Lab Exercise #3:   
Finite impulse response (FIR) filters

# Overview

The examples in this exercise introduce some of the concepts of finite impulse response (FIR) filtering. Also explored are various different methods of estimating the magnitude frequency response of a filter implemented in real time, and the relative computational efficiency of different implementation options.

Instructor note: Items in bright red are solutions to be deleted before posting for students.

## Hardware

To carry out this exercise you will need a STM32F407 Discovery EVM with audio interface card, an oscilloscope, an audio frequency signal generator, a PC with a soundcard running *MDK ARM*, *GoldWave* and *MATLAB*, and various connecting cables.

### The moving average filter

This filter is widely used in DSP and, arguably, is one of the easiest of all digital filters to understand. It is particularly effective at removing (high frequency) random noise from a signal.

The moving average filter operates by taking the arithmetic mean, or average, value of a number of past input samples, in order to form each output sample. This may be represented by the equation

Where *x*(*n*) represents the *n*th sample of an input signal and *y*(*n*) the *n*th sample of the filter output and which is equal to the average value of the previous *N* input samples. A five point moving average filter is implemented by the example program stm32\_average\_intr.c.

### frequency response of The moving average filter

A simple experiment, using the test signal contained in file mefsin.wav demonstrates that the moving average filter attenuates some frequency components of a signal more than others.

Listen to the test signal using *GoldWave*, *Windows Media Player*, or similar. It contains a recording of some speech corrupted by the addition of a sinusoidal tone. Then connect the PC soundcard output to the LINE IN socket on the audio card. Run program stm32\_average\_intr.c on the Discovery and use headphones connected to the audio card to listen to the filtered version of the test signal. You should find that the sinusoidal tone has been attenuated very significantly and that the speech sounds less bright than in the original signal. Both observations are consistent with the moving average filter having a low pass frequency response.

The following exercises introduce a number of different, and more quantitative, methods of assessing the frequency response of the filter.

#### Observation of frequency response using a sinusoidal input signal

The frequency response of a filter tells us its gain at different frequencies and hence one way of assessing the frequency response of the filter is simply to measure its gain using a sinusoidal input signal at a number of different frequencies. As shown in figure 1, connect the output of a sinusoidal signal generator to the (left channel of the) LINE IN socket on the audio card and connect the (left channel of the) LINE OUT socket either to an oscilloscope or to the input of the soundcard on a PC running *GoldWave*. Run program stm32\_average\_intr.c and vary the frequency of the applied sinusoid between 100 Hz and 5000 Hz. Keep the amplitude of the sinusoidal input signal constant at approximately 2 volts peak to peak. Record the amplitude of the output signal, at a number of different frequencies, on the axes of figure 4.

Figures 2 and 3 show the output from program stm32\_average\_intr.c displayed using the FFT function of a *Rigol DS1052E* oscilloscope and using the spectrum display in *GoldWave*. The input signals were sinusoids at frequencies of 2.0 kHz and 1.3 kHz respectively. Figure 2 might be plotted as a single point on the axes of figure 4 at 2.0 kHz and -24dB, and figure 3 might be plotted as a single point on the axes of figure 4 at 1.3 kHz and -30dB. However, the absolute level, in dBs, is not important (and depends on the amplitude of the input signal). What is important is its relative level across the range of frequencies.

As the frequency of the inputs signal is varied, the amplitude of the output signal should change. The gain of the filter is higher at low frequencies than at high frequencies and there are some frequencies at which the gain is almost zero. Overall, the moving average filter has a low pass characteristic.



Figure 1. Connection diagram for measuring the magnitude frequency response of the five point moving average filter implemented by program stm32\_average\_intr.c using a signal generator and an oscilloscope.

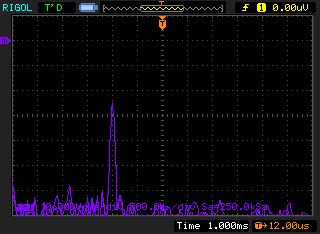


Figure 2. Output signal from program stm32\_average\_intr.c viewed using FFT function of *Rigol DS1052* oscilloscope. It’s difficult to read, but the text at the bottom of the screen indicates 500 Hz/div and 10 dBVrms/div. The marker on the left hand side of the display indicates a level of 0 dBVrms.

