***DSP Education Kit***

**LAB 2 INSTRUCTOR VERSION**

**Sampling, Aliasing, and Reconstruction**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc28871059)

[1.1 Lab overview 1](#_Toc28871060)

[2 Requirements 1](#_Toc28871061)

[2.1.1 WM8994 codec 2](#_Toc28871062)

[3 Sampling and Aliasing–Generating Sinusoids of Arbitrary Frequency 3](#_Toc28871063)

[3.1 Exercise 6](#_Toc28871064)

[3.2 Observations 7](#_Toc28871065)

[3.3 Computational costs 8](#_Toc28871066)

[3.3.1 Exercise 8](#_Toc28871067)

[4 Square Wave Generation Using the WM8994 Codec 9](#_Toc28871068)

[4.1 Exercise 9](#_Toc28871069)

[5 Step and Impulse Responses of the WM8994 DAC 10](#_Toc28871070)

[5.1 Exercise 10](#_Toc28871071)

[5.2 Observations 11](#_Toc28871072)

[6 Magnitude Frequency Response of the WM8994 DAC 12](#_Toc28871073)

[6.1 Exercise 13](#_Toc28871074)

[6.2 Observations 14](#_Toc28871075)

[7 Time Domain Response of the WM8731 Antialiasing Filter 14](#_Toc28871076)

[8 Plotting Sample Values Read from the WM8994 ADC Using MATLAB 16](#_Toc28871077)

[9 Magnitude Frequency Response of the WM8994 Antialiasing Filter 17](#_Toc28871078)

[10 Comparison of the WM8994 DAC and the STM32F746G 12-BIT DAC 18](#_Toc28871079)

[11 Emulation of the Low Pass Filtering Characteristics of the WM8994 DAC Using the 12-Bit DAC 23](#_Toc28871080)

[11.1 Using IIR filter 23](#_Toc28871081)

[*11.2* Using FIR filter 25](#_Toc28871082)

[12 Frequency Responses of the IIR and FIR Interpolation Filters 28](#_Toc28871083)

[13 Estimating WM8994 Codec Bandwidth Using an Adaptive Filter 31](#_Toc28871084)

[13.1 Exercise 33](#_Toc28871085)

[13.2 Observations 34](#_Toc28871086)

[14 OPTIONAL: Estimating WM8994 Codec Bandwidth Using Two STM32F746G Discovery Boards 35](#_Toc28871087)

[14.1 Exercise 37](#_Toc28871088)

[15 Special Features of the WM8994 Codec 37](#_Toc28871089)

[16 Conclusions 38](#_Toc28871090)

# Introduction

## Lab overview

The examples in this exercise concern the characteristics of the WM8994 codec (analogue to digital converter (ADC) and digital to analogue converter (DAC)) used on the STM32F746G Discovery board.

In this lab, we will:

- Examine the effect of sampling rate on the bandwidth of a digital signal processing system.

- Demonstrate the phenomenon of aliasing.

- Compare the reconstruction (of an analogue output signal from a sequence of discrete sample

values) characteristics of the WM8994 DAC with those of the 12-bit DAC built into the

STM32F746G microcontroller.

Instructor note: Text in red and images in red boxes are for instructors only.

# Requirements

To carry out this lab, you will need:

* An STM32F746G Discovery board
* A PC running Keil MDK-Arm
* MATLAB
* GoldWave
* An oscilloscope
* Two 3.5 mm audio jacks
* Optional: An audio frequency signal generator
* Optional: An additional STM32F746G Discovery board for two-board experiment.

### WM8994 codec

The STM32F746G Discovery board features a Cirrus Logic WM8994 stereo audio codec, which is accessed via I2C for control and I2S, using the STM32F746G microcontroller’s serial audio interface (SAI) peripheral, for data. Analogue input and output signals are accessible via two three-pole (TRS) 3.5 mm jack sockets (LINE IN (CN11) and HEADPHONE OUT(CN10)) as shown in Figure 1.

As configured for these exercises, the WM8994 converts an analogue input signal into 16-bit signed integer sample values and the DAC converts 16-bit signed integer sample values into an analogue output signal.

Additionally, the WM8994 has a digital microphone interface, and the STM32F746G Discovery board provides two MEMS microphones as input devices.

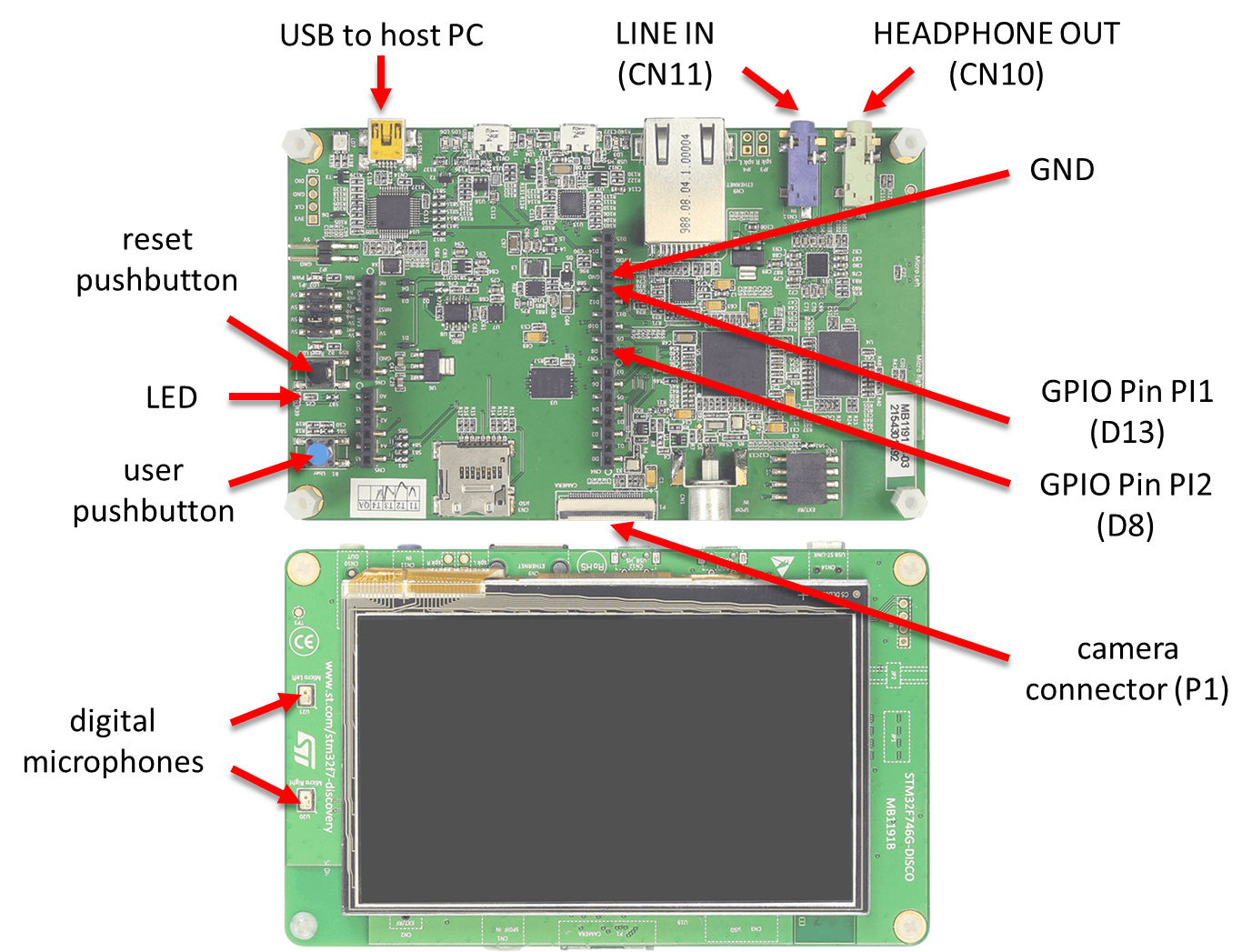


Figure 1: STM32F746G Discovery board

# Sampling and Aliasing–Generating Sinusoids of Arbitrary Frequency

Program stm32f7\_sine\_intr.c, shown in the code snippet below, generates sinusoidal analogue output waveforms via the WM8994 codec using calls to the function arm\_sin\_f32() to compute each sample value. The program uses interrupt-based I/O, and its sampling rate is set to 8 kHz. The frequency of the sinusoidal analogue output waveform is controlled by the value assigned to the variable sine\_frequency, and its amplitude is controlled by the value assigned to the variable amplitude.

