Lab Exercise #2:   
ANALOGUE INPUT AND OUTPUT

# Overview

The examples in this exercise concern the characteristics of the WM5102 codec (analogue to digital converter (ADC) and digital to analogue converter (DAC)) used on the Wolfson Pi Audio Card. The effect of sampling rate on the bandwidth of a digital signal processing system is examined and the phenomenon of aliasing is demonstrated.

# Details

## Hardware

To carry out this exercise you will need a Freescale FRDM-K64F, a Wolfson Pi Audio Card, an oscilloscope, an audio frequency signal generator, a PC running *Keil MDK-ARM*, *GoldWave* and M*ATLAB*, and suitable connecting cables.

### Wolfson Pi Audio Card

The *Wolfson Pi Audio Card* include the WM5102 stereo audio codec, which is accessed via I2C (for control) and I2S (for data) interfaces. Analogue input and output signals are accessible via three-pole 3.5mm jack sockets (LINE IN, LINE OUT, MIC\_IN and HEADPHONES\_OUT).

As configured for these exercises, the WM5102 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.

### SAMPLING and aliasing – generating sinusoids of arbitrary frequency

Consider program sine\_intr.c, listed in figure 1. This program generates sinusoidal analogue output waveforms via the WM5102 codec using calls to the function sin(). The program uses interrupt-based i/o and its sampling rate is set to 8 kHz.

1. // sine\_intr.c
2. #include "audio.h"
3. volatile int16\_t audio\_chR=0;
4. volatile int16\_t audio\_chL=0;
5. #define SAMPLING\_FREQ 8000
6. float32\_t frequency = 1000.0;
7. float32\_t amplitude = 2000.0;
8. float32\_t theta\_increment;
9. float32\_t theta = 0.0f;
10. void I2S\_HANDLER(void) {
11. audio\_IN = i2s\_rx();
12. audio\_chL = (audio\_IN & 0x0000FFFF);
13. audio\_chR = ((audio\_IN >>16)& 0x0000FFFF);
14. theta\_increment = 2\*PI\*frequency/SAMPLING\_FREQ;
15. theta += theta\_increment;
16. if (theta > 2\*PI) theta -= 2\*PI;
17. audio\_chL = (int16\_t)(amplitude\*sin(theta));
18. audio\_chR = audio\_chL;
20. audio\_OUT = ((audio\_chR<<16 & 0xFFFF0000)) + (audio\_chL & 0x0000FFFF);
21. i2s\_tx(audio\_OUT);
22. }
23. int main(void)
24. {
25. audio\_init ( hz8000, line\_in, intr, I2S\_HANDLER);
27. while(1){}
28. }

Figure 1: Listing of program sine\_intr.c.

Suppose that the output sinusoid frequency is set to 1 kHz (frequency = 1000.0). As described in exercise #1, once each sampling instant the program statements on lines 20 to 24 in interrupt service routine function I2S\_HANDLER() are executed. The value of the variable theta is updated and a sample value equal to the sine of that angle (theta) is computed and written to the DAC. The DAC *reconstructs* a continuous-time signal from the discrete-time sample values that are written to it. In this case, the reconstructed signal is a sinusoidal waveform with a frequency of 1 kHz, as shown in figure 2.

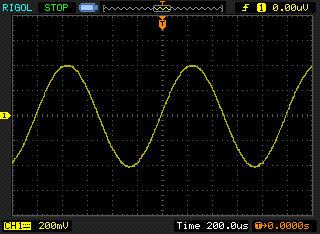


Figure 2: Analogue output waveform generated using program sine\_intr.c with value of variable frequency equal to 1000.0.

Change the value of the variable frequency in program sine\_intr.c according to table 1 and record the frequency of the output signal you observe on an oscilloscope connected to the right channel of LINE OUT on the audio card.

|  |  |
| --- | --- |
| Value assigned to variable frequency | Frequency of analogue output signal (Hz) |
| 1500 |  |
| 2573 |  |
| 7000 |  |
| 3500 |  |
| 4500 |  |

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

Your results should be consistent with the following two observations.

1. Using program sine\_intr.c it is possible to generate continuous-time sinusoidal waveforms of arbitrary frequency, e.g. 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 sine\_lut\_intr.c in exercise #1.
2. Using a sampling rate of 8 kHz, the WM5102 DAC is incapable of generating a sinusoidal output signal with a frequency greater than 4 kHz

### square wave generation using the WM5102 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 1ms, and a frequency of 1 kHz. Edit program 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 figure 3 space.

Sketch the analogue output waveform.

****

Figure 3: Analogue output waveform generated using program sine8\_intr.c using samples of a 1kHz square wave.

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

Is it a square wave?

Explain the shape of the output waveform?

### Step and impulse responses of the WM5102 DAC Reconstruction filter

Program 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 look closely at the output waveform using an oscilloscope. What you are seeing might be interpreted as a square wave signal that has been passed through the digital to analogue converter (reconstruction filter) in the WM5102 codec.

