EE 371, Lab 5

Digital Signal Processing

(Adopted from Intel's University Program)

Lab Objectives

This is an exercise in using the audio coder/decoder (CODEC) on the DE1-SoC board. The lab involves connecting a microphone to the audio CODEC to provide input sound, altering the received sound by filtering out noise, and then playing the resulting sound through speakers/headphones.

Background

Sounds, such as speech and music, are signals that change over time. The amplitude of a signal determines the volume at which we hear it. The way the signal changes over time determines the type of sounds we hear. For example, an 'ah' sound is represented by a waveform shown in Figure 1.

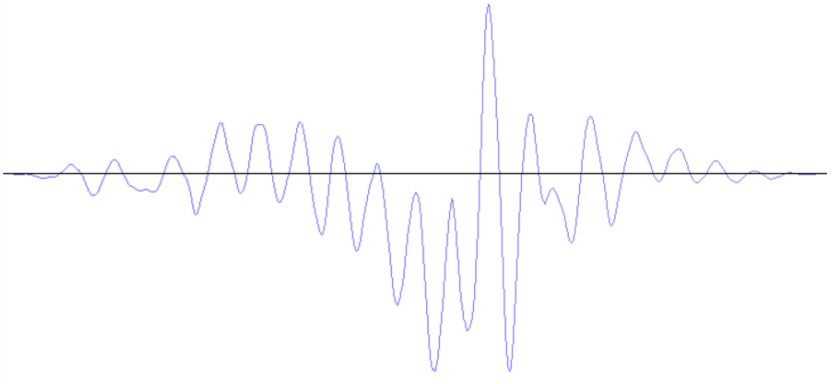

Figure 1. A waveform for an 'ah' sound

The waveform is an analog signal, which can be stored in a digital form by using a relatively small number of samples that represent the analog values at certain points in time. The process of producing such digital signals is called *sampling*.

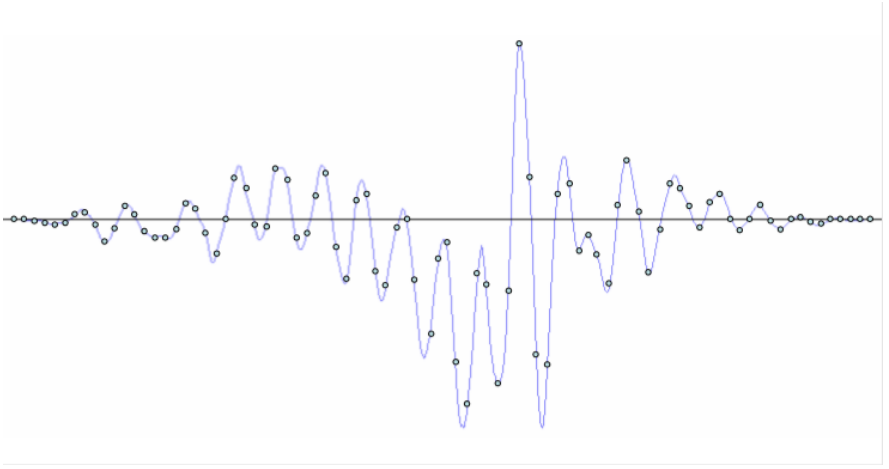

Figure 2. A sampled waveform for an ’ah’ sound.

The points in Figure 2 provide a sampled waveform. All points are spaced equally in time and they trace the original waveform.

The DE1-SoC board is equipped with an audio CODEC capable of sampling sound from a microphone and providing it as input to a circuit. By default, the CODEC provides 48000 samples per second, which is sufficient to accurately represent audible sounds.

This lab involves the design of several circuits that take input from a microphone through the CODEC, record and process this sound data, and then play it back through speakers. To simplify the task, a simple system that can record and playback sounds on the board is provided as a "starter kit". The system, shown in Figure 3, comprises a Clock Generator, an Audio CODEC Interface, and an Audio/Video Configuration modules.