// stm32f7\_sine\_intr.c.

#include "stm32f7\_wm8994\_init.h"

#include "stm32f7\_display.h"

#define SOURCE\_FILE\_NAME "stm32f7\_sine\_intr.c"

#define SAMPLING\_FREQ 8000

extern int16\_t rx\_sample\_L;

extern int16\_t rx\_sample\_R;

extern int16\_t tx\_sample\_L;

extern int16\_t tx\_sample\_R;

float32\_t sine\_frequency = 1000.0;

float32\_t amplitude = 10000.0;

float32\_t theta\_increment;

float32\_t theta = 0.0;

void BSP\_AUDIO\_SAI\_Interrupt\_CallBack()

{

// when we arrive at this interrupt service routine (callback)

// the most recent input sample values are (already) in global variables

// rx\_sample\_L and rx\_sample\_R

// this routine should write new output sample values in

// global variables tx\_sample\_L and tx\_sample\_R

theta\_increment = 2\*PI\*sine\_frequency/SAMPLING\_FREQ;

theta += theta\_increment;

if (theta > 2\*PI) theta -= 2\*PI;

BSP\_LED\_On(LED1);

// tx\_sample\_L = (int16\_t)(amplitude\*sin(theta));

// tx\_sample\_L = (int16\_t)(amplitude\*sinf(theta));

tx\_sample\_L = (int16\_t)(amplitude\*arm\_sin\_f32(theta));

tx\_sample\_R = tx\_sample\_L;

BSP\_LED\_Off(LED1);

plotSamplesIntr(tx\_sample\_L, 32);

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,

GRAPH);

while(1){}

}

When you run the program, you should see a start screen appear on the LCD as shown in Figure 2. Press the blue user pushbutton to continue, and you should see on the LCD a graphical representation of the first 32 discrete sample values written to the DAC. Figure 3 shows the display corresponding to an output signal frequency of 367 Hz (sine\_frequency = 367.0).

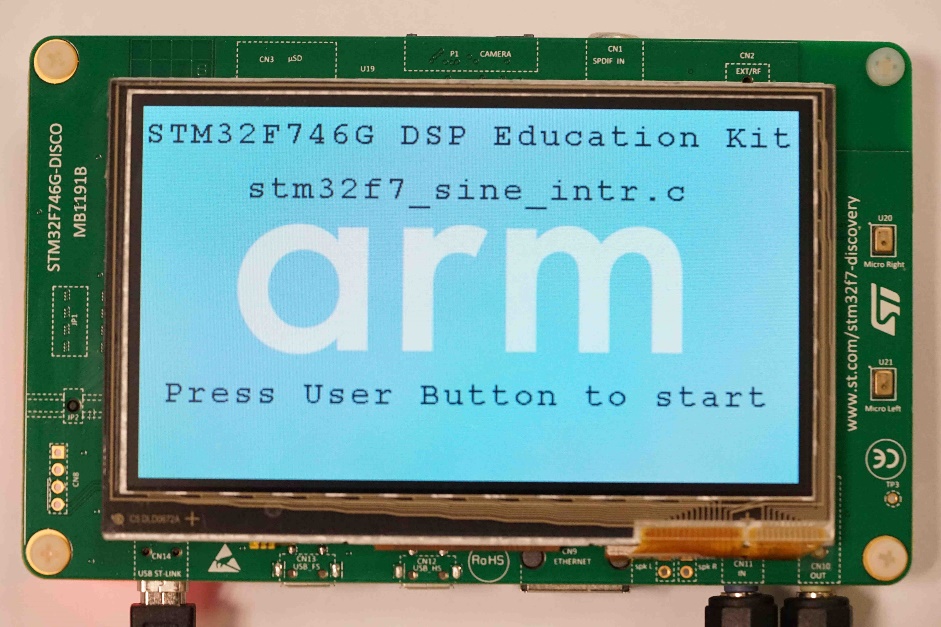


Figure 2: Start screen for program stm32f7\_sine\_intr.c

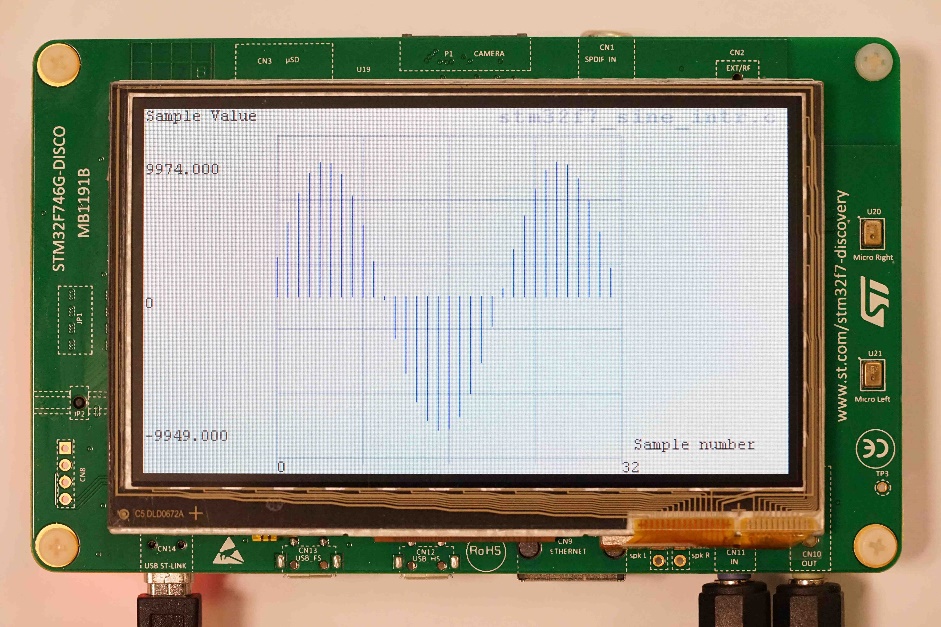


Figure 3: Graphical representation of the first 32 sample values output by program stm32f7\_sine\_intr.c with value of variable sine\_frequency equal to 367.0

Connect either channel of the HEADPHONE OUT output on the Discovery board to an oscilloscope and verify, using both time-domain and frequency-domain oscilloscope displays, that the output signal from the program *as supplied* is a 1000 Hz sinusoid (sine\_frequency = 1000.0).

At each sampling instant, the program statements in the interrupt service routine function BSP\_AUDIO\_SAI\_Interrupt\_CallBack()are executed. The value of the variable theta is incremented by 2\*PI\*sine\_frequency/SAMPLING\_FREQ, and a sample value equal to the sine of that angle (theta) is computed using function arm\_sin\_f32() and written to the DAC. The DAC *reconstructs* a continuous-time signal from the discrete-time sample values that are written to it, and, in this case, the reconstructed analogue output signal is a sinusoidal waveform as shown in Figure 4.

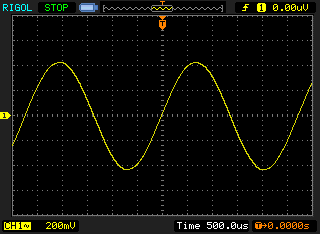


Figure 4: Analogue output waveform generated using program stm32f7\_sine\_intr.c with a value of variable sine\_frequency equal to 367.0

## Exercise

1. Change the value of the variable sine\_frequency in program stm32f7\_sine\_intr.c according to Table 1 and record the frequencies of the output signals you observe on an oscilloscope connected to HEADPHONE OUT.

|  |  |
| --- | --- |
| Value assigned to variable sine\_frequency | Frequency of analogue output signal (Hz) |
| 367 | 367 |
| 2573 | 2573 |
| 7000 | - |
| 3500 | 3500 |
| 4500 | - |

Table 1: Frequency of analogue output signal for different values of variable frequency in program stm32f7\_sine\_intr.c

## Observations

Each time the program is run, the LCD will display the first 32 sample values written to the DAC as shown, for example, in Figure 3. Your results should be consistent with the following two observations.