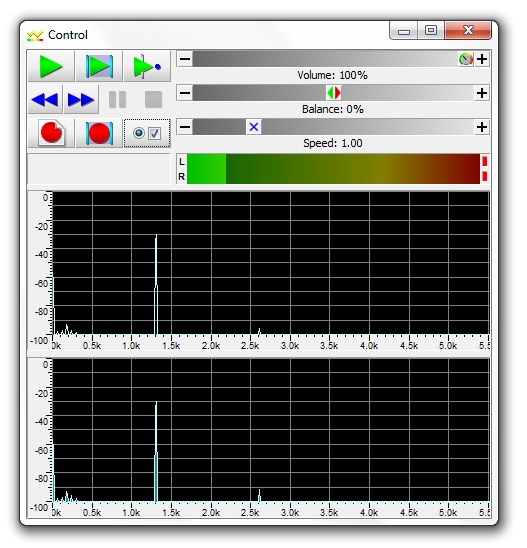


Figure 3. Output signal from program stm32\_average\_intr.c viewed using spectrum display in *GoldWave*.

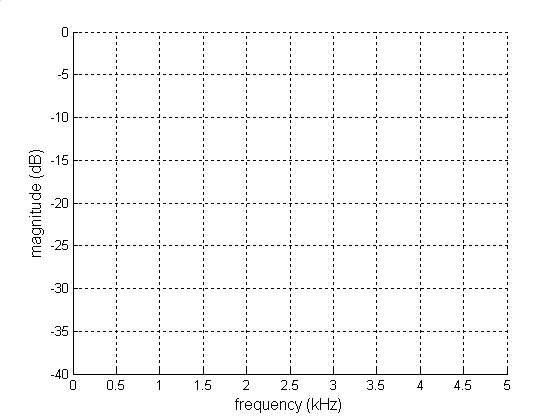


Figure 4. Magnitude frequency response of five point moving average filter implemented by program stm32\_average\_intr.c and measured using sinusoidal input signals.

#### Observation of frequency response using a pseudo-random input signal

Alternatively, an indication of the magnitude frequency response of the filter may be obtained by applying a pseudo-random input signal, containing equally-weighted components at all frequencies, to the filter and observing the spectral content of the filter output. Program stm32\_average\_prbs\_intr.c combines the moving average filter of program stm32\_average\_intr.c with a pseudo-random noise generator (implemented within the program using function prbs()) .

Use either the FFT function of an oscilloscope or *GoldWave* to view the spectral content of the filtered noise output by the program. Sketch what you see on the axes of figure 5 and verify that the results indicate a similar magnitude frequency response to that measured using a signal generator and plotted in figure 4.

Using pseudo-random noise as a test signal is a quick and easy method of obtaining an indication of the magnitude frequency response of a filter.

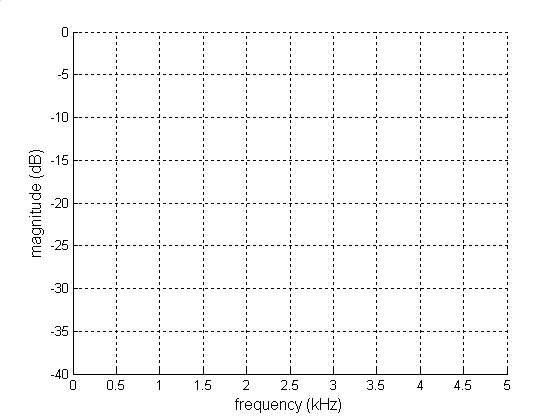


Figure 5. Magnitude frequency response of five point moving average filter measured using program stm32\_average\_prn\_dma.c.

#### Identification of magnitude frequency response using two Discovery boards

In lab exercise #2, program stm32\_sysid\_CMSIS\_intr.c was used to identify the characteristics of the antialiasing and reconstruction filters of the WM5102 codec. Here, the same program is used to identify the characteristics of the moving average filter. Connect two Discovery boards (and audio cards) as shown in figure 6. On one of the boards run program stm32\_average\_intr.c (once this has been downloaded into flash memory, the program will run when power is applied to the Discovery board via the USB cable) and on the other run program stm32\_sysid\_CMSIS\_intr.c. The latter program identifies the characteristics of whatever system is connected across its output and input. At this stage, there is no need to understand how the adaptive filter in program stm32\_sysid\_CMSIS\_intr.c works. After program stm32\_sysid\_CMSIS\_intr.c has run for a few seconds, halt the program and save the values of the 256 filter coefficients realCoeffs32 to a data file, as you did in lab exercise #2.



