filter **a)**.*

Question 2 [1 pt / 10 **]**: If the Z-Transform of $h[n]$ is $H(z)$, indicate what transformation in $h[n]$ causes its Z-Transform to become $H(z^2)$. Specify that modification for all n describing the modified (or new) impulse response. Implement the modification in Matlab and check the resulting frequency response. What frequency reponse (**b)** or **c)**) in Figure 2 corresponds to this modified filter ?

Answer:

Question 3 [1 pt / 10 **]**: If the Z-Transform of $h[n]$ is $H(z)$, indicate what transformation in $h[n]$ causes its Z-Transform to become $H(-z)$. Specify that modification for all $n = 0,1, \cdots N-1$, where N=81. Implement the modification in Matlab and check the resulting frequency response. What frequency reponse (**b)** or **c)**) in Figure 2 corresponds to this modified filter?

Answer:

4 FIR Filter with Coefficients Specified in Separate Header File

In previous labs, we tested FIR and IIR filters whose coefficients were specified explicitly on the C code. This is quite acceptable when the number of coefficients is low, in the order of 10, or less. When the number of coefficients is large, it is more practical to specify the filter coefficients in a separate header file. That is the case of the current lab that uses a header file (`FPS81.h`) that is included in the C code baseline to be used in this lab: `stm32f7_fir_intr_FPS81.c` . Both files are included in a ZIP that is available on Moodle.

Thus, the C code `stm32f7_fir_intr_FPS81.c` implements an FIR filter for which the filter coefficients $h[n]$ are not specified within this source file but are read from a separate header file which is included by using the preprocessor command

`#include "FPS81.h"`

This also means that in order to change the characteristics of the FIR filter implemented, it is sufficient to change in the preprocessor command the name of the header file to be included, and *Rebuild* the project. For more on this possibility, see Section 7.

5 Completing the C code

In this lab experiment, we will use the `main()` project file that is named **`stm32f7_fir_intr_FPS81.c`** and that is available on the Moodle platform. Its C code, which is not complete, is listed next.

```
// stm32f7_fir_intr_FPS81.c

#include "stm32f7_wm8994_init.h"
#include "FPS81.h"
#include "stm32f7_display.h"
#define SOURCE_FILE_NAME "stm32f7_fir_intr_FPS81.c"

extern int16_t rx_sample_L;
extern int16_t rx_sample_R;
extern int16_t tx_sample_L;
extern int16_t tx_sample_R;

#define Nm1 2*N-1

enum filtertype{Plain, Shifted, Upsampled};

float32_t x[Nm1];
```

```
void BSP_AUDIO_SAI_Interrupt_CallBack()
{
  int16_t i,j;
  float32_t yn = 0.0;


  // uncomment just one of the following three lines
   enum filtertype myfilter=Plain;
  // enum filtertype myfilter=Shifted;
  // enum filtertype myfilter=Upsampled;


  x[0]  = (float32_t)(rx_sample_L);
  BSP_LED_On(LED1);

  switch (myfilter)
  {
        case Plain:
           for (i=0 ; i<N ; i++) yn += h[i] * x[i];
           break;
        case Shifted:
           // please complete your code here
           break;
        case Upsampled:
           // please complete your code here
           break;
        default:
           for (i=0 ; i<N ; i++) yn += h[i] * x[i];
  }

  for (j=(Nm1-1) ; j>0 ; j--) x[j] = x[j-1];

  BSP_LED_Off(LED1);
  tx_sample_R = (int16_t)(yn);
  tx_sample_L = tx_sample_R;

  return;
}

int main(void)
{
  stm32f7_wm8994_init(AUDIO_FREQUENCY_8K,
                      IO_METHOD_INTR,
                      INPUT_DEVICE_INPUT_LINE_1,
                      OUTPUT_DEVICE_HEADPHONE,
                      WM8994_HP_OUT_ANALOG_GAIN_0DB,
                      WM8994_LINE_IN_GAIN_0DB,
                      WM8994_DMIC_GAIN_9DB,
                      SOURCE_FILE_NAME,
                      NOGRAPH);

  while(1){}
}
```

Before you compile and run this code, you need to complete it (before the lab class!).