1. Using program stm32f7\_sine\_intr.c, it is possible to generate continuous-time sinusoidal waveforms of arbitrary frequency, for example, 2573 Hz. The program makes it far easier to change the frequency of the waveform than was the case using pre-computed sample values in program stm32f7\_sine\_lut\_intr.c in Laboratory Manual #1.

Note that while the table lookup method of generating sinusoidal waveforms used in program stm32f7\_sine\_lut\_intr.c appeared inflexible, it is quite common to generate sinusoidal (or other) waveforms from lookup tables by stepping through their contents but not necessarily one sample at a time. This is illustrated in program stm32f7\_sweep\_intr.c. There is a fundamental weakness in program stm32f7\_sine\_intr.c, which is that computing each output sample value by calling function sin() is computationally very expensive. In general, it is not a good idea to make calls to trigonometric functions in standard math libraries from real-time programs.

1. Using a sampling rate of 8 kHz, the WM8994 DAC is incapable of generating a sinusoidal output signal with a frequency greater than 4 kHz. More generally, the WM8994 DAC is incapable of generating a sinusoidal output signal with a frequency greater than half its sampling frequency.

Note that the output sample values computed by the program are consistent with the value of the variable frequency. There are no computational errors in the program.

The fact that sample values computed using a value of sine\_frequency greater than half the sampling rate result in an analogue output waveform with a frequency that is less than half the sampling rate is an example of *aliasing*.

You might also have observed that for analogue output signal frequencies greater than around 1 kHz, it is difficult to make much sense of the graph of sample values shown on the LCD. Nonetheless, each of the sample sequences displayed corresponds to a near perfectly sinusoidal analogue output signal.

## Computational costs

You can compare the computational costs of using alternative function calls sin(), arm\_sin\_f32(), and sinf() by commenting out those not used (and rebuilding the project) and using an oscilloscope to measure the duration of the pulses on GPIO pin PI1. GPIO pin PI1 is set using program statement:

BSP\_LED\_On(LED1);

, and reset using program statement

BSP\_LED\_Off(LED1);

in program stm32f7\_sine\_intr.c.

### Exercise

Record these pulse durations in Table 2 and compare with the duration of the corresponding pulses generated by program stm32f7\_sine\_lut\_intr.c in Lab Manual #1. The durations of the pulses correspond to the execution times of the different methods of determining the value of the next output sample. The pulses are repeated at the sampling rate (8 kHz), that is, at intervals of 0.125 ms.

|  |  |
| --- | --- |
| Function or program | GPIO pin PI1 pulse duration |
| Arm\_sin\_f32() | 550 ns |
| sinf() | 600 ns |
| sin() | 19 µs |
| stm32f7\_sine\_lut\_intr.c | 150 ns |

Table 2: GPIO pin PI1 pulse durations measured using programs stm32f7\_sine\_intr.c and stm32f7\_sine\_\_lut\_intr.c

**Note**: In fact, the lookup table method of generating sinusoidal waveforms of arbitrary frequency is widely used. Program stm32f7\_sweep\_intr.c illustrates this using a lookup table containing 8000 samples of one cycle of a sinusoid and varying the incremental steps taken through the table to generate a range of different frequency sinusoids. **Run program stm32f7\_sweep\_intr.c and listen to the analogue output signal using headphones.**

# Square Wave Generation Using the WM8994 Codec

Consider the sample sequence:

{10000, 10000, 10000, 10000, -10000, -10000, -10000, -10000}

These are samples of a square wave with a period of eight samples, that is, a period of 1 ms and a frequency of 1 kHz (at a sampling rate of 8 kHz).

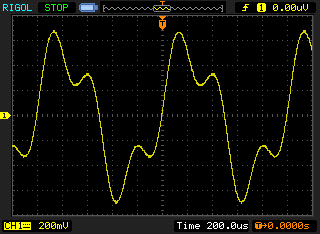
Edit program stm32f7\_sine\_lut\_intr.c, replacing the sample values representing a sinusoid in array sine\_table with the sequence given above. That is, replace the relevant line in the source file with:

sine\_table[LOOPLENGTH] = {10000, 10000, 10000, 10000, -10000, -10000, -10000, -10000};

**Run the program and sketch the resultant output waveform (viewed using an oscilloscope) on the axes below.**

## Exercise

1. Run program stm32f7\_sine\_lut\_intr.c after the modification mentioned above and sketch the analogue output waveform observed in the oscilloscope.



1. How would you describe the time-domain representation of the analogue output?

The waveform appears to be the sum of a 1 kHz sinusoid and a smaller amplitude 3 kHz sinusoid.

1. Is it a square wave?

It is not a square wave. The output should obviously be a sinusoidal wave (sine).

If you have an oscilloscope capable of representing signals in the frequency-domain, using this gives a useful insight into the nature of the analogue output signal.

# Step and Impulse Responses of the WM8994 DAC

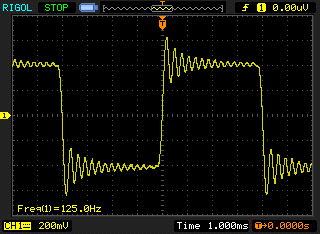
Program stm32f7\_square\_intr.c repeatedly outputs a data sequence comprising 32 consecutive values of 10000 followed by 32 consecutive values of -10000. Run the program and examine the analogue output waveform using an oscilloscope connected to either channel of HEADPHONE OUT. What you are seeing might be interpreted as a 125 Hz square wave signal that has been passed through the digital to analogue converter (reconstruction filter) in the WM8994 codec (although strictly speaking, and subtly differently, it is *samples* of a 125 Hz square wave signal that have been passed through the digital to analogue converter). On the Discovery board’s LCD, you should see a graphical representation of the sample values written to the DAC.

The graphical representations of signals on the LCD deliberately show each sample value as a stick or bar in order to emphasize their discrete nature.

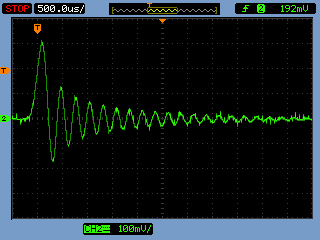
## Exercise

1. After running the program as mentioned above, sketch the analogue output waveform seen on the oscilloscope on the axes below and, from this, deduce the impulse response of the reconstruction filter.

The analogue output waveform reveals the step response of the DAC. That step response is oscillatory and lightly damped. An impulse is the time derivative of a step, and, correspondingly, the impulse response of a system is equal to the time derivative of its step response. The impulse response of the DAC will be oscillatory and lightly damped and decays to zero. The peaks of the step response will correspond to zero crossings in the impulse response.



1. Sketch what you think the impulse response of the DAC MIGHT BE on the axes below and explain how you deduced this in the space below that figure (think of each transition in the square wave as a step)



You can view the *actual* impulse response of the WM8994 DAC using program stm32f7\_dimpulse\_intr.c, or you can use the d/dt function on an oscilloscope to differentiate the waveform generated using program stm32f7\_square\_intr.c and see a sequence of alternately positive and negative impulse responses.

## Observations

The analogue output waveform generated by program stm32f7\_square\_intr.c contains frequency components up to a maximum frequency of 4 kHz. Higher frequency components (that would make the edges of the analogue output waveform sharper) are missing. This may be illustrated using either the Fast Fourier Transform (FFT) function of an oscilloscope or the spectrum display in *GoldWave*. In Figure 5, it is apparent that output from the DAC is negligible at frequencies greater than 4 kHz. This may be considered as an insight into the magnitude frequency response of the DAC.

