***DSP Education Kit***

**LAB 03**

**Finite Impulse Response (FIR) Filters**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc29195002)

[1.1 Lab overview 1](#_Toc29195003)

[2 Requirements 1](#_Toc29195004)

[3 The Moving Average Filter 1](#_Toc29195005)

[4 Frequency Response of the Moving Average Filter 2](#_Toc29195006)

[5 Observation of Frequency Response Using a Sinusoidal Input Signal 2](#_Toc29195007)

[5.1 Exercise 4](#_Toc29195008)

[6 Observation of Frequency Response Using a Pseudorandom Input Signal 5](#_Toc29195009)

[6.1 Exercise 5](#_Toc29195010)

[7 OPTIONAL: Identification of Magnitude Frequency Response Using Two Discovery Boards 7](#_Toc29195011)

[7.1 Exercise 8](#_Toc29195012)

[8 Identification of Magnitude Frequency Response Using One Discovery Board 9](#_Toc29195013)

[8.1 Exercise 9](#_Toc29195014)

[9 Higher-Order Moving Average Filters 10](#_Toc29195015)

[9.1 Exercise 10](#_Toc29195016)

[10 Finite Impulse Response (FIR) Filters 11](#_Toc29195017)

[10.1 Exercise 11](#_Toc29195018)

[11 FIR Filter Programs with Coefficients Specified in Separate Header File 13](#_Toc29195019)

[12 Generating FIR Filter Coefficient Header Files Using MATLAB 13](#_Toc29195020)

[13 Frequency Responses of Simple FIR Filters 17](#_Toc29195021)

[13.1 Exercise 17](#_Toc29195022)

[14 Designing a High Pass Filter Using the Window Method 19](#_Toc29195023)

[14.1 Exercise 19](#_Toc29195024)

[15 Execution Time of Coded FIR Filters 19](#_Toc29195025)

[15.1 Exercise 20](#_Toc29195026)

[16 Conclusions 21](#_Toc29195027)

[17 Additional References 21](#_Toc29195028)

# Introduction

## Lab overview

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

# Requirements

To carry out this lab, you will need:

* An STM32F746G Discovery board
* A PC running Keil MDK-Arm
* MATLAB
* GoldWave
* An oscilloscope
* 3.5 mm audio jack
* An audio frequency signal generator
* Optional: External microphone, although you can also use the microphones on the board
* Optional: additional STM Discovery board and audio jack

# The Moving Average Filter

The Moving Average filter is widely used in DSP and is arguably 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 following equation:

where *x*(*n*) represents the *n*th sample of an input signal and *y*(*n*) is the *n*th sample of the filter output, 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 stm32f7\_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 sound output to the LINE IN socket on the Discovery board. Alternatively, you could play the test signal using a smartphone. Run program stm32f7\_average\_intr.c on the Discovery board and use headphones connected to HEADPHONE OUT 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 subsequent exercises in this lab manual 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 Discovery board and connect the (left channel of the) HEADPHONE OUT socket either to an oscilloscope or to the input of the soundcard on a PC running *GoldWave*.



Figure 1: Connection diagram for measuring the magnitude frequency response of the five-point moving average filter implemented by program astm32verage\_intr.c using a signal generator and an oscilloscope

Figures 2 and 3 show the output from program stm32f7\_average\_intr.c displayed using the Fast Fourier Transform (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. The trace shown in Figure 2 might be plotted as a single point on the sketch axes at 2.0 kHz and −20 dB, and Figure 3 might be plotted as a single point at 1.3 kHz and −30 dB. However, the calibration of the oscilloscope and of the PC soundcard will not be the same, and measurements from the two cannot be combined. The absolute level, in dBs, read from either is not important (and depends, in any case, on the amplitude of the input signal). What is important is its *relative* level across the range of frequencies measured using one or other method.

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.