Question 4 [1 pt / 10]: in the C code, under "`case Shifted:`" which corresponds to one of the above filter transformations, replace the comment "`// please complete your code here`" by one of the following 3 alternatives (only one is correct):

Alternative A:

```
for (i=0 ; i<N ; )
{
   yn += h[i] * x[i]; i++;
}
```

Alternative B:

```
for (i=0 ; i<N ; )
{
   yn -= h[i] * x[i]; i++;
}
```

Alternative C:

```
for (i=0 ; i<(N-1) ; )
{
   yn += h[i] * x[i]; i++;
   yn -= h[i] * x[i]; i++;
}
yn += h[i] * x[i];
```

Explain what alternative is correct taking into consideration your answer to either Question 2, or Question 3.

Question 5 [1 pt / 10]: in the C code, under "`case Upsampled:`" which corresponds to one of the above filter transformations, replace the comment "`// please complete your code here`" by one of the following 3 alternatives (only one is correct):

Alternative A:

```
for (i=0, j=0 ; i<N ; i++)
{
   yn += h[i] * x[j];
   j+=2;
}
```

Alternative B:

```
for (i=0, j=0 ; i<N ; i++)
{
   yn += h[2*i] * x[j];
   j++;
}
```

Alternative C:

```
for (i=0, j=0 ; i<N ; i++, j++)
{
   yn += h[2*i] * x[2*j];
}
```

Explain what alternative is correct taking into consideration your answer to either Question 2, or Question 3.

6 Measuring the Frequency Responses

After you complete the C code as recommended in the previous section, proceed, as indicated next, to compile the code, upload it to the STM32F7 board, and to run it.

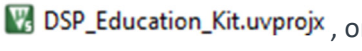
After unzipping them, take the `stm32f7_fir_intr_FPS81.c` (which should be completed according to the suggestions in Section 5) and `FPS81.h` files to the "src" directory that is located under the folder:

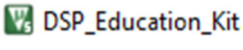
C:\uvision\Keil\STM32F7xx_DFP\2.9.0\Projects\STM32746G-Discovery\Examples\DSP Education Kit\

Remember that the directory where you can find the the **`DSP_Education_Kit.uvprojx`** project file is:

C:\uvision\Keil\STM32F7xx_DFP\2.9.0\Projects\STM32746G-Discovery\Examples\DSP Education Kit\MDK-ARM

You can copy-paste this link directly to File Explorer in Windows for a quick and easy access. For your convenience, this link is also available on a TXT file on Moodle.

As in previous labs, we use the **`DSP_Education_Kit.uvprojx`** project file as our baseline project. This project file is represented by the icon 

, or just 

. Double-click on this file/icon. This starts the Keil MDK-Arm development

environment (µVision). Replace the existing `main()` file in that project by the new `main()` that is `stm32f7_fir_intr_FPS81.c` .

Now, proceed as usual to compile the code, downloading it to the STM32F746G board (by starting the debugger), and then to run the code.

6.1 Setting-up for this lab experiment

Set the function generator to generate a sine wave having 5 Vpp and 100 Hz. Using a "T" and a BNC-BNC cable, take the output of the function generator to CHAN1 of the oscilloscope. Connect the output of a sinusoidal signal generator (i.e. the function generator) to the (left channel of the) LINE IN socket on the Discovery board (Remember: make sure that you use the adapter with the **blue mini-jack** whose interface board has a resistor divider. It is meant to protect the analog input of the ***kit*** against excessive voltage levels).

Then, using another BNC-BNC cable, take the LEFT channel (yes, LEFT channel) of the STM32F746G LINE OUT output to the CHAN2 input of the oscilloscope.

Using the oscilloscope SETTINGS button and menu, make sure that the Vpp and frequency of both input and output signals are being measured in real-time.