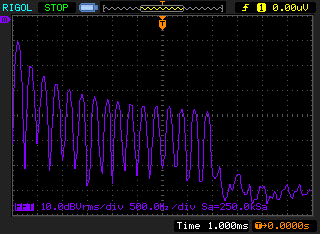


Figure 5: Magnitudes of the frequency components present in the analogue waveform generated using program stm32f7\_square\_intr.c

# Magnitude Frequency Response of the WM8994 DAC

You can get a further idea of the magnitude frequency response of the WM8994 DAC using program stm32f7\_prbs\_intr.c. This program uses function prbs() to generate a pseudorandom binary sequence, which contains a complete range of different frequency components at equal magnitudes. When this sequence is written to the DAC, the frequency content of the reconstructed analogue output signal reflects the frequency response of the DAC.

Run the program and look at the analogue output signal using the FFT function of an oscilloscope or using *GoldWave*. You may have to adjust the input volume level in *GoldWave* in order to see the frequency content of the signal to best effect.

On the LCD, you should see a graphical representation of the first 128 sample values being written to the DAC. A pseudorandom binary sequence is one example of a signal having a power spectral density that is constant.

Function prand() generates pseudorandom sample values using the Park-Miller algorithm (a random number generator).

Replace the program statement:

tx\_sample\_L = prbs(8000);

With program statement:

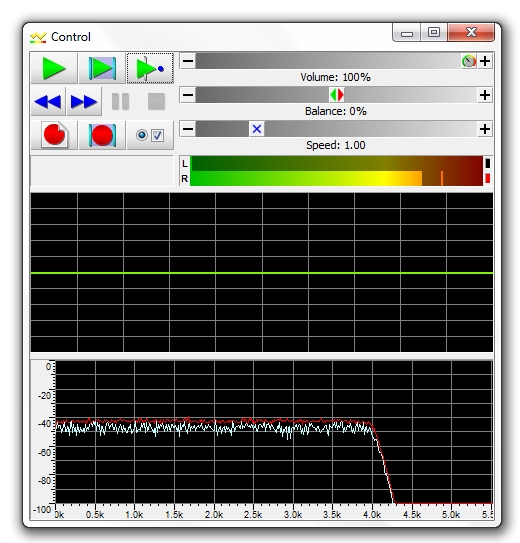
tx\_sample\_L = prand();

in program stm32f7\_prbs\_intr.c and build and run the program again. The frequency content of the analogue output signal viewed using an oscilloscope FFT function or *GoldWave* should be similar to that seen before and, once again, indicative of the frequency response of the WM8994 DAC.

In terms of its analogue output signal, program stm32f7\_prbs\_dma.c is similar to program stm32f7\_prbs\_intr.c but repeatedly updates the graph shown on the LCD (using the sample values generated by function prbs()) and outputs two different forms of pseudorandom signal (generated using functions prbs() and prand()) on the right and left channels of HEADPHONE OUT.

## Exercise

1. Sketch the magnitude frequency response you observe using either program on the axes below.



1. Run program stm32f7\_prbs\_intr.c again, having changed the sampling frequency to 48 kHz (as shown below).

stm32f7\_wm8994\_init(**AUDIO\_FREQUENCY\_48K**,

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\_0DB,

GRAPH);

1. Note in the space provided below the bandwidth of the noise signal generated:

**BANDWIDTH of analogue output signal generated using program stm32f7\_prbs\_intr.c with sampling RATE 48 kHz = around 5k**

## Observations

We have demonstrated in several different ways that **the WM8994 DAC cannot generate signal components having frequencies greater than half the sampling frequency**. This is true no matter howwe produce output sample sequences. The DAC is a low-pass filter. It follows that it is inadvisable to allow analogue input signal components having frequencies greater than half the sampling frequency to be sampled. This can be achieved by passing analogue input signals through a low-pass *antialiasing* filter prior to sampling by the ADC. An oversampling digital antialiasing filter with characteristics similar to those of the reconstruction filter in the DAC is built into the ADC in the WM8994 codec.

The subsequent examples in this lab manual illustrate the presence, and characteristics, of the antialiasing filter that precedes the ADC in the input signal path in the WM8994.

# Time Domain Response of the WM8731 Antialiasing Filter

In order to investigate the step response of the antialiasing filter in the WM8994, connect a signal generator to the left channel of the LINE IN socket. Adjust the signal generator to give a square wave output with a frequency of 125 Hz and an amplitude of approximately 1.2 V. Run program stm32f7\_loop\_buf\_dma.c, noting that the output signal from HEADPHONE OUT is not a perfect square wave. The input samples read from the ADC are plotted on the LCD as shown in Figure 6, and the analogue input and output signals are shown in the oscilloscope trace of Figure 7. The variations in the values of the input samples read from the ADC are due to the low-pass characteristic of the antialiasing filter.



Figure 6: Sample values read from WM8994 ADC using program stm32f7\_loop\_buf\_dma.c. Analogue input signal is a square wave with frequency 125 Hz

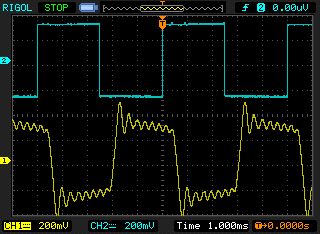
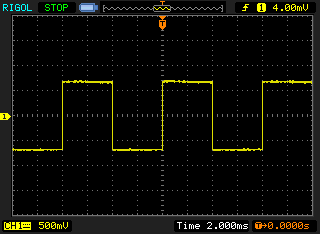


Figure 7: Analogue input and output waveforms observed using program stm32f7\_loop\_buf\_dma.c. Analogue input signal is a square wave with frequency 125 Hz

The action of program stm32f7\_loop\_buf\_dma.c is represented graphically in Figure 8.



Continuous voltage waveform

observable using oscilloscope

Discrete sample values

observable on LCD

ADC

Discrete-time

Continuous-time

WM8994

Figure 8: Graphical representation of the action of program stm32f7\_loop\_buf\_dma.c

# Plotting Sample Values Read from the WM8994 ADC Using MATLAB

To save the program output into a file and view them in Matlab, follow these steps:

1. Run the program **stm32f7\_loop\_buf\_dma.c** and press the user button to start the program.
2. Halt it by clicking on the ***Stop*** toolbar button in the MDK\_Arm debugger.
3. Type the variable name **lbuffer** as the ***Address*** in the debugger’s ***Memory 1*** window. Right click on the ***Memory 1***window and set the displayed data type to ***Decimal*** and ***Float****.*
4. To view the 128 most recent input sample values read from the ADC using MATLAB, save these to a file, using the command

**SAVE <filename> <start address>, <end address>**

where start address is that of array lbuffer and end address is equal to (start address + 0×200)

1. Launch Matlab and plot the contents of that file using the MATLAB function **stm32f7\_bar\_real()**.

You should see something similar to the display shown in Figure 9, and this should be similar to the graph plotted on the LCD. While the LCD is a useful and quick way of viewing the input samples, importing saved data into MATLAB enables more detailed analysis.



Figure 9: Input sample values read from the ADC using program stm32f7\_loop\_buf\_dma.c when the analogue input signal is a 125 Hz square wave

# Magnitude Frequency Response of the WM8994 Antialiasing Filter

The low pass characteristic of the WM8994 antialiasing filter can be demonstrated in a slightly different way using program stm32f7\_loop\_buf\_dma.c. Configure the signal generator to give a sinusoidal output. Run program stm32f7\_loop\_buf\_dma.c and look at the sample values displayed on the LCD. Increase the frequency of the sinusoidal input signal. You should find that for input frequencies greater than 4 kHz (half the sampling frequency), the sample values tend to zero. The antialiasing filter is blocking sinusoidal input signals with frequencies greater than half the sampling frequency.

# Comparison of the WM8994 DAC and the STM32F746G 12-BIT DAC

It’s insightful to compare the characteristics of the DAC in the WM8994 codec with those of one of the 12-bit DACs built into the STM32F746G microcontroller. The 12-bit DAC might be described as a zero-order hold DAC. When a sample is written to it, it outputs a voltage proportional to the value of that sample and maintains that output voltage until another sample is written to it.