Figure 6. Connection diagram for identification of magnitude frequency response of five point moving average filter using program stm32\_sysid\_CMSIS\_intr.c.

Then use MATLAB function stm32f4\_logfft() to display the adaptive filter coefficients FirCoeffs32 and the magnitude of their FFT. Figure 7 shows the result plotted on the same axes as the theoretical magnitude frequency response of the five point moving average filter.

What are the differences between the measured and theoretical frequency responses, and how do you explain them?

The measured response rolls off at frequencies below 100 Hz due to a number of factors including the ac-coupling of the LINE IN and LINE OUT connections to the codec on the audio card and a high pass digital filter used immediately after the ADC in the codec. The magnitude frequency response rolls off at frequencies above 3500 Hz due to the digital reconstruction filters in the DACs and the digital anti-aliasing filters before the ADCs. The measured frequency response is slightly lower overall than the theoretical response. There are potential dividers between the LINE IN input sockets on the audio card and the WM5102 inputs (L and R) that introduce attenuation to the signal path. This attenuation has been compensated for by programming an 8dB gain into the signal path through the codec. However, this 8dB gain may not match exactly the attenuation due to external circuitry.

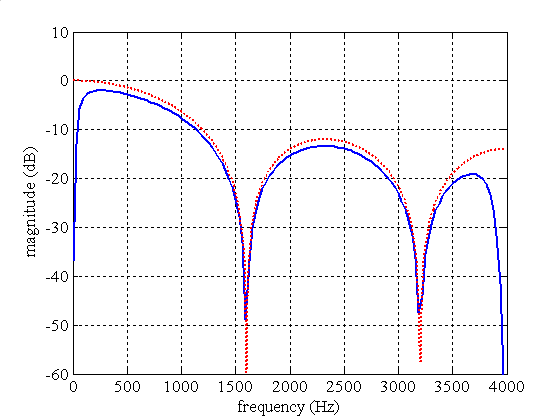


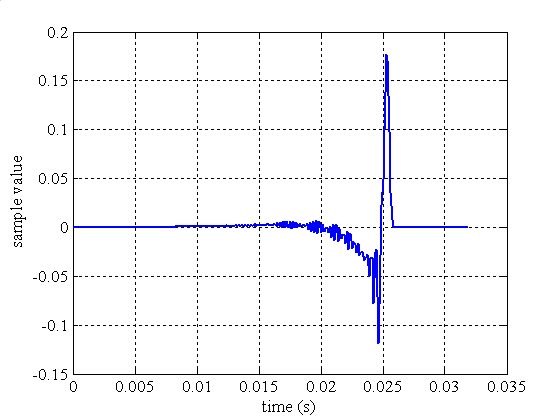
Figure 7. Magnitude frequency response of the five point moving average filter implemented using program stm32\_average\_intr.c on one Discovery board, identified using program stm32\_sysid\_CMSIS\_intr.c running on a second Discovery board and plotted on the same axes as its theoretical magnitude frequency response.

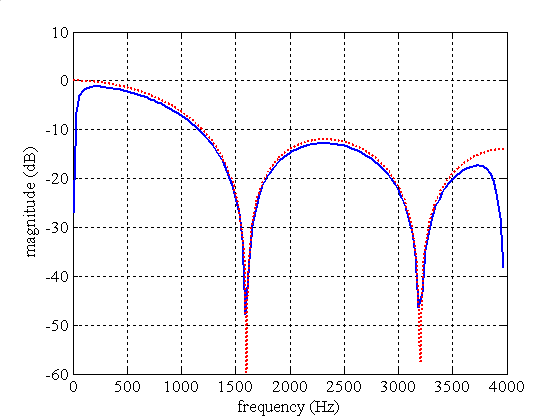
#### Identification of magnitude frequency response using one Discovery board

Using just one Discovery board (and audio card) you can perform an experiment similar to the previous one by implementing the moving average filter before the pseudo-random noise is written to the DAC. Connect LINE OUT on the audio card to LINE IN, as shown in figure 8, and build and load program stm32\_sysid\_average\_CMSIS\_intr.c. Run the program for a few seconds. You can listen to the filtered noise while the program is running if you connect headphones to the audio board. Follow the same procedure as before in order to save the adaptive filter coefficients to a file, and plot them using MATLAB.

