Design Document: Autotune

By Cam Coleman (cc4535) , Adam Banees (ab4972) , and Khaela Harrod (klh2173)

Table of Contents

Introduction

In this project, we will create a program that utilizes a board to correct the pitch of any given sound. We will use hardware devices such as a microphone, speaker and keyboard to implement our design and test our program.

The idea is to get audio data from a microphone through the audio CODEC and store it into the FPGA memory. We will set up a USB keyboard function to say when to start recording. Once we have the recording, the audio data is then read through the device drivers via Avalon bus and altered through our autotune algorithm. Once altered, the audio file is then sent back through the bus and written into the FPGA memory and output the resulting file via SCP.

System Block Diagram

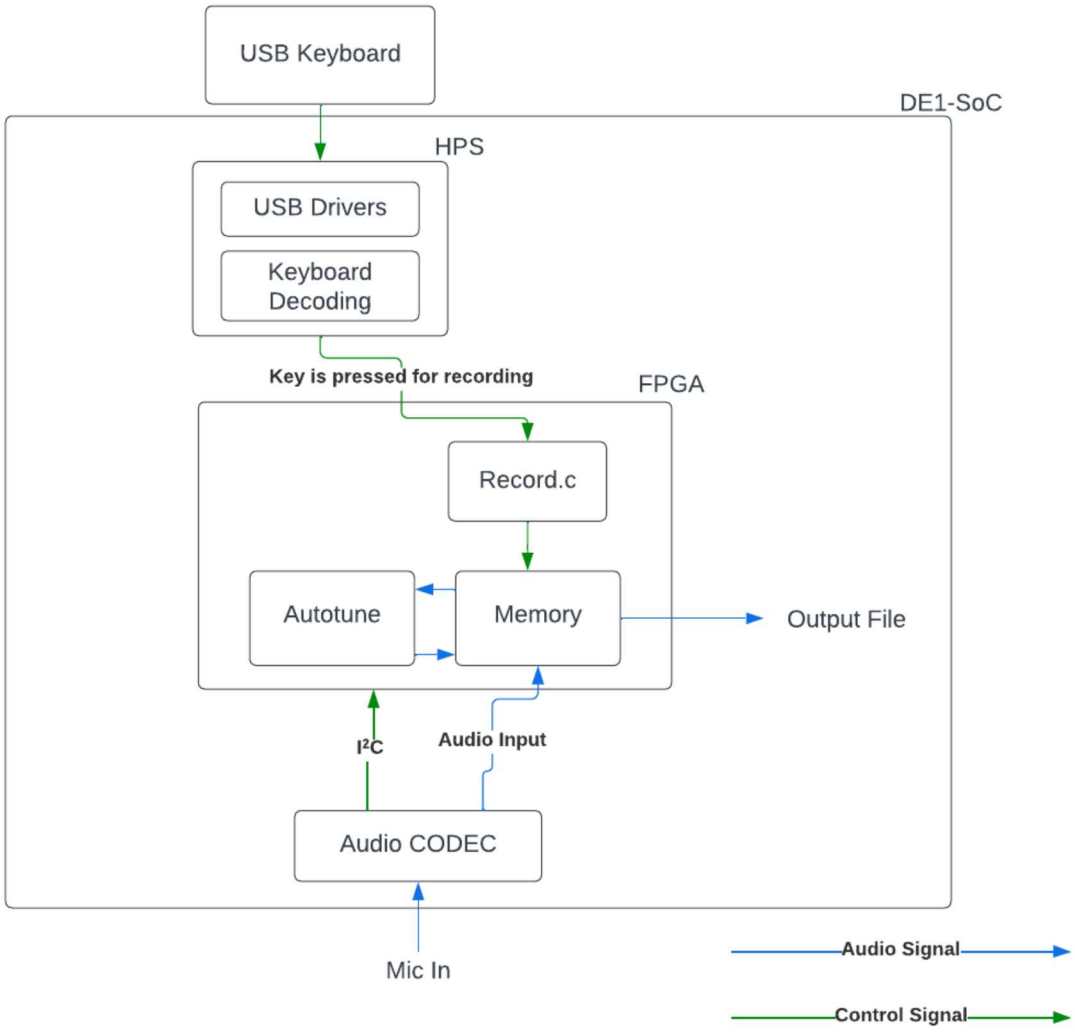

Figure 1: Block Diagram for Project Design

I2C

I2C has become a popular serial communication standard used in embedded systems to link microcontrollers and peripherals. The data line (SDA) and the clock line (SCL) are the two lines that I2C employs for communication. Communication takes place when one device pulls the data line down while the clock line is up on these high-voltage connections. The data transfer between the devices is then synchronized by switching the clock line. I2C permits the use of numerous devices, each

with a distinct address, on a single bus. This makes it simple to interface with many peripherals, including sensors, EEPROMs, and LCD screens. All in all, I2C is a simple and dependable communication protocol that is frequently used in embedded systems for its ease of implementation and versatility.

Record.c

Signal telling memory what you want

USB Keyboard

A USB keyboard is a computer peripheral that connects to a computer via the USB interface. It is a common input method for entering text and commands into a computer. A USB keyboard normally consists of a set of keys that are organized in a certain pattern, the most popular being the QWERTY layout. It also comes with a cord that plugs into a computer's USB port. Our USB keyboard is connected to the HPS which is then used to tell the FPGA when to start recording.

Algorithms

Applications for pitch tracking algorithms include speech recognition, music analysis, and audio signal processing. One of the simplest and most popular pitch tracking algorithms is the autocorrelation approach. Finding the lag that corresponds to the highest peak requires computing the audio signal's autocorrelation function. The period of the fundamental frequency, which can be used to determine the pitch, corresponds to this lag. The Fast Fourier Transform (FFT) and IFFT (Inverse Fast Fourier Transform) routines in C can be used to create the autocorrelation approach. The harmonic product spectrum (HPS) approach calculates the sum of the power spectrum and its subsampled versions. A new spectrum with noticeable harmonic peaks is the result of this. The pitch can be extracted from the fundamental frequency of the strongest harmonic peak. The HPS method can be implemented in C using FFT and IFFT functions.

In order to reduce high-frequency noise and smooth out signals, low pass filtering is frequently used in audio signal processing. Low pass filters can be implemented using a variety of techniques in C for audio applications. Typical C low pass filtering and audio smoothing algorithms include filters with finite impulse response (FIR). This kind of filter is popularly utilized in audio

applications and has a finite impulse response. To obtain the output signal, the input signal is convolved with a finite-length impulse response. Several methods, including windowing, the Parks-McClellan algorithm, and the frequency sampling approach, can be used to create FIR filters. These can be built in C using the FFT algorithm or a direct convolution algorithm.

Another type of filter with infinite impulse response that is frequently used in audio applications is the infinite impulse response (IIR) filter. The input signal and the output signal are applied iteratively to a set of filter coefficients in this process. Several methods, including Butterworth, Chebyshev, and elliptic filters, can be used to create IIR filters. Using recursive algorithms like Direct Form I or Direct Form II, they can be implemented in C. A median filter is a method that computes the median of the most recent n samples and the n samples that came before them, where n is the window size. It helps to eliminate spikes or outliers from the signal. Using an iterative loop to cycle through the samples and a buffer to store the previous samples, median filters can be written in C.

Resource Budgets

Using the Avalon audio interface, we can change the sample rate and bit depth of the data that the Audio CODEC is sending to the FPGA. If we have a sample rate of 32 kHz, a bit depth of 16 bits from 1 channel and 5 seconds of audio we get the following:

Total Number of samples = (Sample rate) * (Duration) = 32,000 * 5 = 160,000 samples

Number of bytes per sample = (Bit depth) / 8 bytes per sample = 16 / 8 = 2 bytes per sample

Total Memory Required = (Total number of samples) * (Number of bytes per sample) = 160,000 * 2 = 320,000 bytes = 312.5 KB(approx.)

The audio file fits our FPGA memory constraint of 512 KB so we will not have to worry of running out of space or forgetting about space for overhead.