Program stm32f7\_sine\_lut\_DAC12\_intr.c is similar to program stm32f7\_sine\_lut\_intr.c except that in addition to writing the eight different pre-computed sample values in array sine\_table[] to the WM8994 DAC, it also writes them (suitably scaled and shifted) to the 12-bit DAC.

The output of the 12-bit zero-order hold DAC is accessible via pin 17 on connector P1 on the STM32F746G Discovery board. Using an oscilloscope connected to that pin, you should see an analogue output waveform similar to that shown in Figure 10.

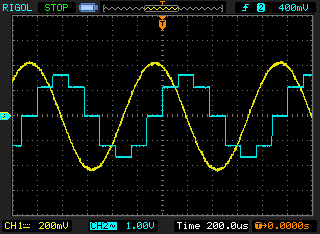


Figure 10: Analogue output waveform generated using program stm32f7\_sine\_lut\_DAC12\_intr.c (blue: 12-bit DAC, yellow: WM8994 DAC)

Programs stm32f7\_square\_DAC12\_intr.c, stm32f7\_prbs\_DAC12\_intr.c, and stm32f7\_dimpulse\_DAC12\_intr.c all write sequences of sample values to both the WM8994 and 12-bit DACs, and their analogue output signals may be compared using an oscilloscope.

The outputs are shown below:

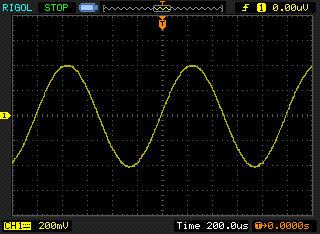


Figure 11: Analogue output waveform generated by WM8994 DAC using program stm32f7\_sine\_lut\_intr.c

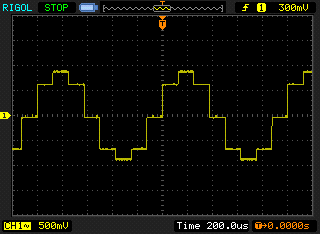


Figure 12: Analogue output waveform generated by 12-bit DAC using program stm32f7\_sine\_lut\_DAC12\_intr.c

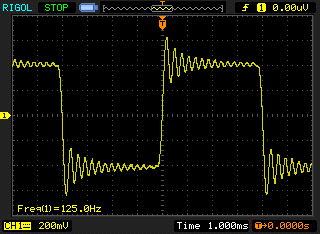


Figure 13: Analogue output waveform generated by WM8994 DAC using program stm32f7\_square\_intr.c

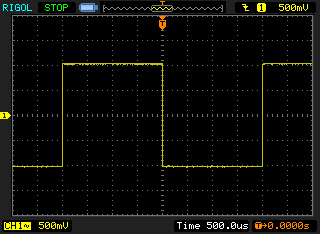


Figure 14: Analogue output waveform generated by 12-bit DAC using program stm32f7\_square\_DAC12\_intr.c

While it might be argued that the 12-bit DAC is better than the WM8994 DAC at generating a square wave, overall and certainly for audio signals, the characteristics of the WM8994 DAC are preferable.

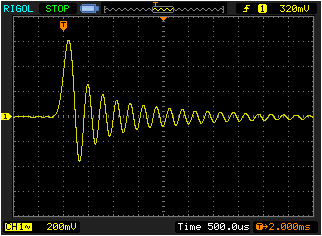


Figure 15: Analogue output waveform generated by WM8994 DAC using program stm32f7\_dimpulse\_intr.c

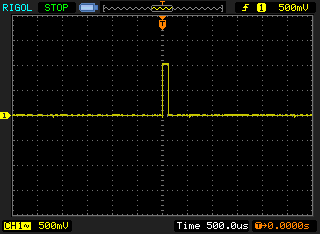


Figure 16: Analogue output waveform generated by 12-bit DAC using program stm32f7\_dimpulse\_DAC12\_intr.c

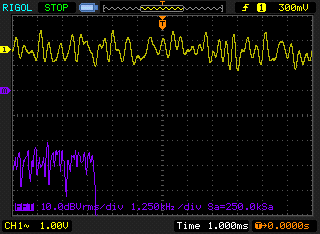


Figure 17: Analogue output waveform generated by WM8994 DAC using program stm32f7\_prbs\_DAC12\_intr.c

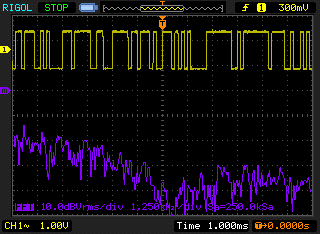


Figure 18: Analogue output waveform generated by 12-bit DAC using program stm32f7\_prbs\_DAC12\_intr.c

The analogue output signals generated by program stm32f7\_prbs\_DAC12\_intr.c shown in Figure 17 and Figure 18 illustrate clearly that while the magnitude frequency response of the WM8994 codec is close to that of an ideal low pass filter with a cut-off frequency of fs/2, the 12-bit DAC has a far less pronounced low pass magnitude frequency response. Indeed, since the impulse response of the 12-bit DAC is a rectangular pulse, as shown in Figure 16, then its magnitude frequency response is anticipated to be equal to the magnitude of a *Dirichlet* or *aliased sinc* function.

# Emulation of the Low Pass Filtering Characteristics of the WM8994 DAC Using the 12-Bit DAC

## Using IIR filter

The WM8994 DAC achieves its near-ideal low pass filtering characteristics using a combination of *up-sampling* and *digital filtering*. These processes are emulated by program stm32f7\_sine\_lut\_recon48\_IIR\_intr.c. This program generates an analogue output signal using the 12-bit DAC based on eight pre-computed sample values representing one cycle of a sinusoid. These sample values are accessed at a rate of 8 kHz. However, sample values that are written to the 12-bit DAC is at the higher rate of 48 kHz. At that higher sampling rate, the pre-computed sample values are used only at every sixth sampling instant. The program uses an IIR digital filter to interpolate between those pre-computed sample values, that is, to provide five out of every six sample values written to the 12-bit DAC at 48 kHz.

The resultant analogue output waveform is shown in Figure 19. You do not need to have an in-depth understanding of digital filtering in order to simply observe similarities between the performance of the 12-bit DAC and the WM8994 DAC in this section.

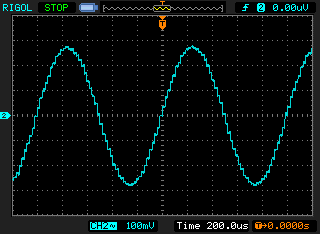


Figure 19: Analogue output waveform generated by 12-bit DAC using program stm32f7\_sine\_lut\_recon48\_IIR\_intr.c