Can you see any difference between the magnitude response measured this way, relative to that measured using two Discovery boards?

Since the signal path identified in this case includes only one DAC, LINE OUT connection, ADC and LINE IN connection the roll-offs of gain below 100 Hz and above 3500 Hz are slightly less pronounced, but this may be difficult to identify. It’s evident from the impulse response (which runs from right to left in these graphs) that there is less delay in the shorter identified path.





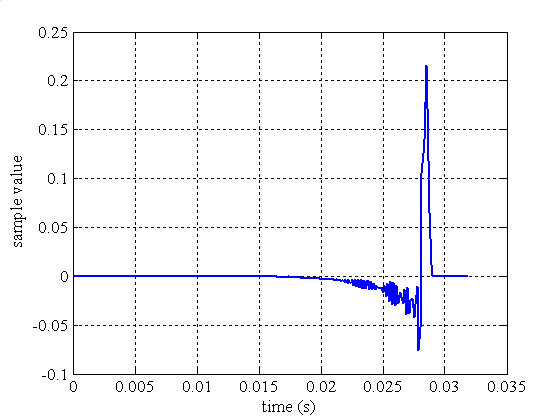




Figure 8. Connection diagram for identification of magnitude frequency response of five point moving average filter using program stm32\_sysid\_average\_CMSIS\_intr.c.

#### Higher order moving average filters

What would happen if the moving average were calculated over a different number of previous samples?

Modify program stm32\_average\_prbs\_intr.c so that *N* = 11 and observe the frequency response of the filter using either the FFT function of an oscilloscope or using*GoldWave*. Record your observations on the axes of figure 9.

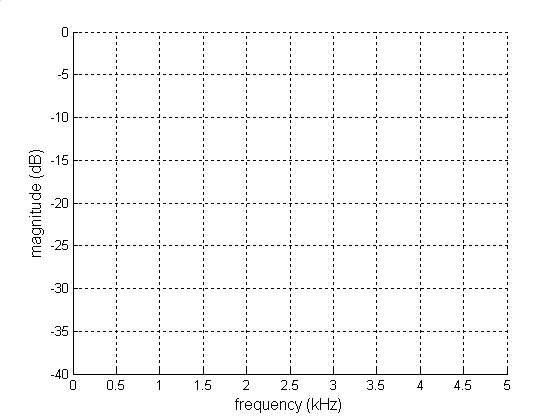


Figure 9. Magnitude frequency response of eleven point moving average filter measured using *GoldWave*.

### Finite impulse response (FIR) filters

What would happen if the values of the filter coefficients *h*(*n*) were changed?

Modify program stm32\_average\_prbs\_intr.c so that *N* = 5 and *h*(*n*) = {0.0833, 0.2500, 0.3333. 0.2500, 0.0833}. Observe the frequency response of the filter using *GoldWave* and record your observations on the axes of figure 10.

Either initialize the values of array h[] when it is declared, and remove the for loop, at the start of main(), that sets the values to 1.0/N , or add code, at the start of main(), that initializes h[].

You should find that the higher frequency components of the input signal (pseudo-random noise) have been attenuated more than they were by the five point moving average filter and also that the `notches' in the frequency response at 1600 Hz and 3200 Hz have disappeared. You have effectively applied a *Hann window* to the coefficients of the five point moving average filter.

According to some accounts, you have actually applied a seven-point, rather than a five-point, Hann window since the first and last values of the window are equal to zero.