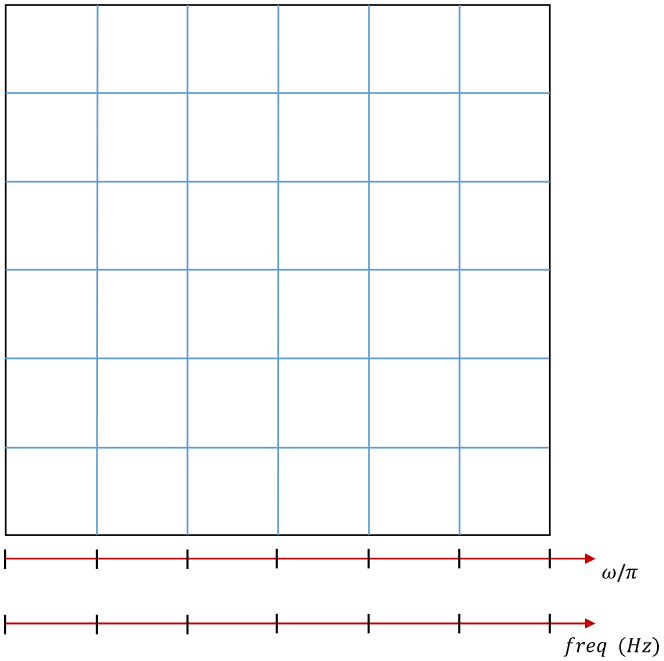
6.2 Sketching the frequency response of the baseline FIR filter

The frequency response of a filter reflects its gain at different frequencies. As seen in previous lab experiments, one way of assessing the frequency response of the filter is simply to measure its gain using a sinusoidal input signal at a few frequencies of interest.

In this section, we measure the magnitude frequency response of the baseline FIR filter that is signaled in the C code as "Plain", and whose theoretical frequency response is depicted in Figure 1.

In this experiment, you will vary the frequency from 100 Hz, up to 3.8 kHz, and take note of the Vpp and frequency of the output wave represented on the oscilloscope. Only four frequencies should suffice: 100 Hz (which we will take as an approximation for 0 Hz), 3800 Hz (which we will take as an approximation for 4 kHz), and two other frequencies that are anticipated in Question 1.

Based on these measurements, sketch in the following plot the approximate frequency response magnitude of the baseline FIR filter that is implemented by the above C code when you vary the frequency from 100 Hz up to 3.8 kHz.



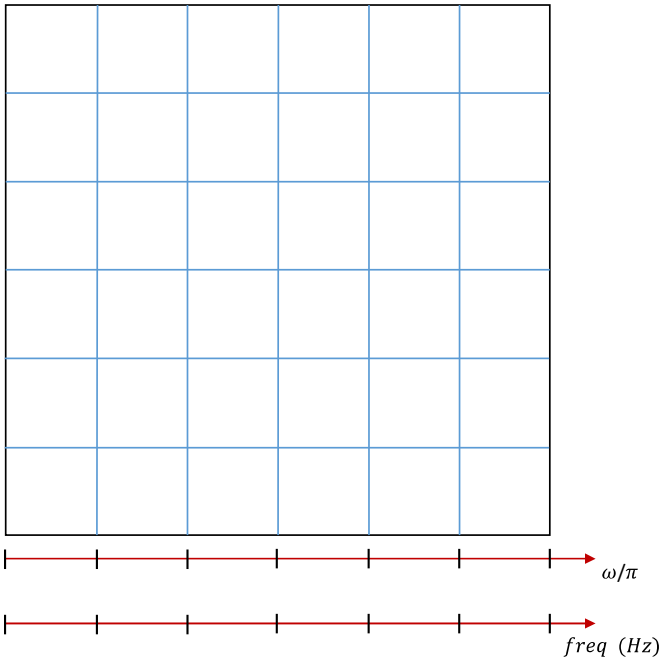
Question 6 [1 pt / 10 **]**: Is this sketch consistent with the expected frequency response magnitude that is represented in Figure 1 and in Question 1 ?

6.3 Sketching the frequency response of the H(-z) FIR filter

In this section, we measure the magnitude frequency response of the modified FIR filter whose transfer function is H(-z) and that is signaled in the C code as one of the two "Shifted" or "Upsampled "cases. Its theoretical frequency response is depicted in Figure 2.

In this experiment, you will vary the frequency from 100 Hz, up to 3.8 kHz, and take note of the Vpp and frequency of the output wave represented on the oscilloscope. As in the previous experiment, only a few frequencies should suffice in addition to 100 Hz (which we will take as an approximation for 0 Hz), and 3800 Hz (which we will take as an approximation for 4 kHz).