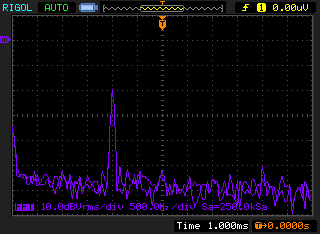


Figure 2: Output signal from program stm32f7\_average\_intr.c viewed using FFT function of Rigol DS1052 oscilloscope. It’s difficult to read, but the text at the bottom of the screen indicates 500 Hz/div and 10 dBVrms/div. The marker on the left-hand side of the display indicates a level of 0 dBVrms

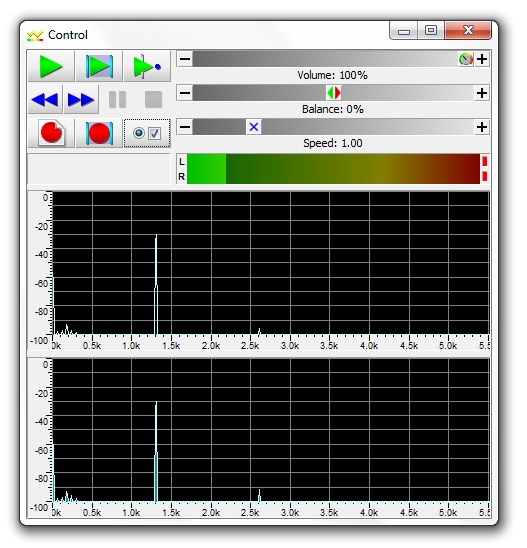


Figure 3: Output signal from program stm32f7\_average\_intr.c viewed using spectrum display in GoldWave

## Exercise

Run program stm32f7\_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 different frequencies, on the axes below.

Sketch of the magnitude frequency response of five point moving average filter implemented by program stm32f7\_average\_intr.c and measured using sinusoidal input signals:



# Observation of Frequency Response Using a Pseudorandom Input Signal

Alternatively, an indication of the magnitude frequency response of the filter may be obtained by applying a pseudorandom input signal, containing equally weighted components at all frequencies, to the filter and observing the spectral content of the filter output. Program stm32f7\_average\_prbs\_intr.c combines the moving average filter of program stm32f7\_average\_intr.c with a pseudorandom noise generator (implemented within the program using function prbs()).

## Exercise

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 below and verify that the results indicate a similar magnitude frequency response to that measured using a signal generator and plotted in the previous task.

**Note**: Using pseudorandom noise as a test signal is a quick and easy method of obtaining an indication of the magnitude frequency response of a filter.

Sketch of magnitude frequency response of five point moving average filter measured using program stm32f7\_average\_prbs\_intr.c:



# OPTIONAL: Identification of Magnitude Frequency Response Using Two Discovery Boards

In Lab 2, program stm32f7\_sysid\_CMSIS\_intr.c was used to identify the characteristics of the antialiasing and reconstruction filters of the WM8994 codec. Here, the same program is used to identify the characteristics of the moving average filter.

Connect two Discovery boards as shown in Figure 4. On one of the boards, run program stm32f7\_average\_intr.c (once this has been downloaded into flash memory, the program will run when power is applied to the board via the USB cable), and on the other, run program stm32f7\_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 stm32f7\_sysid\_CMSIS\_intr.c works.



Figure 4: Connection diagram for identification of magnitude frequency response of five point moving average filter using program stm32f7\_sysid\_CMSIS\_intr.c

As the program runs, the adaptive filter coefficients and the magnitude of their FFT are shown on the LCD. These graphs represent the impulse and magnitude frequency responses of the signal path from A to B in Figure 4, including the moving average filter.

After program stm32f7\_sysid\_CMSIS\_intr.c has run for several seconds, halt the program and save the values of the 256 filter coefficients firCoeffs32 to a .dat file, as you did in Lab 2.

Then, use MATLAB function stm32f7\_logfft.m to display the adaptive filter coefficients firCoeffs32 and the magnitude of their FFT. Figure 5 shows the result plotted on the same axes as the theoretical magnitude frequency response of the five-point moving average filter.



Figure 5: Magnitude frequency response of the five point moving average filter implemented using program **stm32f7\_average\_intr.c** on one STM32F746G Discovery board, identified using program **stm32f7\_sysid\_CMSIS\_intr.c** running on a second board (blue line), and plotted on the same axes as its theoretical magnitude frequency response (dashed red line)

## Exercise

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

# Identification of Magnitude Frequency Response Using One Discovery Board

By just using one STM32F746G Discovery board, you can also perform an experiment similar to the previous one by implementing the moving average filter before the pseudorandom noise is written to the DAC.

Connect HEADPHONE OUT on the audio card to LINE IN, as shown in Figure 6, and build and load program stm32f7\_sysid\_average\_CMSIS\_intr.c. As in the previous example, run the program for a few seconds and follow the same procedure as before to save the adaptive filter coefficients to a file and plot them using MATLAB.