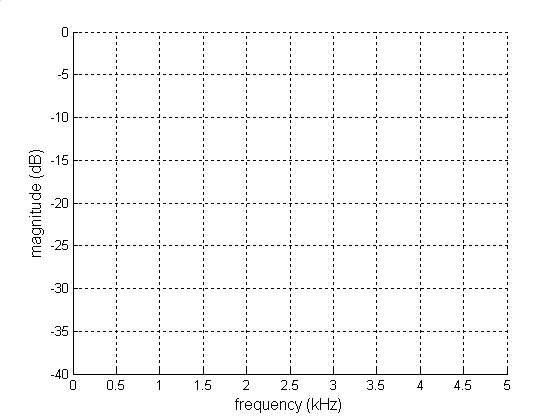


Figure 10. Magnitude frequency response of five point moving average filter with Hann window measured using *GoldWave*.

Changing the values of the filter coefficients has (not unexpectedly) changed the magnitude frequency of the filter.

Use the discrete time Fourier transform (DTFT) to derive an algebraic expression for the frequency response of the filter having coefficients *h*(*n*) = {0.0833, 0.2500, 0.3333. 0.2500, 0.0833}. Write down each step of the derivation in the space provided in figure 11 and sketch the magnitude of that theoretical frequency response on the axes of figure 12.



Figure 11. Derivation of theoretical frequency response of five point moving average filter with Hann window using DTFT.

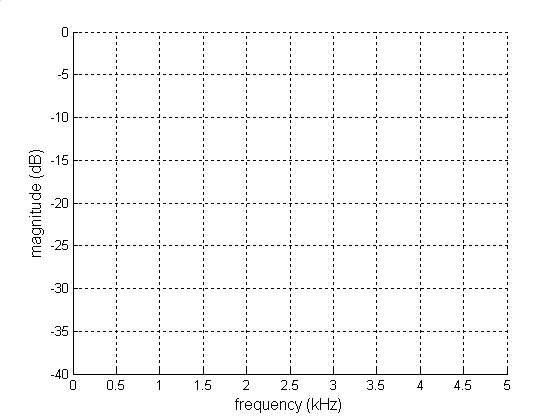


Figure 12. Theoretical magnitude frequency response of five point moving average filter with Hann window.

### FIR filter programs with coefficients specified in separate header file

Programs stm32\_fir\_intr.c and stm32\_fir\_prbs\_intr.c implement FIR filters with analogue input from LINE IN on the audio board and pseudo-random input from function prbs() respectively. In both cases, the filter coefficients *h*(*n*) are not specified within these source file but are read, when compiling the programs, from a separate header file. In order to change the characteristics of the filter implemented by these programs, simply change the preprocessor command

#include "ave5.h"

to, for example,

#include "lp33.h"

The file ave5.h listed in figure 13 contains filter coefficient values that will result implementation of a five point moving average filter.

// testave.h

// this file was generated using function stm32f4\_fir\_coeffs.m

#define N 5

float32\_t h[N] = {

2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001

};

Figure 13. Listing of file ave5.h

A number of different example coefficient files have been provided. Investigate the characteristics of one or two of these using program stm32\_fir\_prbs\_intr.c .

Program stm32\_sysid\_fir\_CMSIS\_intr.c is the counterpart of stm32\_sysid\_average\_CMSIS\_intr.c and may be used to investigate too, if you prefer to use system identification rather than looking at filtered pseudo-random noise.

### 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 stm32\_fir\_intr.c and others, a coefficient header file must define a constant N and declare and initialise the contents of an array h[], that contains N floating point values.

For larger numbers of coefficients the MATLAB function stm32f4\_fir\_coeffs(), defined in file stm32f4\_fir\_coeffs.m, can be used. This function should be passed a MATLAB vector of coefficient values and will prompt the user for an output filename.

For example, the coefficient file ave5.h , listed in Figure 13, was created by typing the following at the MATLAB command prompt.

>> x = [0.2, 0.2, 0.2, 0.2, 0.2];

>> stm32\_fir\_coeffs(x)

enter filename for coefficients ave5.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. Then function stm32f4\_fir\_coeffs() can be used to create a coefficient file compatible with programs including stm32\_fir\_intr.c. It is recommended that the filter coefficients values passed to function stm32\_fir\_coeffs() are normalised such that their sum is unity. This may be achieved by typing

>> x = x/sum(x);

at the MATLAB command line (assuming that the vector of filter coefficients is named x).