Based on these measurements, sketch in the following plot the approximate frequency response magnitude of the H(-z) FIR filter when you vary the frequency from 100 Hz up to 3.8 kHz.



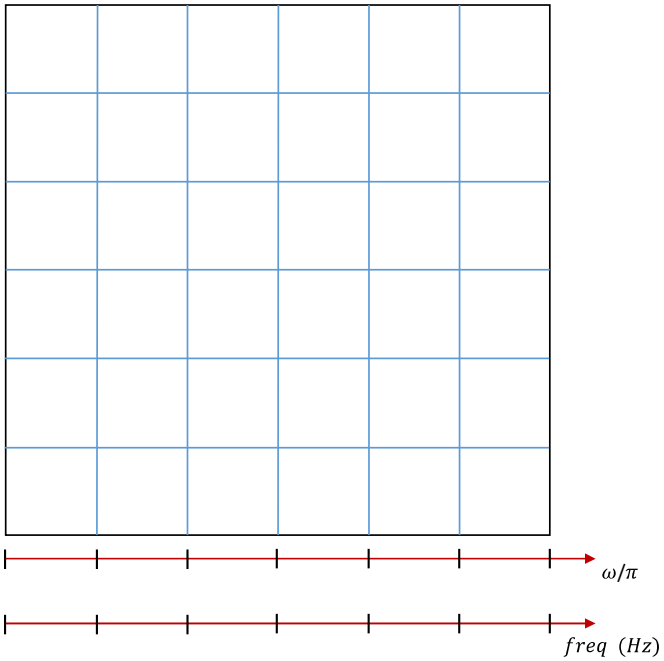
Question 7 [2 pt / 10 **]**: Is this sketch consistent with the expected frequency response magnitude that is represented in Figure 2 ?

6.4 Sketching the frequency response of the H(z^2) FIR filter

In this section, we measure the magnitude frequency response of the modified FIR filter whose transfer function is H(z^2) and that is signaled in the C code as one of the two "Shifted" or "Upsampled "cases. Its theoretical frequency response magnitude is depicted in Figure 2.

In this experiment, you will vary the frequency from 100 Hz, up to 3.8 kHz, and take note of the Vpp and frequency of the output wave represented on the oscilloscope. As in the previous experiment, only a few frequencies should suffice in addition to 100 Hz (which we will take as an approximation for 0 Hz), and 3800 Hz (which we will take as an approximation for 4 kHz).

Based on these measurements, sketch in the following plot the approximate frequency response magnitude of the H(z^2) FIR filter when you vary the frequency from 100 Hz up to 3.8 kHz.



Question 8 [2 pt / 10 **]**: Is this sketch consistent with the expected frequency response magnitude that is represented in Figure 2 ?

Question 9: All of the above implementations of linear-phase filters are not totally efficient. What else could be done in order to improve the realization/implementation efficiency ?

7 Extra-class: Generating FIR Filter Coefficient Header Files Using MATLAB

If the number of filter coefficients is small, a coefficient header file may be edited by hand. To be compatible with program `stm32f7_fir_intr.c` and others, a coefficient header file must define a constant `N` and declare and initialize the contents of an array `h[]`, which contains `N` floating point values.

For larger numbers of coefficients, the MATLAB function `stm32f7_fir_coeffs()`, defined in file `stm32f7_fir_coeffs.m` (and that is included on the ZIP that is available on Moodle), can be used. This function should be passed a MATLAB array of real-valued coefficient values and will prompt the user for an output filename.

For example, the coefficient file `maf5.h` was created by typing the following at the MATLAB command prompt.

```
>> x = [0.2, 0.2, 0.2, 0.2, 0.2];
>> stm32f7_fir_coeffs(x)
enter filename for coefficients maf5.h
```

The coefficient filename must be entered in full, including the suffix `.h`.

Alternatively, the MATLAB filter design and analysis tool `fdatool` can be used to calculate FIR filter coefficients and to export them to the MATLAB workspace (File – Export… – Export To Workspace / Export As Coefficients). Then, function `stm32f7_fir_coeffs()` can be used to create a coefficient file compatible with programs including `stm32f7_fir_intr.c`. It is recommended that the filter coefficients values passed to function `stm32f7_fir_coeffs()` are normalized such that their gain is unity. `fdatool` does it automatically, but if you are designing filter coefficients without the aid of `fdatool`, you should aim for a passband gain of 1.