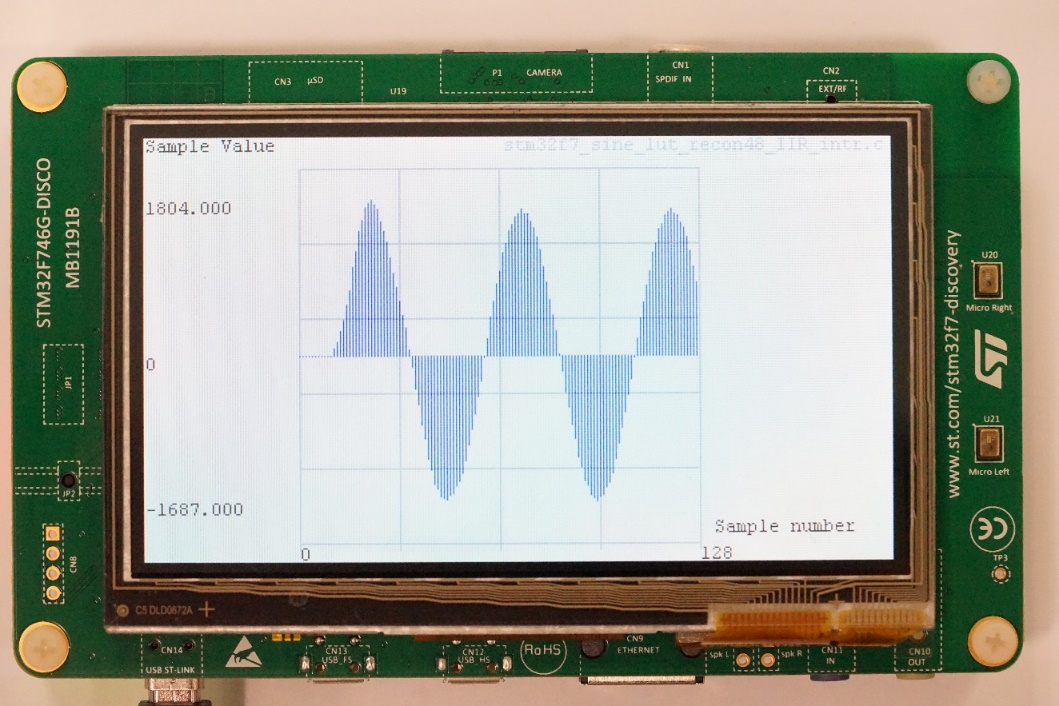


Figure 20: Graphical representation of the first 128 sample values written to the 12-bit DAC by program stm32f7\_sine\_lut\_recon48\_IIR\_intr.c

Program stm32f7\_sine\_lut\_recon48\_IIR\_intr.c does not emulate the WM8994 nor does it use the same method of interpolation. But it is intended to give an insight into how it achieves its performance. Shown on the LCD by program stm32f7\_sine\_lut\_recon48\_IIR\_intr.c are the first 128 sample values written to the 12-bit DAC.

Remember when looking at Figure 11 and Figure 12 that they were generated using only 8 sample values per cycle of the 1 kHz sinusoidal signal.

6

fs = 8 kHz

fs = 48 kHz

LPF

Up-sample

Interpolate

DAC

STM32F746

12-bit DAC

Figure 21: Graphical representation of the operation of program stm32f7\_sine\_lut\_recon48\_IIR\_intr.c

## Using FIR filter

Program stm32f7\_sine\_lut\_recon48\_FIR\_intr.c uses an FIR, rather than an IIR filter to interpolate between sample values. The output waveform shown in Figure 22 is almost indistinguishable from that shown in Figure 19. However, the LCD graphs shown in Figure 20 and Figure 23 reveal a significant difference between the transient (time-domain) behaviors of the two filters. This aspect of interpolation filter performance is illustrated explicitly in Figure 24 and Figure 25, generated using programs stm32f7\_dimpulse\_recon48\_IIR\_intr.c and stm32f7\_dimpulse\_recon48\_FIR\_intr.c, respectively.



Figure 22: Analogue output waveform generated by 12-bit DAC using program stm32f7\_sine\_lut\_recon48\_FIR\_intr.c

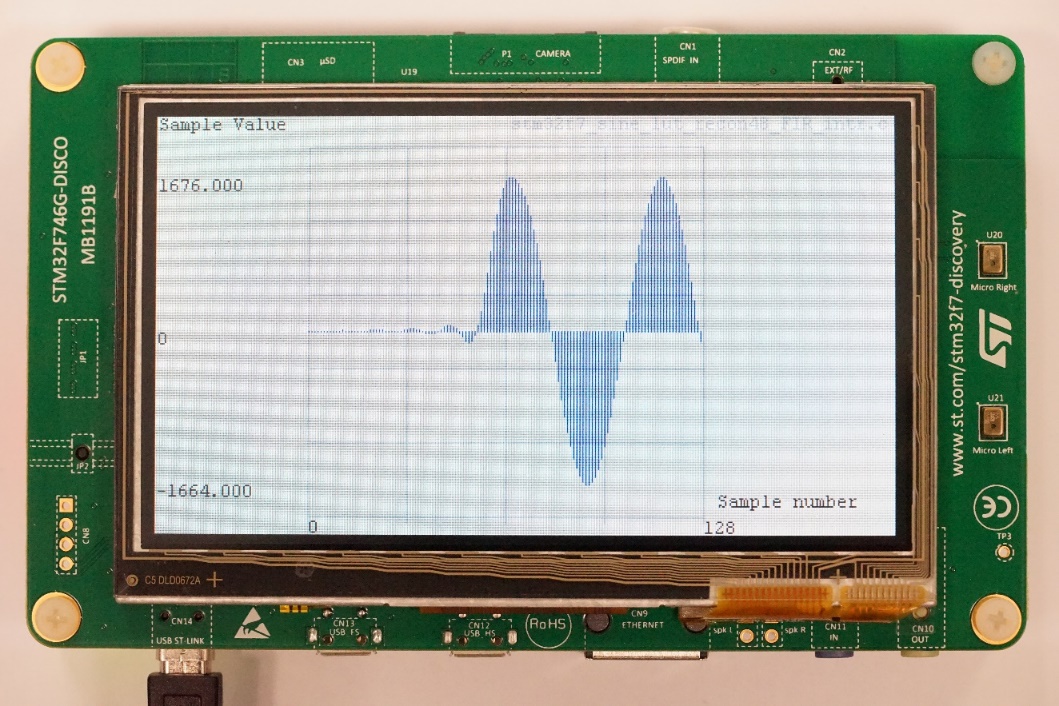


Figure 23: Graphical representation of the first 128 sample values written to the 12-bit DAC by program stm32f7\_sine\_lut\_recon48\_FIR\_intr.c

The lower trace (channel 2) in Figure 24 and Figure 25 shows the impulse response of the interpolation filters (in combination with the zero-order hold characteristic of the 12-bit DAC) and the upper trace (channel 1) shows the state of GPIO pin PI1, which changes at the same time as a single non-zero sample value is input to the filter. The IIR interpolation filter exhibits less *latency* (the time delay between the non-zero value being input to the filter and the peak of its impulse response) than the FIR interpolation filter. This is a desirable property in an audio codec.

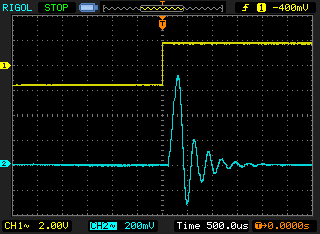


Figure 24: Analogue output waveform generated by 12-bit DAC using program stm32f7\_dimpulse\_recon48\_IIR\_intr.c and signal output on GPIO pin PI1, which indicates time at which single non-zero sample value was written to IIR filter

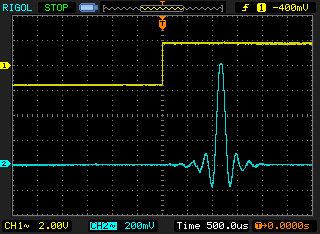


Figure 25: Analogue output waveform generated by 12-bit DAC using program stm32f7\_dimpulse\_recon48\_FIR\_intr.c and signal output on GPIO pin PI1, which indicates time at which single non-zero sample value was written to FIR filter

Figure 26 and Figure 27 show analogue output waveforms generated using programs stm32f7\_square\_recon48\_IIR\_intr.c and stm32f7\_square\_recon48\_FIR\_intr.c, respectively. Comparing these with the analogue output waveform generated using the WM8994 DAC with program stm32f7\_square\_intr.c, it seems plausible to suggest that the WM8994 DAC incorporates an IIR interpolation filter. Some other audio codecs, for example, the Cirrus Logic WM8731, would generate analogue output waveforms closer to that shown in Figure 21|.

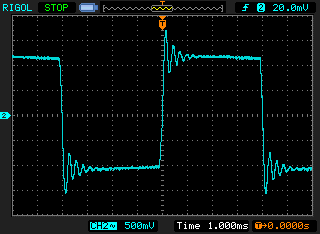


Figure 26: Analogue output waveform generated by 12-bit DAC using program stm32f7\_square\_recon48\_IIR\_intr.c

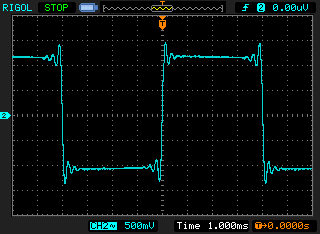


Figure 27: Analogue output waveform generated by 12-bit DAC using program stm32f7\_square\_recon48\_FIR\_intr.c

# Frequency Responses of the IIR and FIR Interpolation Filters

Even though the time-domain responses of the IIR and FIR interpolation filters are quite distinct, the magnitude frequency responses suggested by the output waveforms generated by programs stm32f7\_prbs\_recon48\_IIR\_intr.c. and stm32f7\_prbs\_recon48\_IIR\_intr.c. and shown in Figure 28 and Figure 29 are quite similar.

