Contents

1 Overview 2
2 Step and Impulse Response of DAC Reconstruction Filter 2
3 Magnitude Frequency Response of DAC Reconstruction Filter 4
4 Step Response of Anti-Alias Filter 6
5 Magnitude Frequency Response of Anti-Alias Filter 8
1 Overview

Real-time digital signal processing provides numerous advantages in processing continuous-time (or analog) signals. However, there are inherent limitations in the DSP system itself which can limit the types of signals that can be processed. Through this exercise you will characterize some of the important components of the DSP system in order to understand their impact on overall system capability and performance.

![Block diagram of basic DSP system.](image)

2 Step and Impulse Response of DAC Reconstruction Filter

We begin our investigation by examining the reconstruction filter of Figure 1. Our strategy will be to generate specific waveforms within the DSP and output them through the DAC and reconstruction filter, thus isolating the output portion of the system.

The program listing `square_intr.c` repeatedly outputs a data sequence comprising 32 consecutive values of 10000 followed by 32 consecutive values of -10000. By observing the system output, we can see what might be interpreted as a square wave signal that has been passed through the DAC and reconstruction filter in the WM8731 codec, therefore giving us insight into its step response.

Run the program and observe the output using an oscilloscope. You should see a result similar to that shown in Figure 3.

**Question 1:** Include a screen capture of the oscilloscope measurement. Based on that capture, sketch what you think the impulse response of the DAC and reconstruction filter might look like. Justify your answer.

*Hint:* An impulse is the time derivative of a step and, correspondingly, the impulse response of an LTI system is equal to the time derivative of its step response.

**Question 2:** Modify the program listing `square_intr.c` so that the system output is an approximation to the DAC and reconstruction filter impulse response. Include the code modifications you made and explain them. Include a screen capture of the oscilloscope measurement.

*Hint:* Create and utilize a new look-up table that approximates a Kronecker impulse function.
// square_intr.c

#include "audio.h"

void I2S_HANDLER(void) {  /* ***** I2S Interruption Handler*****/  

const int loop_size = 64;  
const int16_t square_table[loop_size] = { 
  10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 
  10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 
  10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 
  10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 
  -10000, -10000, -10000, -10000, -10000, -10000, -10000, -10000, 
  -10000, -10000, -10000, -10000, -10000, -10000, -10000, -10000, 
  -10000, -10000, -10000, -10000, -10000, -10000, -10000, -10000, 
  -10000, -10000, -10000, -10000, -10000, -10000, -10000, -10000, 
  -10000, -10000, -10000, -10000, -10000, -10000, -10000, -10000,  
};  
static int square_ptr = 0;  
int16_t audio_chR =0;  
int16_t audio_chL =0;  

audio_IN = i2s_rx();  
audio_chL = audio_IN & 0x0000FFFF;  
audio_chR = (audio_IN >>16)& 0x0000FFFF;  

audio_chL = square_table[square_ptr++];  
audio_chR = audio_chL;  
square_ptr %= loop_size; // return to start if we reached the end of the array  

audio_OUT = ((audio_chR < <16) & 0xFFFF0000 ) | (audio_chL & 0x0000FFFF);  
i2s_tx(audio_OUT);  
}

int main(void) {  
  audio_init (hz8000, line_in, intr, I2S_HANDLER);  
  while(1){}  
}
3 Magnitude Frequency Response of DAC Reconstruction Filter

The analog 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 can be illustrated using either the FFT function of an oscilloscope or a spectrum analyzer. Observe the spectrum of the output signal. You should see a spectrum similar to the one shown in Figure 4.

![Figure 4: Magnitude of frequency components present in the analog output waveform generated using square_intr.c.](image)

A more complete representation of the magnitude frequency response of the DAC and reconstruction filter can be obtained using program prbs_intr.c shown in Figure 5. 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. By observing which frequencies are present at the system output, we can discern how the system attenuates certain frequencies.

Run the program and observe the output in both the time and frequency domains.

---

**Question 3:** Include a screen capture of the spectrum analyzer measurement. Comment on what you observe.

**Question 4:** Change the system sampling frequency to 48kHz and again observe the system output. Include a screen capture of the spectrum analyzer measurement. Comment on what you observe, and indicate what changed relative to the previous question.
// prbs_intr.c

#include "audio.h"

void I2S_HANDLER(void) { /* ***** I2S Interruption Handler *****/
    int16_t audio_chR=0;
    int16_t audio_chL=0;
    audio_IN = i2s_rx();
    audio_chL = audio_IN & 0x0000FFFF;
    audio_chR = (audio_IN >>16)& 0x0000FFFF;
    audio_chL = prbs();
    audio_chR = audio_chL;
    audio_OUT = ((audio_chR<<16) & 0xFFFF0000) | (audio_chL & 0x0000FFFF);
    i2s_tx(audio_OUT);
}

int main(void) {
    audio_init(hz8000, line_in, intr, I2S_HANDLER);
    while(1){}
}
4 Step Response of Anti-Alias Filter

We will now follow a similar path to observing the characteristics of the anti-alias filter at the analog input to the real-time DSP hardware. We will use an external signal generator and retrieve ADC samples using the debug capabilities in the Keil IDE.