Hardware and Software Interface

The main hardware-software interface we will be using in our project is the Avalon bus. Avalon interfaces simplify system design by allowing you to easily connect components in Intel FPGA. [1] We will be interfacing with the audio data sent from the Audio CODEC to the FPGA memory of the DE1-SoC board and implementing our autotune algorithm in the userspace. Our device driver will read the data coming from this memory and write the data after the audio file has been altered. In this project, we will use the Avalon Memory Mapped Interface (Avalon - MM) to read and write the addresses given from these registers.

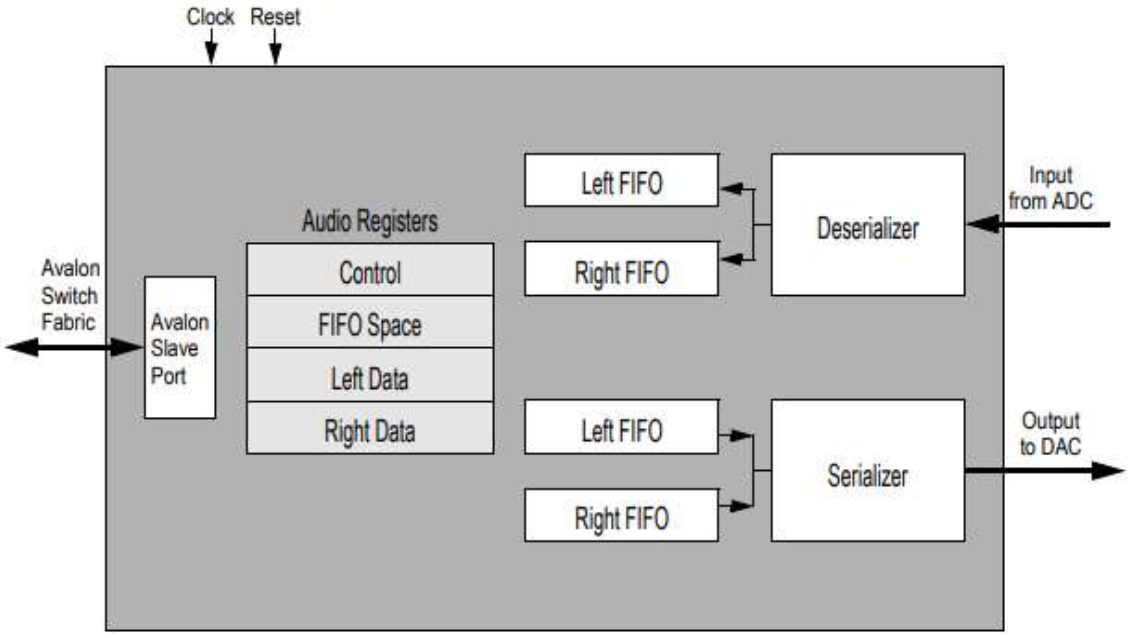

Figure 2: Block Diagram for Audio Core with Memory-Mapped Interface [2]

As for the interfacing for the keyboard, we will be using USB. We will implement something similar to Lab 2 with interfacing of the keyboard and simply program a record file to start recording our voice on the microphone by pressing spacebar.

Milestones

Milestone 1: Our first goal is to be able to record our voices through the microphone with our hardware and store it to memory.

Milestone 2: Our second goal is to be able to implement the autotune algorithm.

Milestone 3: Our third goal is to be able to initialize record.c via keyboard and output audio file to host computer.

References

[1]https://www.intel.com/content/www/us/en/docs/programmable/683091/20-1/introduction-to-the-interface-specifications.html
[2]https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/Audio_core.pdf
[3]https://learn.sparkfun.com/tutorials/i2c/all#introduction
[4]https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-examples/horizontal/fpga-to-hps-bridges-design-example.html
[5]https://www.projectrhea.org/rhea/index.php/Embedded_Fixed_Point_FFT
[6]https://www.juansaudio.com/amp/iir-vs-fir-understanding-their-differences