Sketch the analogue output waveform seen on the oscilloscope on the axes of figure 4 and, from this, deduce the impulse response of the reconstruction filter.

Sketch what you think the impulse response of the reconstruction filter is on the axes of figure 5 and explain how you deduced this in the space below that figure.



Figure 4: Analogue output waveform generated using program square\_intr.c.



Figure 5: Impulse response of WM5102 DAC reconstruction filter, deduced from waveform shown in figure 4.

You can view the *actual* impulse response of the WM5102 DAC using program dimpulse\_intr.c or by using the d/dt function on an oscilloscope to differentiate the waveform generated using program square\_intr.c.

The analogue output waveform generated by program square\_intr.c contains the frequency components of a 125 Hz square wave up to a maximum frequency of 4 kHz. Higher frequency components (that would make the edges of the square wave sharper) are missing. This may be illustrated using either the FFT function of an oscilloscope or the spectrum display in *GoldWave*. In figure 6, it is apparent that output from the DAC is negligible at frequencies greater than 4 kHz.

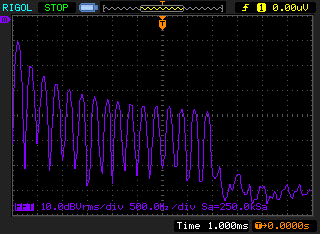


Figure 6: Magnitudes of the frequency components present in the analogue waveform generated using program square\_intr.c.

### Magnitude frequency response of the WM5102 DAC Reconstruction filter

You can get a further idea of the magnitude frequency response of the DAC using program prbs\_intr.c. This program uses function prbs() to generate a pseudo random binary sequence which, in theory, 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 reconstruction filter. 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.

Sketch the magnitude frequency response you observe on the axes of figure 7.



Figure 7: Magnitude frequency response of WM5102 DAC reconstruction filter demonstrated using program prbs\_intr.c.

Run program prbs\_intr.c again, having changed the sampling frequency to 48 kHz.

audio\_init(hz48000, line\_in, intr, I2S\_HANDLER);

and note the bandwidth of the noise signal generated.

So far we have seen that **the WM5102 DAC cannot generate signal components having frequencies greater than half the sampling frequency**. This is true no matter howwe produce output sample sequences. 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 in to the ADC in the WM5102 codec.

### STEP REsponse of the WM5102 antialiasing filter

In order to investigate the step response of the antialiasing filter in the WM5102, connect a signal generator to the left channel of the LINE IN socket. Adjust the signal generator to give a square wave output of frequency 200 Hz and amplitude 0.3 V. Run program loop\_buf\_intr.c, noting that the output signal from LINE OUT is not a perfect square wave. Halt the program. In order to view the 128 most recent input sample values read from the ADC, 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 + 0x200), and plot the contents of that file using the MATLAB function plot\_real(). You should see something similar to the display shown in figure 9. Figure 8 shows the square wave input signal that produced the display of figure 9 and also the corresponding analogue output signal. The variations in the values of the input samples read from the ADC are due to the low-pass characteristic of the antialiasing filter. Compare figure 9 with the step response of the reconstruction filter that you sketched in figure 5.

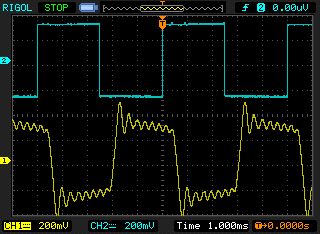


Figure 8: Analogue input and output waveforms observed using program loop\_buf\_intr.c.



Figure 9: Input sample values read from the ADC using program loop\_buf\_intr.c when the analogue input signal is a 200 Hz square wave.

### Magnitude frequency response of the WM5102 Antialiasing filter

The low pass characteristic of the WM5102 antialiasing filter can further be demonstrated using program loop\_buf\_intr.c. Adjust the signal generator to give a sinusoidal output. Run program loop\_buf\_intr.c for a few seconds, and plot the contents of array lbuffer (the 128 most recent input samples read from the ADC) using MATLAB just as in the previous example. Repeat this procedure for a number of different sinusoid frequencies. You should find that for frequencies above 4 kHz the output of the analogue to digital converter, stored in lbuffer, is effectively zero.

### Estimating WM5102 codec bandwidth using an adaptive filter

Another way of observing the limited bandwidth of the codec is to measure its magnitude frequency response using program sysid\_CMSIS\_intr.c. You need not understand exactly how program sysid\_CMSIS\_intr.c 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 10) using an adaptive FIR filter.