The program listing loop_buf_intr.c in Figure 6 is much like the audio loop used in earlier lab sessions. However, notice that past input samples are stored in a buffer. By pausing the program and retrieving the samples, we can observe (e.g., via a simple plot in MATLAB) the data samples from the ADC.

```c
// loop_buf_intr.c

#include "audio.h"

static const int buffer_size = 256;
float32_t rbuffer[buffer_size];
float32_t lbuffer[buffer_size];

void I2S_HANDLER(void) {
    static int16_t rbufptr = 0;
    static int16_t lbufptr = 0;
    int16_t audio_chR=0;
    int16_t audio_chL=0;

    audio_IN = i2s_rx();
    audio_chL = audio_IN & 0x0000FFFF;
    audio_chR = (audio_IN >>16)& 0x0000FFFF;

    lbuffer[lbufptr++] = (float32_t) audio_chL;
    lbufptr %= buffer_size; // return to start if we just wrote to the end
    rbuffer[rbufptr++] = (float32_t) audio_chR;
    rbufptr %= buffer_size; // return to start if we just wrote to the end

    audio_OUT = ((audio_chR <<16) & 0xFFFF0000) | (audio_chL & 0x0000FFFF);
    i2s_tx(audio_OUT);
}

int main(void) {
    audio_init (hz8000, line_in, intr, I2S_HANDLER);
    while(1){
}
}
```

Figure 6: Listing of program loop_buf_intr.c

To investigate the step response of the antialiasing filter in the WM8731, connect a waveform 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 500 mV peak-to-peak. Start a Debug session and run program loop_buf_intr.c. Examine the left channel output signal from LINE OUT, and note that it is not a perfect square wave.

Halt the program by clicking on the Stop toolbar button. Type the variable name lbuffer as the Address in the debugger’s Memory 1 window. Set the displayed data type to Decimal and Float as shown in Figure 7 (right-click in the Memory 1 window). The start address of array lbuffer will be displayed in the top left hand corner of the window.

We can view the 128 most recent input sample values read from the ADC by saving them to a file using the debugger’s Command window. The command
Figure 7: Memory 1 window showing contents of array lbuffer.

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

will save the contents of the specified memory range (from start address to end address) to a file in your workspace. For example, to save 128 samples beginning at start address 0x1FFD0C84, use the command

SAVE output.dat 0x1FFD0C84, 0x1FFD0C84 + 4 * 128

Notice that end address has been set as start address + 4 * 128, specifying 128 samples, each 32-bits (4 bytes) in length.

We can plot the contents of that file using the MATLAB function plot_real() which can be found in Lab_3\MATLAB. After executing plot_real in the MATLAB Command Window, you will be prompted to enter the filename (e.g., output.dat) and the samples will be plotted.

Function ihexread (which is called within plot_real) parses the debugger file and returns the floating point samples in a vector, thus allowing you to utilize the samples as you wish. For example, the call

samples = ihexread('output.dat');

reads the ADC values from output.dat and places them in vector samples.

**Question 5:** Comment on what you observe. How does it compare to the step response of the DAC reconstruction filter? Include a plot of the anti-alias filter step response.
5 Magnitude Frequency Response of Anti-Alias Filter

The low pass characteristic of the WM8731 antialiasing filter can be further investigated using program loop_buf_intr.c. By adjusting the signal generator to provide a sinusoidal output, we can observe the filter response to individual frequencies.

**Question 6:** Adjust the signal generator to give a sinusoidal output with amplitude 500mV peak-to-peak. For each of the frequencies specified below, run program loop_buf_intr.c for a few seconds and plot the contents of array lbuffer using MATLAB.

<table>
<thead>
<tr>
<th>input sinusoid frequency in Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>3500</td>
</tr>
<tr>
<td>4500</td>
</tr>
<tr>
<td>5000</td>
</tr>
<tr>
<td>10000</td>
</tr>
</tbody>
</table>

Carefully inspect the MATLAB plots, paying special attention to the signal amplitudes. What do these plots tell you about the magnitude frequency response?
Comments, Suggestions, and Hints for using the Cypress FM4

• You will find 3.5mm stereo to male header breakout boards, included in the TSC Cypress FM4 kit, convenient for connecting the FM4’s audio line in and headphone out jacks to an Analog Discovery board.
  – “T” connection is the Right audio channel
  – “R” connection is the Left audio channel
  – “S” connection is Ground

• You will find 3.5mm stereo to banana cables, available in TSC, convenient for connecting the FM4’s audio line in and headphone out jacks to traditional bench-top laboratory equipment.

• Note in main() that the sample programs usually take audio from the “mic_in” port. Change this to “line_in” to use the line in port. This is suitable for receiving music from an audio player and from the function generator. Unpowered microphones operate at much lower voltages than line level devices.

• The line in jack on the FM4 should only receive a maximum of 894 mV peak-peak; this is the consumer audio line level. Monitor any signal that is put into this jack by using an oscilloscope. Adjust the level of the signal before connecting it to the line-in jack. To be safe, keep the voltage around 500 mV peak-peak.

• Please do not connect anything other than the supplied microphone to the FM4’s mic in jack.

• The FM4 uses standard 3-conductor 3.5mm (1/8 inch) audio connectors. Some device-specific headsets (e.g., for cell phones) have a mic and either 1 or 2 audio output channels and might not behave exactly correctly on the headphone out jack of the FM4. For example, 2015 Apple Earpods (stereo with a mic, requiring 4 conductors) only seem to work properly when the pause button is depressed.

• Headphones with a standard 3-conductor 3.5mm audio connector are available for checkout in TSC.

• You may find that the Left and Right audio channels are reversed when listening to the FM4 output through headphones.
References