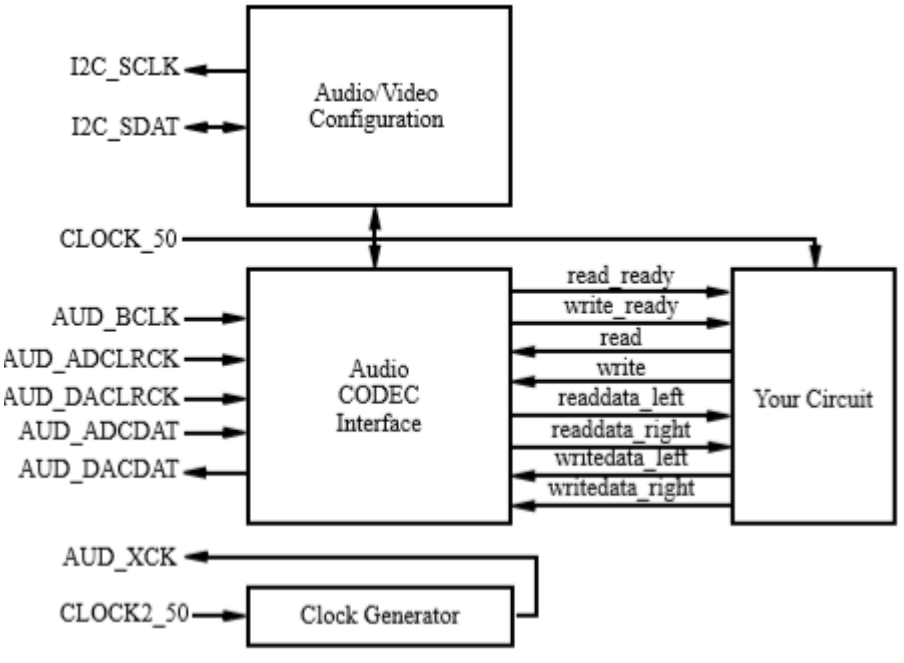

Figure 3. Audio System for this lab

The left-hand side of Figure 3 shows the inputs and outputs of the system. These I/O ports supply the clock inputs, as well as connect the Audio CODEC and Audio/Video Configuration modules to the corresponding peripheral devices on the DE1-SoC board. In the middle of the figure, a set of signals to and from the Audio CODEC Interface module is shown. These signals allow the circuit depicted on the right-hand side to record sounds from a microphone and play them back via speakers.

The system works as follows. Upon reset, the Audio/Video Configuration begins an autoinitialization sequence. The sequence sets up the audio device to sample microphone input at a rate of 48kHz and produce output through the speakers at the same rate. Once the autoinitialization is complete, the Audio CODEC begins reading the data from the microphone once every 48000^{th} of a second, and sends it to the Audio CODEC Interface core in the system. Once received, the sample is stored in a 128-element buffer in the Audio CODEC Interface core. The first element of the buffer is always visible on the readdata_left and readdata_right outputs when the read_ready signal is asserted. The next element can be read by asserting the read signal, which ejects the current sample and a new one appears one or more clock cycles later, if the read_ready signal is asserted.

To output sound through the speakers a similar procedure is followed. Your circuit should observe the write_ready signal, and if asserted write a sample to the Audio CODEC by providing it at the writedata_left and writedata_right inputs and asserting the write signal. This operation stores a sample in a buffer inside of the Audio CODEC Interface, which will then send the sample to the speakers at the right time.

A starter kit that contains this design is provided as part of this lab.

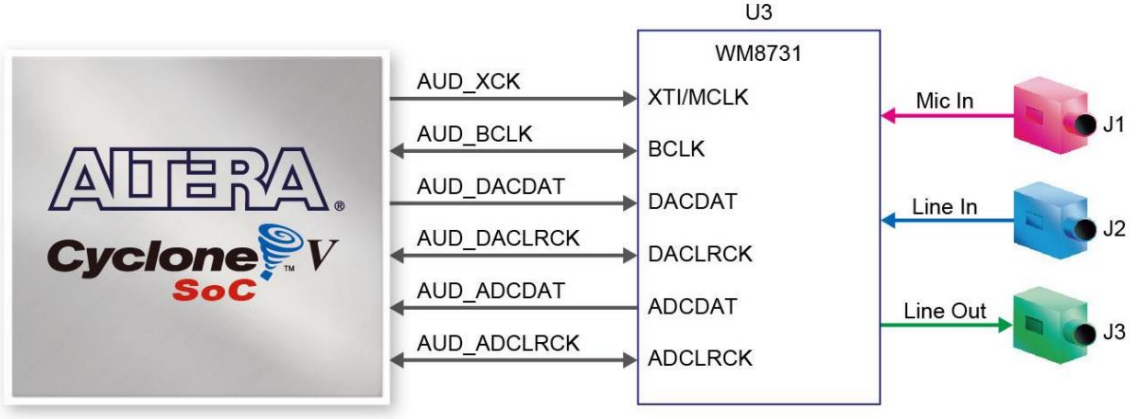

Figure 4. Connections between FPGA and Audio CODEC

Task 1

You are to make a simple modification to the provided starter kit circuit to pass the input from the microphone to the speakers. You should take care to read data from and write data to the Audio CODEC Interface only when its ready signals are asserted.