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

FRDM-K64F + Wolfson Pi Audio Card

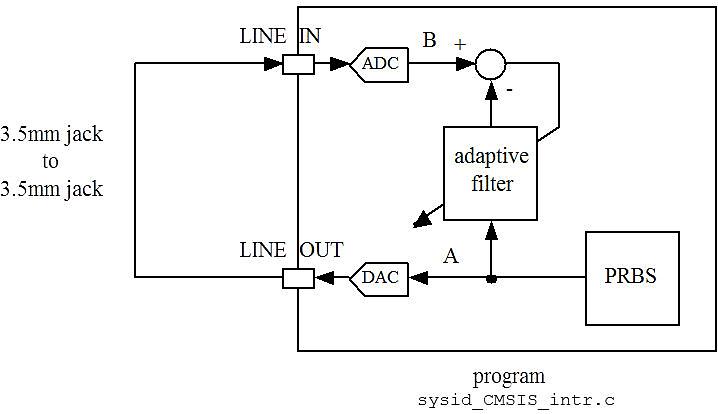


Figure 10: Connection diagram for WM5102 codec bandwidth identification using program sysid\_CMSIS\_intr.c.

Run program sysid\_CMSIS\_intr.c for several seconds and then halt it. Save the values of the adaptive filter coefficients firCoeffs32 to a file using the command

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

where start address is that of array firCoeffs32, and end address is equal to (start address + 0x400), and plot the contents of that file using the MATLAB function logfft(). Due to the order in which the filter coefficients are stored in memory, the impulse response plotted using MATLAB function logfft()is time-reversed, that is it runs from right to left in the MATLAB figure. Sketch the magnitude frequency response on the axes of figure 11.



Figure 11: *Magnitude* frequency response observed using program sysid\_CMSIS\_intr.c

The frequency range over which the magnitude frequency response has been identified is equal to half the sampling rate of the codec. In order to observe the frequency response of the codec beyond half its sampling frequency we will identify the characteristics of an audio card with a sampling rate of 8 kHz using a second system with a sampling rate of 32 kHz.

### Estimating WM5102 codec bandwidth using two audio cards

Connect two FRDM-K64F and Wolfson Pi Audio Card together as shown in Figure 12. Make sure that program loop\_intr.c (sampling rate 8 kHz) is running on one system before running program sysid\_CMSIS\_intr.c (sampling rate 16kHz) for a short time on the other. The sampling rate used by program sysid\_CMSIS\_intr.c is determined by one of the parameters passed to function audio\_init().

Change the statement

audio\_init ( hz8000, line\_in, intr, I2S\_HANDLER);;

in program sysid\_CMSIS\_intr.c to read

audio\_init ( hz16000, line\_in, intr, I2S\_HANDLER);

and change the sampling frequency in program loop\_intr.c to 8 kHz. After running and halting program 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>

Plot the saved values using MATLAB function logfft(). Sketch the magnitude frequency response on the axes of figure 13.

FRDM-K64F + Wolfson Pi Audio Card

FRDM-K64F + Wolfson Pi Audio Card

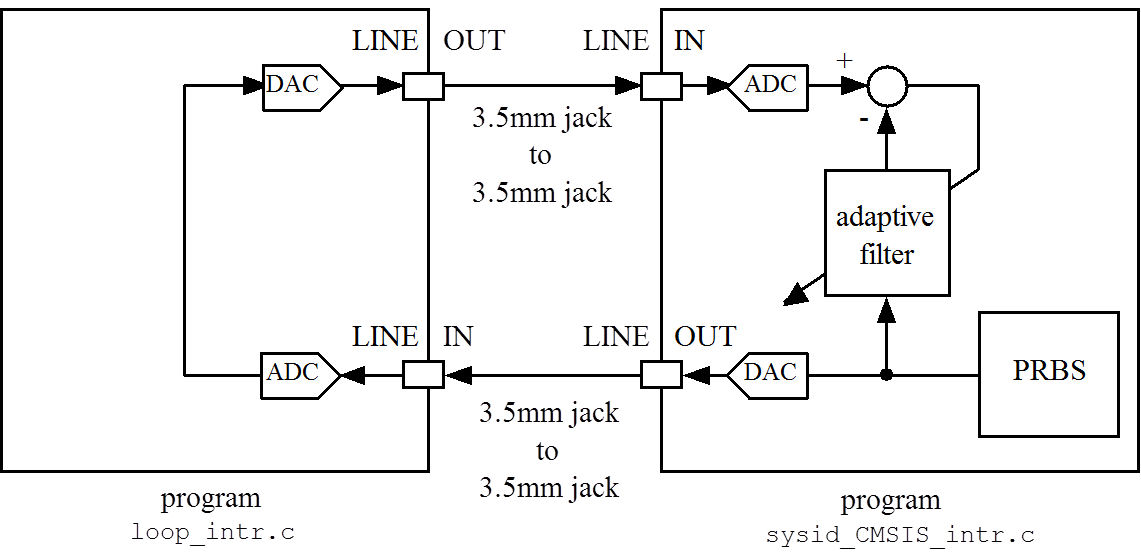


Figure 12: Connection diagram for WM5102 codec bandwidth identification using program sysid\_CMSIS\_intr.c and two audio cards.



Figure 13: *Magnitude* frequency response observed using program sysid\_CMSIS\_intr.c and two audio cards.

# conclusions

At the end of this 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.