Laboratory 3

Exploring IIR, Delay/Reverb, Audio Output using the STM32F411E DISCO

1. PREREQUISITES

- You have the ENCM 515 lab kit (the STM32F411E DISCO board, a USB A to mini-B cable, and headphones
- You have successfully installed the STM32CubeIDE and the STM32Cube Programmer[1]
- Check D2L for other materials that might be required

2. OUTLINE

In this lab, the aim is to implement a classic IIR filter (the Biquad filter) while outputting audio. As with the previous lab, this activity will assume that you are increasingly comfortable with using the IDE, writing programs, and optimizing them.

By the end of this lab, you should be:

- Able to implement a simple IIR filter
- Comfortable with using the ITM port and SWV Trace Log to profile your code
- Able to use the STM32CubeProgrammer to load data in and out of memory
- Be able to optimize filter code using a variety of techniques, including the MAC and SIMD instructions of the Arm Cortex-M4, as well as implement circular buffers.

Q> There are several Questions throughout the lab for you to answer. You should collate your answers/observations/screenshots in a Word document (or equivalent), such as the lab sheet available on D2L. If there are demo components, you should show these to an Instructor/TA. If you are doing this lab remotely, you will need to screen-record/video the demo components.

Submit your completed Lab 3 sheet as a pdf together any other added/modified .c or .h files to a zip archive and upload to D2L.

PART ONE

3. PREP

Using STM32CubeProgrammer, download `voice8k-noisy.bin` to 0x08020000 to your STM32F411E DISCO. As with the previous lab, this file is audio data, sampled at 8000 Hz, and stored in the well-known `wav` format. This data is in the 16-bit PCM format.

➢ *Use this margin to make notes!*

Open STM32CubeIDE and import the **lab3_proj** project into your workspace. **It is very similar to lab 2!** To do this, go to File > Import, and follow the prompts. You should be able to build the project without any issues after importing. The project has a number of additional files that you can play around with outside the lab. Once imported, build the project.

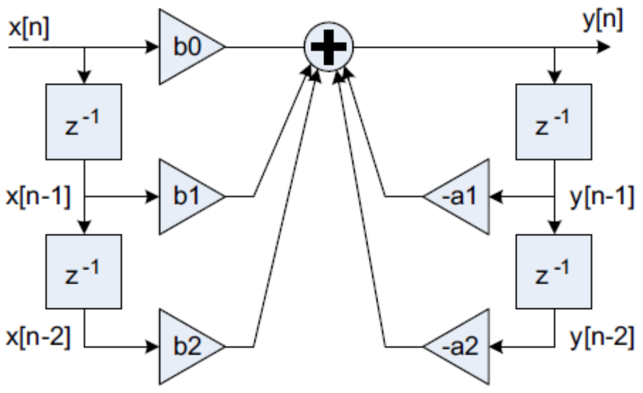
Note: This project is has some custom code for initializing peripherals that isn't the same as the autogenerated code that is produced by modifying the .ioc. DO NOT MODIFY THE .IOC OR AUTO-GENERATE NEW CODE.

[1] https://www.st.com/en/development-tools/stm32cubeprog.html

4. LAB TASKS

If you run/debug the code "out of the box" you should be able to hear some sound! Notice that it is a little bit noisy, however, so we'll try and implement a filter to improve the quality a little. For this lab, we will focus on **fixed-point** implementations (i.e., the usual 16-bit data we've been working on for the past few labs).

A very common/popular filter design is the Biquad filter, which is a second-order IIR filter. Biquad filters are often used to construct higher-order filters. You can search online for material about the theory behind Biquad filters, but for this course, you should simply understand what's going on (and therefore, what's required for software), by examining the system diagram. For our purposes, we will initialize the sample history to 0s.



Direct Form I Biquad filter

Q1> Write the code for a function to implement the Biquad filter with the following coefficients: `b0 = b2 = 10158, b1 = 20283, {-a1} = -2261, {-a2} = -10060`**. Include an annotated version of your source code in your report. Take a screenshot of the assembly code for your function (after compiling), and explain which assembly instructions correspond to which parts of the filter's operation (e.g., what part is for managing the state variables (history of the input/output), what part corresponds to multiplication with the coefficients, etc.) How many cycles are required for processing a sample?**

Q2> Analyze the code that you have written and use one or more techniques to optimize different parts of the code. Explain which technique(s) you've used and why you've used them. How many cycles per sample do you achieve after using your optimizations?

Hopefully, now that we've cleaned up the audio signal, we can try to perform some light effects processing! In this lab, we'll do two simple things.

First, let's implement an echo, which is represented by the following difference equation:

$$y[n] = x[n] + \alpha x[n - n_0]$$

where n_0 is the amount of delay and α is the delay mix parameter. Play around with a few options of α until you pick one that you like.

Q3>Implement a new function that produces an output with echo. Aim for a delay of around 125 ms. How large does your history (state variable array) have to be for this delay? Is this an FIR or IIR filter? Add to your report an analysis of the compiled code. Discuss any "optimizations" that you needed to use to make this function work under the time constraints.

Next, let's implement some reverberation. The simplest form of reverberation is to use a comb filter (again, take some time to read up online about it). For our purposes, the difference equation is:

$$y[n] = x[n] + \alpha y[n - n_0]$$

Q4>Implement a new function that produces an output with reverb. Is this an FIR or IIR filter? Add to your report an analysis of the compiled code. Add to your report an analysis of the compiled code. Discuss any "optimizations" that you needed to use to make this function work under the time constraints.

Q5>Write a reflection of what you have observed and learned in this lab, drawing connections between what we've discussed in class and what you've done in this lab.

To submit:

- **Add your lab sheet and any other added/modified .c or .h files to a zip archive and upload to D2L.**