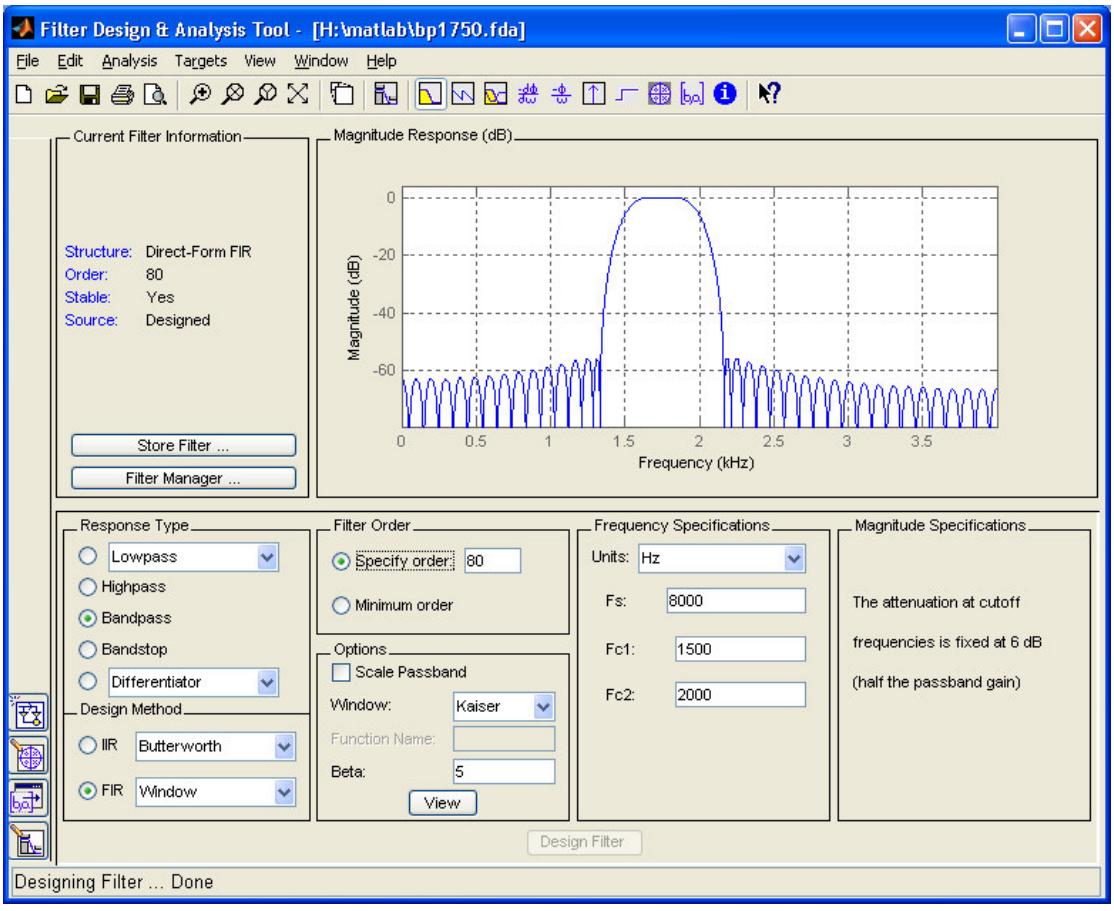


Figure 3: Design of a bandpass FIR filter using MATLAB ***fdatool***

Coefficient header file `bp1750.h` was generated using MATLAB function `stm32f7_fir_coeffs()` after designing the filter using `fdatool` (as shown in 3).

Note: Additional header and source files are provided in the lab-specific folder provided.

The file `maf5.h` listed in the following code snippet contains filter coefficient values that will result in implementation of a five-point moving average filter.

```
// maf5.h
// this file was generated using function stm32f7_fir_coeffs.m

#define N 5

float h[N]={0.2, 0.2, 0.2, 0.2, 0.2};
```

8 Conclusions

This laboratory exercise has introduced FIR filters and explored several different frequency response modifications.

9 Additional References

Moving average filters:

https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_Ch15.pdf