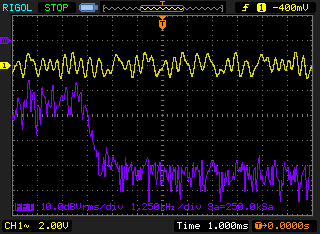


Figure 28: Analogue output waveform generated by 12-bit DAC using program stm32f7\_prbs\_recon48\_IIR\_intr.c

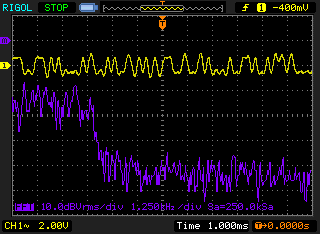


Figure 29: Analogue output waveform generated by 12-bit DAC using program stm32f7\_prbs\_recon48\_FIR\_intr.c

Compare these with the plot of Figure 30, which uses a WM8994 DAC. The roll-off of the magnitude frequency response of the WM8994 DAC at 4 kHz is significantly sharper than that of the 12-bit DAC with oversampling.

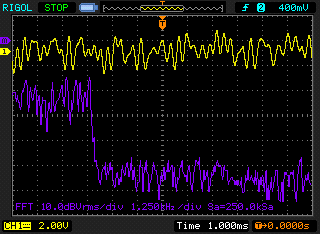


Figure 30: Analogue output waveform generated by WM8994 DAC using program stm32f7\_prbs\_intr.c

# Estimating WM8994 Codec Bandwidth Using an Adaptive Filter

Yet another way of illustrating the limited bandwidth of the codec is to measure its magnitude frequency response using program stm32f7\_sysid\_CMSIS\_intr.c. You need not understand exactly how this program works in order to use it. Effectively, it identifies the characteristics of the path between its discrete-time output and its discrete-time input (points A and B in Figure 31) using an adaptive FIR filter.

Connect HEADPHONE OUT to LINE IN on the audio card using a 3.5 mm jack to 3.5 mm jack cable. The signal path from A to B that will be identified by program stm32f7\_sysid\_CMSIS\_intr.c comprises the series combination of the digital to analogue and analogue to digital converters.



Figure 31: Connection diagram for WM8994 codec bandwidth identification using program stm32f7\_sysid\_CMSIS\_intr.c

Once the program is running, you can toggle between graphs on the LCD of:

1. The adaptive filter coefficients, stored in array firCoeffs32, which correspond to the impulse response of the identified signal path, and
2. The magnitude of the FFT of the adaptive filter coefficient that corresponds to the magnitude frequency response of the identified signal path.

Each graph is updated periodically, allowing the gradual convergence of the adaptive filter to be observed. The learning rate, BETA of the adaptive filter in this program, has deliberately been set very low.

The changing values of the adaptive filter coefficients may also be viewed in a memory window in *MDK-Arm* as shown in Figure 34.

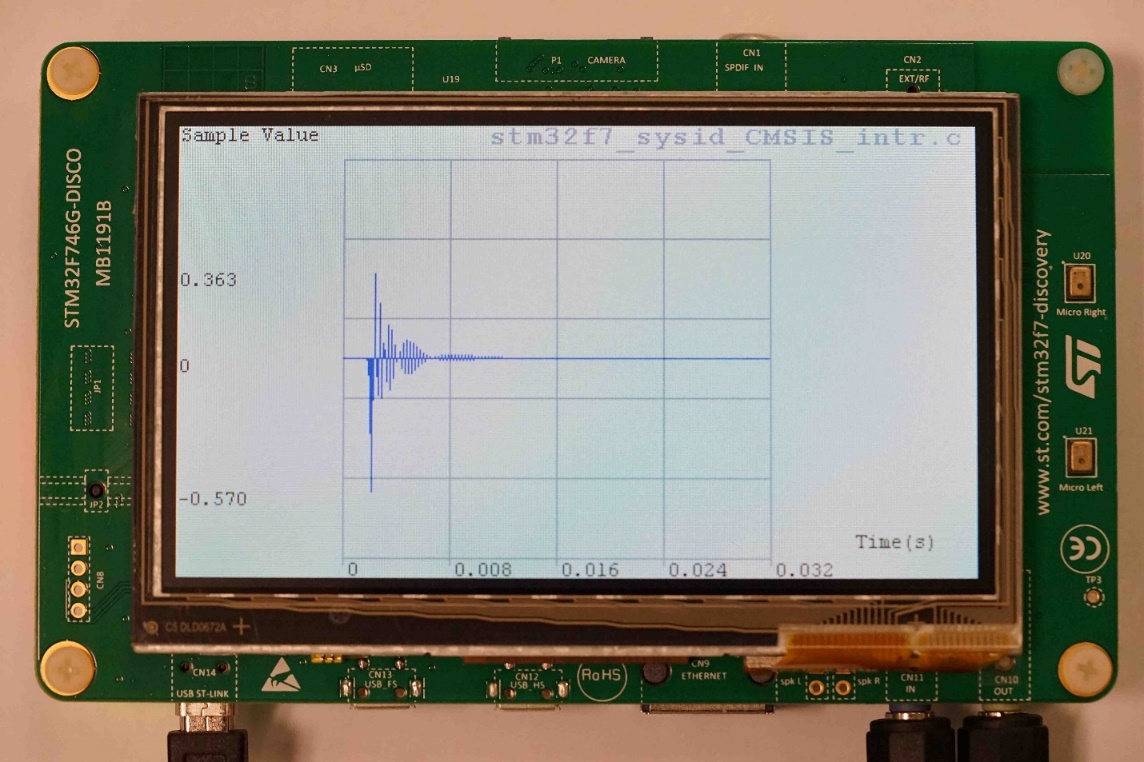


Figure 32: Adaptive filter coefficients displayed by program stm32f7\_sysid\_CMSIS\_intr.c