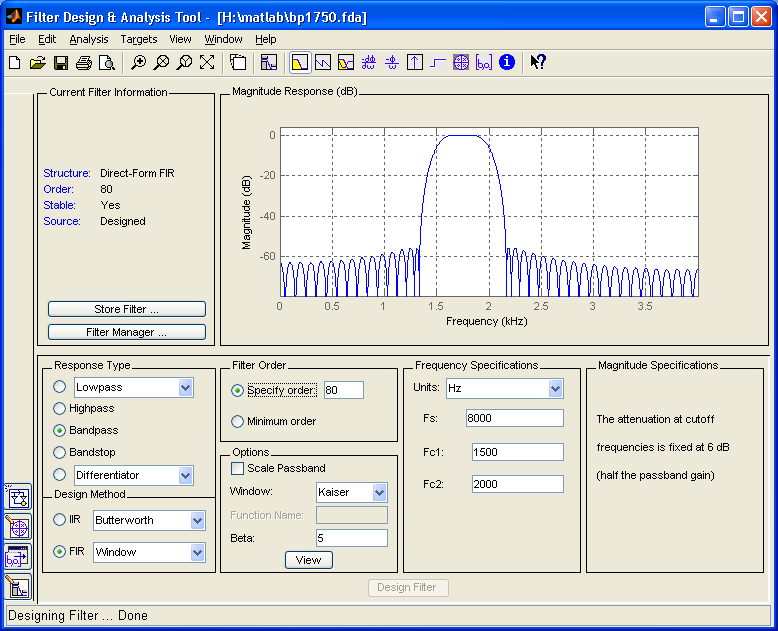


Figure 14. Design of a bandpass FIR filter using MATLAB *fdatool*.

Implementing different filters using coefficient header files

Coefficient header file bp1750.h was generated using MATLAB function stm32f4\_fir\_coeffs() after designing the filter using fdatool.

Incorporate these filter coefficients into program stm32\_sysid\_fir\_CMSIS\_intr.c, run the program, save the values of the 256 adaptive filter coefficients firCoeffs32 to a file and plot them using MATLAB. While program stm32\_sysid\_fir\_CMSIS\_intr.c is running you can listen to the filtered PRBS using headphones. Expected results are shown in figure 15.

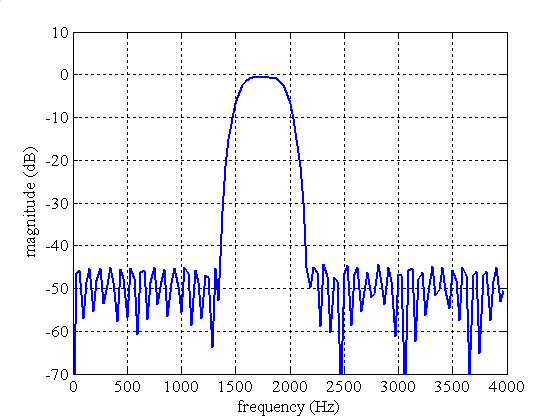
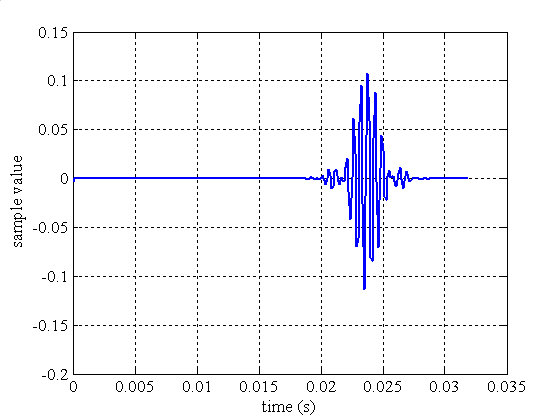
****

Figure 15. Experimentally measured impulse response and magnitude frequency response corresponding to filter coefficients defined in header file bp1750.h, derived using program stm32\_sysid\_fir\_CMSIS\_intr.c.

Some other example coefficient header files are provided. Investigate their characteristics using either the technique outlined above, or using program stm32\_fir\_prbs\_intr.c.