Compile your circuit and download it onto the DE1-SoC board. Connect microphone and speakers to the Mic In (pink) and Line Out (green) ports of the board and speak to the microphone to hear your voice through the speakers.

Note: you may hear a lot of noise, or your speaker might be very quiet. Try listening in a quiet place and/or using some 3.5mm headphones.

Task 2

In this task, you will learn a basic signal processing technique known as *filtering*. Filtering is a process of adjusting a signal - for example, removing noise. Noise in a sound waveform is represented by small, but frequent changes to the amplitude of the signal. A simple logic circuit that achieves the task of noise-filtering is an averaging Finite Impulse Response (FIR) filter. The schematic diagram of the filter is shown in Figure 5.

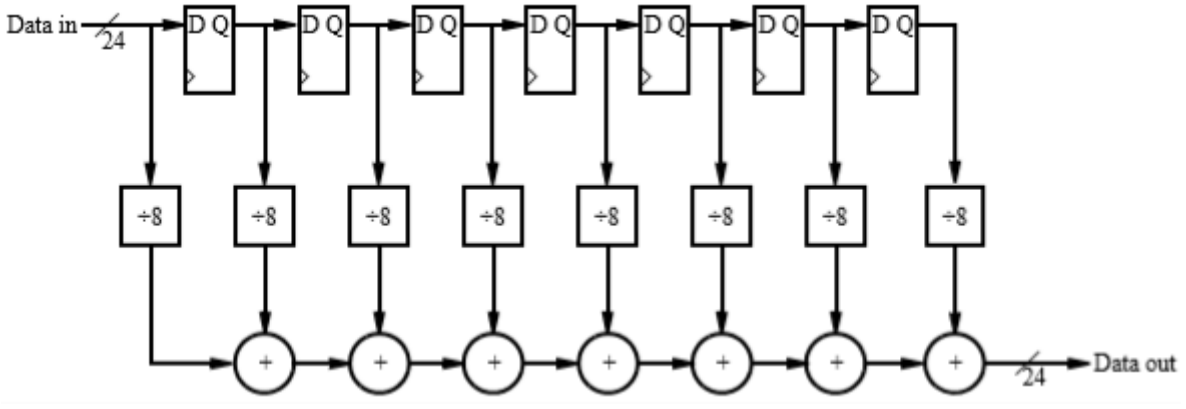

Figure 5. A simple averaging FIR filter

An averaging filter, like the one shown in Figure 55, removes noise from a sound by averaging the values of adjacent samples. In this particular case, it removes small deviations in sound by looking at changes in the adjacent 8 samples. When using low-quality microphones, this filter should remove the noise produced when you speak to the microphone, making your voice sound clearer.

You are to implement the circuit shown in Figure 5 to process the sound from the microphone, and output the filtered sound through the speakers. Do you notice any difference between the quality of sound in this part as compared to Task 1? (Provide your answer in the lab report).

NOTE:

It is possible to obtain high-quality microphones with noise-canceling capabilities. In such circumstances, you are unlikely to hear any effect from using this filter. If this is the case, we suggest introducing some noise into the sound by adding the output of the circuit in Figure 6 to the sample produced by the Audio CODEC.

```
module noise_generator (clk, enable, Q);

    input logic clk, enable;

    output logic [23:0] Q;

    logic [2:0] counter;

        always_ff @(posedge clk)

                if (enable)

                    counter = counter + 1'b1;

        assign Q = {{10{counter[2]}}, counter, 11'd0};

    endmodule
```

Figure 6. Circuit to generate some noise.

The circuit is a simple counter, whose value should be interpreted as a signed value. The circuit should be clocked by a 50MHz clock, and the enable signal should be driven high when the Audio CODEC module can both produce and accept a new sample.