Figure 6: Connection diagram for identification of magnitude frequency response of five point moving average filter using program stm32f7\_sysid\_average\_CMSIS\_intr.c

## Exercise

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

# Higher-Order Moving Average Filters

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

Modify either program stm32f7\_average\_prbs\_intr.c or program stm32f7\_sysid\_average\_CMSIS\_intr.c so that N= 11 and observe the frequency response of the eleven-point moving average filter using:

* either the FFT function of an oscilloscope or using*GoldWave* in the case of program stm32f7\_average\_prbs\_intr.c or
* on the LCD and/or using MATLAB in the case of program stm32f7\_sysid\_average\_CMSIS\_intr.c.

Record your observations in the exercise below.

## Exercise

1. Sketch of magnitude frequency response of eleven-point moving average filter:



# Finite Impulse Response (FIR) Filters

What would happen if the values of the filter coefficients h(n) were changed? To answer this question, modify either program stm32f7\_average\_prbs\_intr.c or program stm32f7\_sysid\_average\_CMSIS\_intr.c again so that N= 5 and h(n) = {0.0833, 0.2500, 0.3333, 0.2500, 0.0833} (comment out the program statement in function main() that assigns values to array h).

## Exercise

1. Observe the frequency response of the filter and record your observations on the axes below.

You should find that higher frequencies are attenuated more than they were by the five-point moving average filter and that the “notches” in the frequency response of the five-point moving average filter 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.

Magnitude frequency response of five-point moving average filter with Hann window:



1. **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 below and sketch the magnitude of that theoretical frequency response on the axes.

Derivation of theoretical frequency response of five-point moving average filter with Hann window using DTFT:

Theoretical magnitude frequency response of five-point moving average filter with Hann window:



# FIR Filter Programs with Coefficients Specified in Separate Header File

Programs stm32f7\_fir\_intr.c, stm32f7\_fir\_prbs\_intr.c and stm32f7\_sysid\_fir\_CMSIS\_intr.c implement FIR filters for which the filter coefficients *h*(*n*) are not specified within these source files but are read from a separate header file. To change the characteristics of the FIR filters implemented by these programs, simply change the preprocessor command

#include “maf5.h”

to, for example,

#include “lp33.h”

and *Rebuild* the project.

**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};

Several different example coefficient files have been provided. Investigate the characteristics of one or two of these using program stm32f7\_fir\_prbs\_intr.c or program stm32f7\_sysid\_fir\_CMSIS\_intr.c and observe the filtered noise signal output at HEADPHONE OUT using an oscilloscope or *GoldWave* the graphs on the LCD.

# 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, 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.

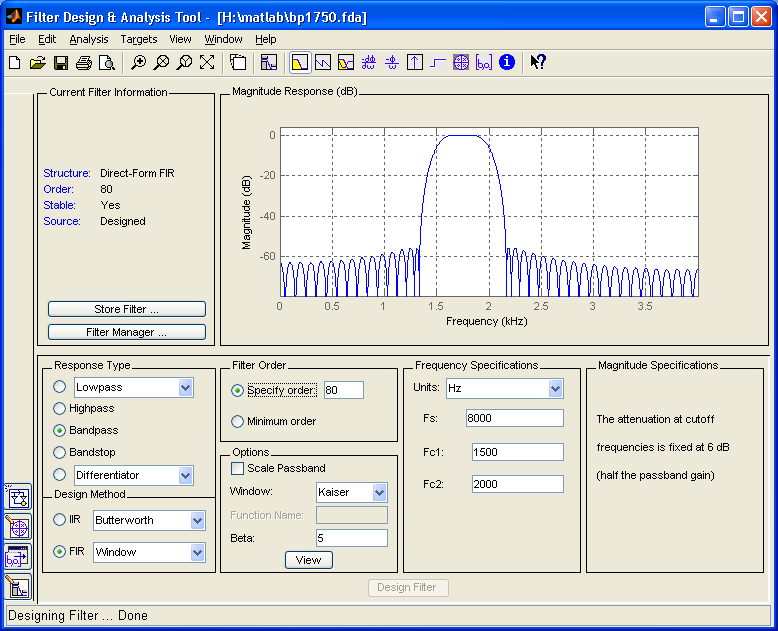


Figure 7: Design of a bandpass FIR filter using MATLAB **fdatool**

Coefficient header file bp1750.h was generated using MATLAB function stm32f7\_fir\_coeffs()after designing the filter using fdatool (as shown in Figure 7).

