Manuel F

FInal Report

SMMD project reverb

1.Task

Goal of this project was to build an audio effect device based on the STM32F796I-DISCO development platform. The desired audio effect was a reverb. In order to realize this idea the following requirements had to be met (all non bold requirements are considered optional):

1. **reading samples from the onboard audio codec**
2. **processing the read samples**
3. **outputting the processed samples via the onboard audio codec**
4. handling user IO via the included touchscreen

1.1 Audio Input

The audio input should handle fetching samples from the onboard audio codec using the HAL. Due to performance constraints, it should do so while ensuring minimal CPU usage.

1.2 Audio processing

Audio processing should create a reverb effect using DSP audio filters. If possible, a Schroeder reverb filter setup should be used. For further information on a Schroeder reverb please consult this [paper](https://pdfs.semanticscholar.org/48a5/32a015e3cbb863e28b6cca8ad4ca849065b7.pdf). Should time or CPU constraints not allow this implementation, a FIR filter setup using the impluse response of a reverberating room can be used instead.

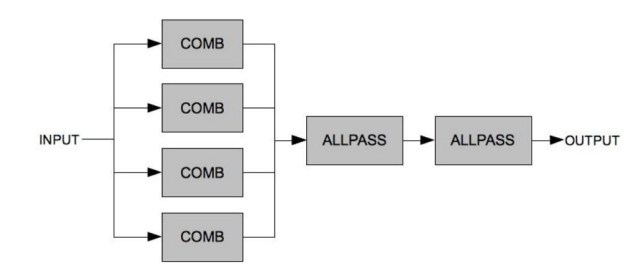


Figure 1 ‑ 1 simple schroeder reverberator

1.3 Audio output

The audio output should handle outputting samples to the onboard audio codec using the HAL. Due to performance constraints, it should do so while ensuring minimal CPU usage. Additionally possiblie clipping caused by the audio processing should be prevented before outputting.

1.4 User IO

The user IO should allow the user to enable and disable the reverb effect. Should multiple reverb effects be implemented, the user should be able to enable/disable them individually and combine them. Ideally this IO is to be implemented using the Touchscreen of the STM32F769I. Should time or CPU constraints not allow this implementation, IO is to be handled using the user button of the board.

The following chapters shall describe how these requirements were met by first discussing the hardware used in the project. Subsequently the process of developing the software will be described. The final state of the software will be explained in the final chapter, after which a summary will sum up the results of the project.  
  


Figure 1 ‑2 first UI draft

2. Table of contents

Inhalt

[1.Task 1](#_Toc33375560)

[1.1 Audio Input 1](#_Toc33375561)

[1.2 Audio processing 1](#_Toc33375562)

[1.3 Audio output 1](#_Toc33375563)

[1.4 User IO 1](#_Toc33375564)

[3.Hardware 3](#_Toc33375565)

[4.Software 4](#_Toc33375566)

[4.1 Development 4](#_Toc33375567)

[4.2 User Interface 4](#_Toc33375568)

[4.3 Final results 5](#_Toc33375569)

[6.Appendix 8](#_Toc33375570)

[7. Quotations 9](#_Toc33375571)

3.Hardware

The audio effect device is based on the *STM32 F769I-Disco* development platform. It consists of a STM32F769NIH6 microcontroller and supplementary hardware. A capacitive touchscreen can be attached to the board using a DSI connector. Audio IO is handled via a SAI audio codec.



Figure 3‑1 STM32F7I69-Disco

4.Software

Due to growing experience with the hardware and its capabilities and especially its limitations, the architecture of the software has changed a lot during development. This chapter will first discuss the development of the software, and why changes were necessary.

4.1 Development

The early versions of the software were based on interrupt based audio processing. This meant the software would send an interrupt to the processor every time a sample was available. The processor would then execute an ISR called Sample\_Callback(). In this function, the sample would then be placed in a circulary buffer. The buffer was then used to feed the audio filters. The current output sample of the filters would then be written to the audio output. However, this architecture proved to be problematic in multiple ways.

First of all, in order to achieve an acceptable audio quality the sampling rate of the codec was set to 48 kHz. This meant that each call of the ISR only had a theoretical maximum of 21 microseconds to finish its computations, which proved to be insufficient to rum the filters. On top of that in practice the save and load operations performed to handle each interrupt further diminished the theoretical time for each callback.

In order to lower the amount of interrupts and filter executions, the software architecture was changed to a DMA based approach. This meant the audio codec would write a block of samples into memory before calling the ISR. The ISR, now called ProcessBuffer(), would then process the entire block of samples and output it. In order to ensure a static block of samples during processing, the audio codec would alternate between a so called ping and pong buffer on both the in- and outputs. Each of the alternating callbacks required to realize this structure are called half callbacks.

This did save a lot of CPU usage. However, the original plan of using 4 parallel combfilters and 2 serial allpass filters afterwards to realize a Schroeder reverb were still too CPU intensive. This resulted in very loud repeating chirping noises at the output.

In order to address this the filter was scaled back to a simple FIR using a optimized filter function [provided by ARM for their processors](https://www.keil.com/pack/doc/CMSIS/DSP/html/group__FIR.html#ga0cf008f650a75f5e2cf82d10691b64d9). However, even this was not enough to resolve the performance issues. The block size was increased from its default value and the impulse response had to be shortened.