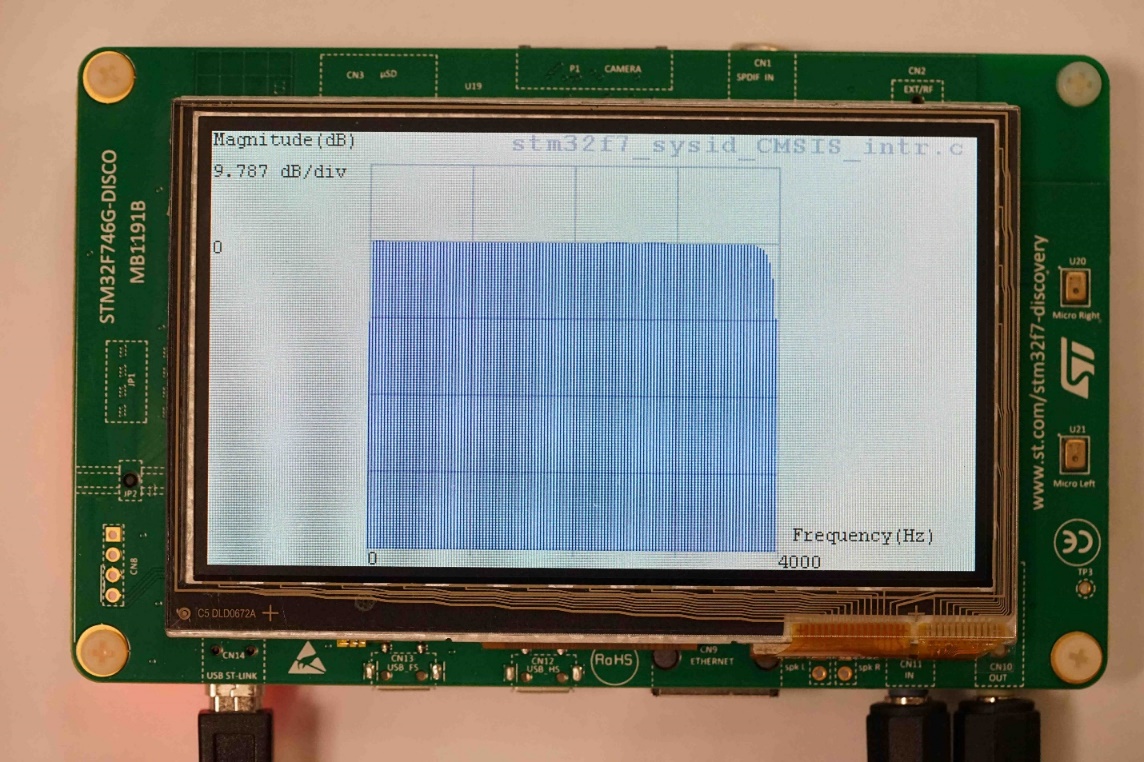


Figure 33: Magnitude of FFT of adaptive filter coefficients displayed by program stm32f7\_sysid\_CMSIS\_intr.c

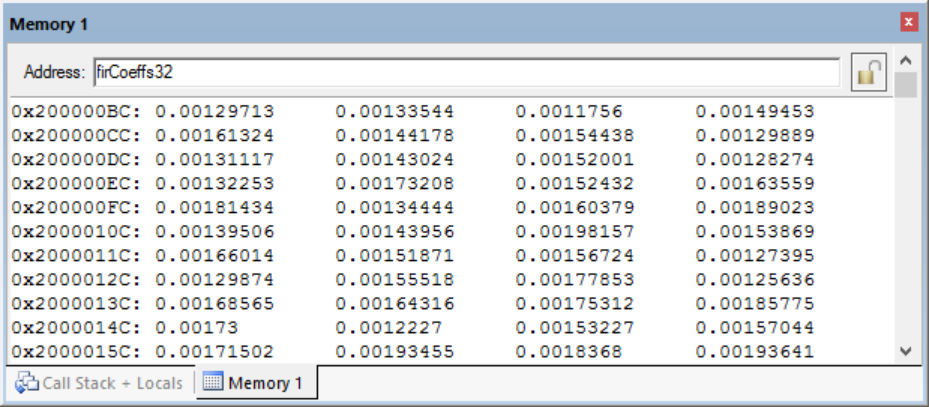


Figure 34: Connection diagram for WM8994 codec bandwidth identification using program stm32f7\_sysid\_CMSIS\_intr.c

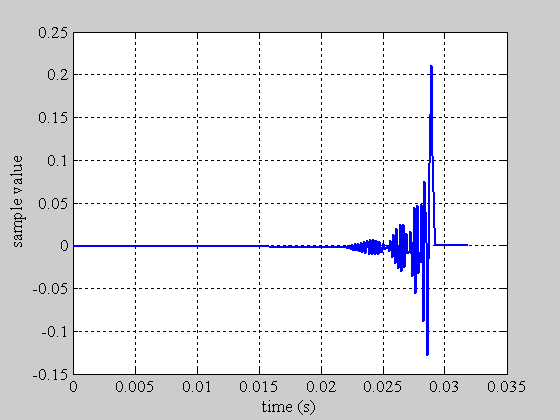
## Exercise

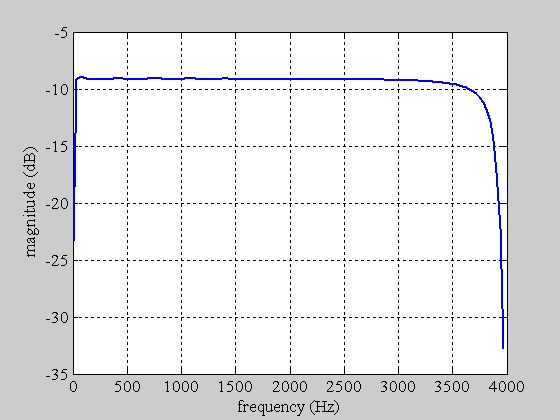
Halt the program after a few seconds and save the values of the adaptive filter coefficients firCoeffs32 to a file using the command

**SAVE <filename.dat> <start address>, <end address>**

where start address is that of array firCoeffs32, and end address is equal to (start address + 0×400), and plot the contents of that file using the MATLAB function stm32f7\_logfft(). The order in which the filter coefficients are stored in array firCoeffs32 is time-reversed.

1. Sketch the magnitude frequency response that you observe, either on the LCD or in MATLAB, on the axes below, adding a scale to the y-axis:





## Observations

The frequency range over which the magnitude frequency response has been identified is equal to half the sampling rate of the codec. To observe the frequency response of the codec beyond half its sampling frequency, we will identify the characteristics of one system with a sampling rate of 8 kHz using a second system with a sampling rate of 16 kHz.

# OPTIONAL: Estimating WM8994 Codec Bandwidth Using Two STM32F746G Discovery Boards

If possible, you can also use two STM32F746G Discovery boards to estimate the WM8994 codec bandwidth by following these steps:

1. Connect two STM32F746G Discovery boards together as shown in Figure 35, that is, connect LINE IN on one board to HEADPHONE OUT on the other and vice versa. The sampling rate used by program stm32f7\_sysid\_CMSIS\_intr.c is determined by one of the parameters passed to function stm32f7\_wm8994\_init().



Figure 35: Connection diagram for WM8994 codec bandwidth identification using program stm32f7\_sysid\_CMSIS\_intr.c and two STM32F746G Discovery boards

1. In program stm32f7\_sysid\_CMSIS\_intr.c, change the program statement

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\_0DB,

SOURCE\_FILE\_NAME,

GRAPH);

to

stm32f7\_wm8994\_init(**AUDIO\_FREQUENCY\_16K**,

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\_0DB,

SOURCE\_FILE\_NAME,

GRAPH);

1. In program stm32f7\_loop\_intr.c, change the program statement

stm32f7\_wm8994\_init(**AUDIO\_FREQUENCY\_48K**,

IO\_METHOD\_INTR,

**INPUT\_DEVICE\_DIGITAL MICROPHONE\_2**,

OUTPUT\_DEVICE\_HEADPHONE,

**WM8994\_HP\_OUT\_ANALOG\_GAIN\_6DB**,

WM8994\_LINE\_IN\_GAIN\_0DB,

WM8994\_DMIC\_GAIN\_17DB,

SOURCE\_FILE\_NAME,

NOGRAPH);

to

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\_0DB,

SOURCE\_FILE\_NAME,

NOGRAPH);

1. Make sure that program stm32f7\_loop\_intr.c (sampling rate 8 kHz) is running on one system before running program stm32f7\_sysid\_CMSIS\_intr.c (sampling rate 16 kHz) on the other.
2. After running and halting program stm32f7\_sysid\_CMSIS\_intr.c, save the values of the 256 adaptive filter coefficients firCoeffs32 to a file using the command

**SAVE <filename> <start address>, <end address>**

## Exercise

Plot the impulse and magnitude frequency responses using MATLAB function stm32f7\_logfft()and sketch the magnitude frequency response on the axes of figure below. Magnitude frequency response observed using program stm32f7\_sysid\_CMSIS\_intr.c and two STM32F746G Discovery boards:



# Special Features of the WM8994 Codec

The WM8994 contains several configurable and programmable features that modify the signal path through the codec including parametric equalizer intended for speaker compensation, and selectable high pass and de-emphasis filters.

It’s beyond the scope of this laboratory exercise to explain these in detail, but you can un-comment some of the program statements in program stm32f7\_sysid\_CMSIS\_intr.c to observe the effect of these on the characteristics of the signal path through the codec.

# Conclusions

At the end of this lab exercise, you should have gained an awareness of the limited bandwidth of digital signal processing systems and of the importance and characteristics of antialiasing and reconstruction filters.