To hear the effect of the noise generator, add the values produced by the circuit to each sample of sound from the Audio CODEC in the circuit in task 1.

Task 3

The implementation of the averaging filter in task 2 may have been effective in removing some of the noise, and all of the noise produced by the noise generator. However, if your microphone is of low-quality or you increase the width of the counter in the noise generator, the filter in task 2 would be insufficient to remove the noise. The reason for this is because the filter in task 2 only looked at a very small time frame over which the sound waveform was changing. This can be remedied by making the filter larger, taking an average of more samples.

In this task, you are to experiment with the size of the filter to determine the number of samples over which you have to average sound input to remove background noise. To do this more effectively, use the design of an averaging FIR filter shown in Figure **Error! Reference source not found.**.

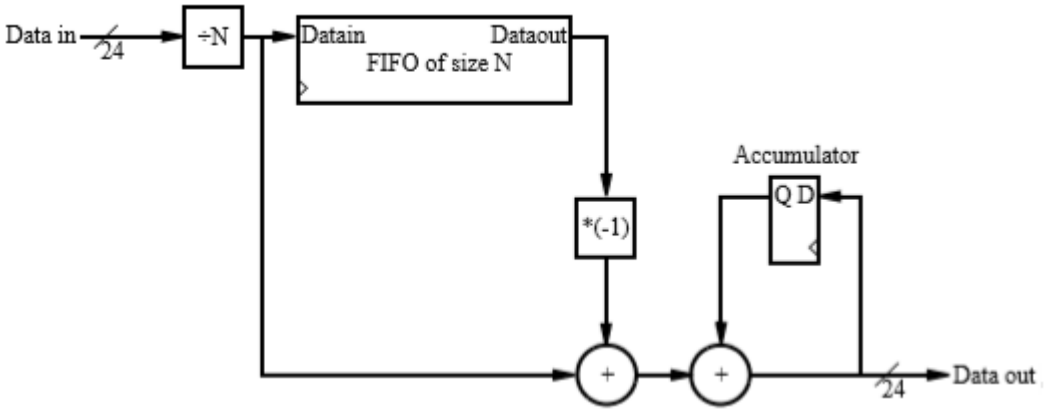

Figure 7. N-sample averaging FIR filter.

To compute the average of the last *N* samples, this circuit first divides the input sample by *N*. Then, the resulting value is stored in a First-In First-out (FIFO) buffer of length *N* and added to the accumulator. To make sure the value in the accumulator is the average of the last *N* samples, the circuit subtracts the value that comes out of the FIFO, which represents the $(n+1)^{th}$ sample.

Implement, compile and download the circuit the DE1-SoC board. Connect microphone and speakers to the Mic and Line Out ports of the board and speak to the microphone to hear your voice through the speakers. Experiment with different values of *N* to see what happens to your voice and any background noise, remembering to divide the samples by appropriate value. We recommend experimenting with values of *N* that are a power of 2, to make the division easier.

Use the SignalTap II functionality of Quartus to verify the contents of your FIFO buffer.

If you have a portable music player, with a connector such that you can supply input to your circuit through the Mic port, try experimenting with different sizes of the filter and its effect on the song you play. Provide the results of your experimentation in the lab report.

Lab Demonstration and Submission Requirements

- Submit a lab report that includes the procedures and results obtained in the lab. Suggestions on what to include follow:
    - Abstract: A brief overview of the entire report
    - Introduction: What the lab is (specifications or background info, etc.
    - Procedure/Results/Analysis:
        - Steps to complete the lab
        - Description of each module (explain how they work, point out code, etc.)
        - Simulations of each module with comments on important things to notice about them.
        - Overall description of what the finished product was/how it behaved
        - Discussions of any problems had while completing the lab, or unsolved errors
    - Conclusion: A final summary, reflection on the lab or what was learned, etc.
- Include any hurdles or challenges (if any) that you faced in finishing this lab and how you overcome them.
- Submit the SystemVerilog code for all tasks and include screenshots for the ModelSim waveforms of your modules and your signal tap results.
- Submit the Flow Summary (produced during compilation) of compiling your system.
- In your report, include the number of hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).
- Submit your report and programs to Canvas. No hard copies. Submit a pdf file as well as a compressed folder of your source files.