Incorporate these filter coefficients into program stm32f7\_sysid\_fir\_CMSIS\_intr.c. Ensure that your board connection is as shown in Figure 8. Run the program, save the values of the 256 adaptive filter coefficients firCoeffs32 to a file, and plot them using MATLAB.

****

Figure 8: Connection diagram for program stm32f7\_sysid\_fir\_CMSIS\_intr.c

The expected results are shown in Figure 9 and Figure 10.

****

Figure 9: Experimentally measured impulse response and magnitude frequency response corresponding to filter coefficients defined in header file bp1750.h, derived using program stm32f7\_sysid\_fir\_CMSIS\_intr

****

Figure 10: The LCD while program stm32f7\_sysid\_fir\_CMSIS\_intr.c is running, using header file bp1750.h

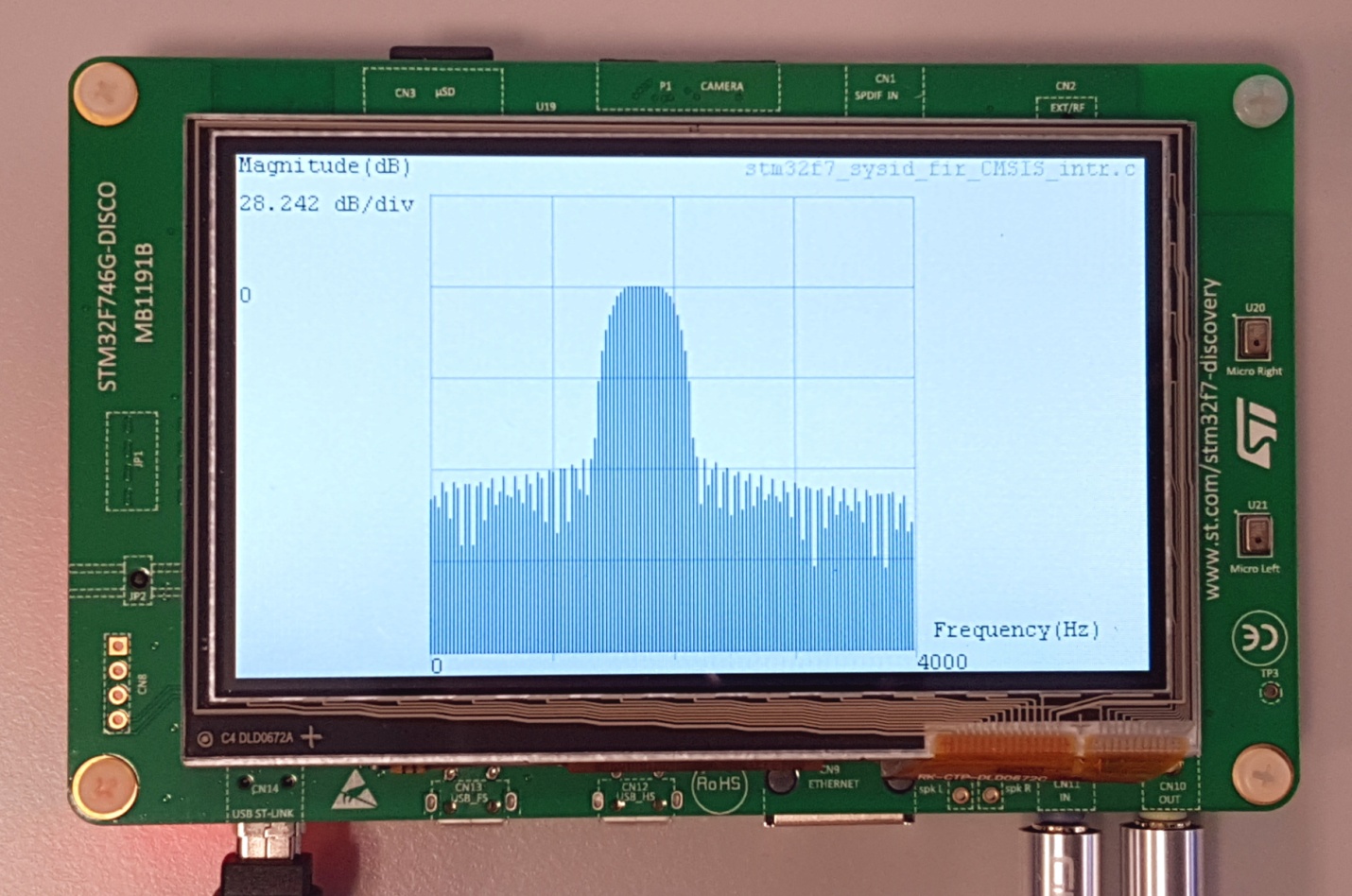


Figure 11: View of LCD while program stm32f7\_sysid\_fir\_CMSIS\_intr.c is running, using header file bp1750.h

# Frequency Responses of Simple FIR Filters

## Exercise

Using the techniques introduced in this lab, 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 the axes below.