### Frequency responses of simple FIR filters

Using the techniques introduced in this exercise, for the following three sets of coefficients

Filter X: h(n) = {0.2, 0.4, -0.4, -0.2}

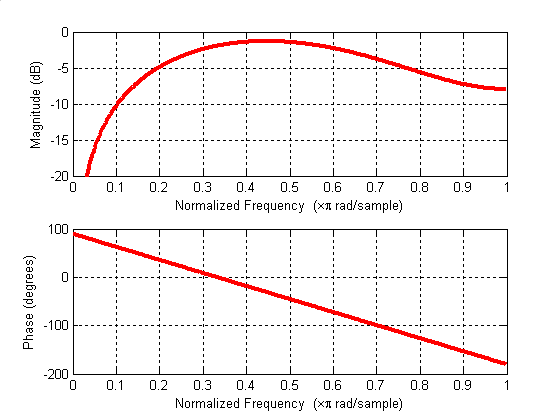
Filter Y: h(n) = {0.5, -0.5}

Filter Z: h(n) = {0.5, 1.0, 0.5}

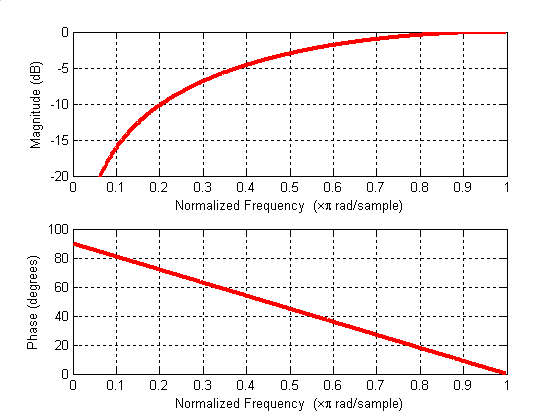
1. Derive the theoretical frequency response, showing your working, and sketching the magnitude and phase frequency responses on axes similar to those in figure X.
2. Estimate the magnitude frequency response using prbs() as input to an experimental real-time implementation of the filter, i.e .use program stm32\_fir\_prbs\_intr.c and print out the results shown in GoldWave or on an oscilloscope with FFT facility.
3. Estimate the magnitude frequency response using program sysid\_fir\_intr.c and plot the results using MATLAB function stm32\_logfft().

For b) and c), discuss how and why the results differ to those derived in a).

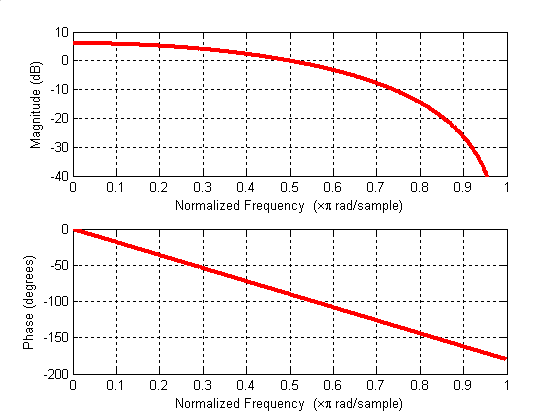
Filter X:



Filter Y :



Filter Z :



### Design a High Pass filter using the window method

Create a coefficient file myhpf.h for use with programs stm32\_fir\_intr.c, and stm32\_fir\_prbs\_intr.c that will implement a high pass filter with a cutoff frequency of 2 kHz and which uses N=31 coefficients. The format of a header file containing filter coefficients is shown by example in figure 23.

If you use MATLAB to create a vector of coefficient values, you can generate a header file using the function stm32\_fir().

Compare the theoretical magnitude frequency response of your design with its magnitude frequency response measured using programs stm32\_fir\_prbs\_intr.c and stm32\_sysid\_fir\_intr.c.

### Execution time of FIR filters coded in C and using CMSIS DSP library function arm\_fir\_f32()

In the programs used so far, e.g. stm32\_fir\_prn\_intr.c, the FIR filter has been implemented quite straightforwardly, without any particular consideration of computational efficiency. Two loops are used. One loop is used to compute the output sample value yn from the contents of arrays h[] and x[], and another is used to shift the contents of the filter delay line in array x[]. The following experiment compares the efficiency of that straightforward implementation with the optimized CMSIS DSP library function arm\_fir\_f32().