4.2 User Interface

The user interface is currently based on the Touch-Panel of the STM32. The software implements a simple LCD routine which draws a round button in the middle of the screen which can be activated by pressing it. If the button is pressed, the color shifts to green instead of the standardized gray. The detection of the touch input gets stored in a flip-flop variable which is available globally.

The software already has a problem: multiple touch recognition. If you leave your finger pressed on the touch screen and move it a few pixels on the screen, the software detects these movements as new touch inputs and activates the button again. The solution would be a "difference function" which checks if the newly detected input is really a new touch input or just a movement of the existing input by a few pixels. The final version of the Software will include this difference function.

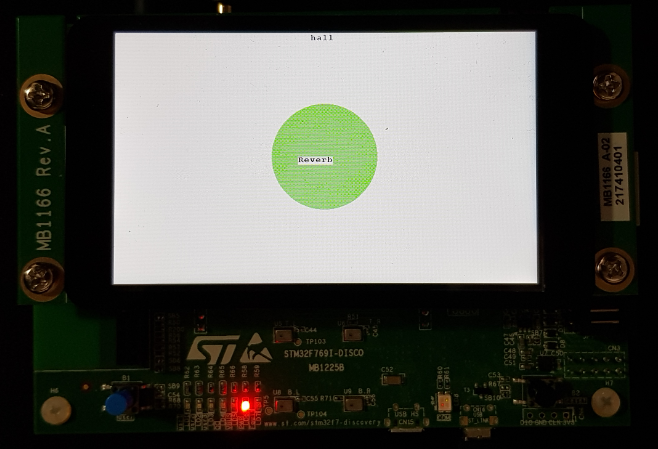


Figure 4‑1 final UI design

4.3 Final results

Consequently, the software in its final form is a DMA based approach. The IO is handled though a ping pong buffer configuration. It uses a FIR filter for processing and is optimized for the processors capabilities. The used filter function stores its state between calls, so no overlap and add routine had to be implemented to handle a continuous effect when switching between buffers. The maximum impulse response length at 48 kHz and a block size of 512 turned out to be 512 samples (about 11 microseconds).

The following code snippet will help illustrate the inner workings of our signal processing:

void ProcessBuffer()

{

*//seperate samples*

seperated\_buf\_index = 0;

for (int n = 0; n < BUFFER\_SIZE; n+=2)

{

audio\_chL[seperated\_buf\_index]= ProcessingBuffer.Input[n];

audio\_chR[seperated\_buf\_index]= ProcessingBuffer.Input[n+1];

seperated\_buf\_index ++;

}

*//cast input to float*

for(int i = 0; i<BUFFER\_SIZE/2;i++){

audio\_chL\_f[i] = (float32\_t)audio\_chL[i];

}

*//initialise filter*

arm\_fir\_instance\_f32 S = {numTaps, pState, h};

*//filter input*

arm\_fir\_f32 (&S, audio\_chL\_f, conv\_resL,BUFFER\_SIZE/2);

*//reassemble buffer*

seperated\_buf\_index = 0;

for (int n = 0; n < BUFFER\_SIZE; n+=2)

{

ProcessingBuffer.Output[n] = (int32\_t) (0.28\*conv\_resL[seperated\_buf\_index]);

ProcessingBuffer.Output[n+1] = audio\_chR[seperated\_buf\_index];

seperated\_buf\_index ++;

}

}

The process buffer function is called every time one of the input buffers finished being filled. As left and right channel samples are stored in an alternating pattern in the buffer they first have to be separated in the first for loop. The second for loop then casts the samples to float32\_t in order to make them compatible with the filter function. The filter is then initialized using the length of the audio channel buffers (numTaps), the state of the last filter execution (pState, global variable) and the filter impulse response (h, stored in the filter header file). After initialization, the filter is then applied to the left audio channel. The right channel is untouched in order to provide a comparison signal. After processing, the audio channel samples are written alternating to the output buffer. The final function includes more code that supports activating and deactivating the filter. Due to CPU limitations the filter is appplied to the left channel only. Alternatives using a shorter impulse response and stereo processing did not provide the desired sound quality, which forced this compromise.

5. Summary

In summary, the project was scaled down a lot during development due to performance constraints of the CPU. The final software is a DMA based FIR filter system.  
The UI could need some fine tuning in case of the touch sensitivity, but overall it works pretty well.

6.Appendix

* **List of abbreviations**
  + FIR - finite impulse response
  + DMA - direct memory access
  + ISR - interrupt service routine
  + CPU - central processing unit
  + codec - encoder/decoder unit
  + SAI - serial audio interface
  + DSI - display serial interface
  + UI - user interface

7. Quotations

[paper on schroeder reverb](https://pdfs.semanticscholar.org/48a5/32a015e3cbb863e28b6cca8ad4ca849065b7.pdf)

[inspiration for schroeder reverb implementation](https://medium.com/the-seekers-project/coding-a-basic-reverb-algorithm-part-2-an-introduction-to-audio-programming-4db79dd4e325)

[source of the used impulse responses](http://www.echothief.com/)

[filterfunction documentation](https://www.keil.com/pack/doc/CMSIS/DSP/html/group__FIR.html#ga0cf008f650a75f5e2cf82d10691b64d9)

[other filterfunctions considered during development](https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html)