Filter X:



Filter Y:



Filter Z:

# Designing a High Pass Filter Using the Window Method

## Exercise

Create a coefficient file myhpf.h for use with programs stm32f7\_fir\_intr.c and stm32f7\_fir\_prbs\_intr.c that will implement a high pass filter at sample frequency 8000 Hz, 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 13.

If you use MATLAB to create a vector of coefficient values, you can generate a header file using the function stm32f7\_fir\_coeffs().

Compare the theoretical magnitude frequency response of your design with its magnitude frequency response measured using programs stm32f7\_fir\_prbs\_intr.c and stm32f7\_sysid\_fir\_intr.c.

# Execution Time of Coded FIR Filters

In the programs used so far, e.g., stm32f7\_fir\_prbs\_intr.c, the FIR filter has been implemented relatively straightforwardly. The following experiment compares the efficiency of that straightforward implementation with the optimized CMSIS DSP library function arm\_fir\_f32().

The time taken to implement the filtering operation may be assessed by making GPIO pin PI1 high just before calculating each output sample value and making that pin low again just after the calculation. The state of GPIO pin PI1 can be viewed in real time using an oscilloscope.

The following program statements can be found in the interrupt service routine BSP\_AUDIO\_SAI\_Interrupt\_CallBack().

BSP\_LED\_On(LED1);

and

BSP\_LED\_Off(LED1);

Make sure that coefficient file bp1750.h is used by the program (#include “bp1750.h”).

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

## Exercise

In this exercise, you will measure and record the corresponding duration/computation time in Table 1.

1. Run program stm32f7\_fir\_prbs\_intr.c. Connect an oscilloscope probe to GPIO pin PI1 (see figure 1) on the Discovery board and measure the length of time that it is high. **Record the time taken to compute one output sample value in Table 1**.
2. Replace source file stm32f7\_fir\_prbs\_intr.c with file stm32f7\_fir\_prbs\_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 the duration of the positive voltage pulses on GPIO pin PI1 in Table 1.**
3. Computing one output sample value at a time is not the most efficient way to use function arm\_fir\_f32()that is optimized for block processing. Program stm32f7\_fir\_prbs\_CMSIS\_dma.c can be used to illustrate this. Replace program stm32f7\_fir\_prbs\_CMSIS\_intr.c with program stm32f7\_fir\_prbs\_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() that computes PING\_PONG\_BUFFER\_SIZE output samples each time it is called. PING\_PONG\_BUFFER\_SIZE is defined in header file stm32f7\_wm8994\_init.h and its default value is 256. Divide the time taken by each call to function process\_buffer() by PING\_PONG\_BUFFER\_SIZE to get a time to compare with the previous results.

**Record measured computation times in Table 1.**

Recall that the maximum time available for computation in programs stm32f7\_fir\_prbs\_intr.c and stm32f7\_fir\_prbs\_CMSIS\_intr.c at a sampling rate of 8 kHz is 125 µs. The maximum time available in programs stm32f7\_fir\_prbs\_dma.c and stm32f7\_fir\_prbs\_CMSIS\_dma.c, i.e., the time between consecutive calls to function process\_buffer(), will be PING\_PONG\_BUFFER\_SIZE/(fs)= 32 ms.

|  |  |  |
| --- | --- | --- |
| Program | using GPIO PD13 | divided by BUFSIZE |
| fir\_prbs\_intr.c |  | N/A |
| fir\_prbs\_dma.c |  |  |
| fir\_prbs\_CMSIS\_intr.c |  | N/A |
| fir\_prbs\_CMSIS\_dma.c |  |  |

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

OPTIONAL: Try changing the number of filter coefficients by including a different .h file, e.g., lp33.h and test again. The time taken to compute each output sample value will depend on the number of filter coefficients used.

# Conclusions

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

# Additional References

**Moving average filters:**

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