The time taken to calculate the value of each output sample may be measured two different ways. One way is to use a timer in *MDK ARM*. Another way is to toggle a GPIO output pin and to use an oscilloscope.

Load program stm32\_fir\_prn\_intr.c and place breakpoints at the statements

GPIO\_SetBits(GPIOD, GPIO\_Pin\_15);

and

GPIO\_ResetBits(GPIOD, GPIO\_Pin\_15);

These statements are in the interrupt service routine SPI2\_IRQHandler() just prior to and just after computing one output sample value.

Make sure that coefficient file bp1750.cof is used by the program (#include “bp1750.cof”). Run the program – it should halt at the first breakpoint – and make a note of the value of the *Sec* item in the *Register* window. This value is the execution time in seconds up to the breakpoint. Run the program again - it should halt at the second breakpoint – and note the value of the *Sec* item again. The difference between the two values of *Sec* is equal to the time taken by the program, in seconds, to compute one output sample, yn.

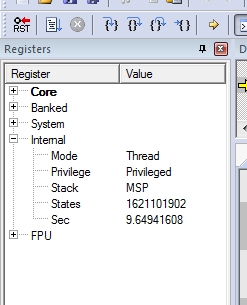


Figure 15. Debug mode *Register* window, showing *Sec* item (execution time in seconds).

Record the time taken to compute one output sample value in table 1.

Remove the breakpoints and run the program. Connect an oscilloscope probe to pin PD15 on the Discovery board and measure the length of time that the output is high. The times measured using the two methods should be approximately equal. The positive pulse on pin PD15 should repeat at the sampling frequency, 8 kHz.

Record the duration of the positive voltage pulses on pin PD15 in table 1.

Next replace source file stm32\_fir\_prn\_intr.c with file stm32\_fir\_prn\_CMSIS\_intr.c and repeat the experiment. In this case, the computation of an output value is carried out using the CMSIS DSP library function arm\_fir\_f32().

Record computation times measured using both methods (breakpoints and pin PD15) in table 1.

Computing one output sample value at a time is not the most efficient way to use function arm\_fir\_f32()which is optimized for block processing.

Program stm32\_fir\_prn\_CMSIS\_dma.c can be used to illustrate this. Replace program stm32\_fir\_prbs\_CMSIS\_intr.c with program stm32\_fir\_prn\_CMSIS\_dma.c and repeat the experiment. In this case, the GPIO pin is set and reset before and after a call to function process\_buffer() which computes BUFSIZE/2 output samples each time it is called. BUFSIZE is defined in header file wm5102\_init.h and its supplied value is 256. Divide the time taken by each call to function process\_buffer() by BUFSIZE/2 to get a time to compare with the previous results.

Recall that the maximum time available for computation in programs stm32\_fir\_prbs\_intr.c and at a sampling rate of 8kHz is 125us. The maximum time available in programs stmdma and stm32dma, i.e. the time between consecutive calls to function process\_buffer() , will be BUFSIZE/2 times one sampling period, i.e. 16ms.

|  |  |  |  |
| --- | --- | --- | --- |
| program | using *Sec* | using GPIO PD15 | divided by BUFSIZE/2 |
| stm32\_fir\_prbs\_intr.c | 10.4us | 10.1us | N/A |
| stm32\_fir\_prbs\_dma.c | 1.48ms | 1.48ms | 11.56us |
| stm32\_fir\_prbs\_CMSIS\_intr.c | 6.89us | 5.84us | N/A |
| stm32\_fir\_prbs\_CMSIS\_dma.c | 316us | 314us | 2.45us |

Table 1: Computation times (per output sample) for N=81 point FIR filter defined in header file bp1750.h.

If you have time, try changing the number of filter coefficients by including a different .h file, e.g. lp33.h. The time taken to compute each output sample value will depend on the number of filter coefficients used.

Factors influencing the time taken to compute each output sample from an FIR filter include, in the case of the CMSIS functions, the sizes of the blocks of data processed.

## CONCLUSIONS

This laboratory exercise has introduced the FIR filter and explored several different methods of measuring its characteristics in the time and frequency domains.