FPGAirPods Project Proposal

Gokul Kolady, Ben Kettle, Niko Ramirez — October 25, 2020 — 6.111

1 OVERVIEW

Our 6.111 final project aims to implement active noise cancellation as seen in over- and in-ear headphones such as the Bose QC35 and the AirPods Pro (hence the name). In order to do this, we will first need to create a physical model of a single headphone (one ear cup) that we will use to test. Inside the ear cup, we will install a speaker (far from the ear) and a digital microphone (closer to the ear). We will also use a digital microphone external to the ear cup in order to capture ambient noise.

The FPGA itself will handle the computational side of the active noise cancellation. This will primarily involve an adaptive filter that will continually improve its coefficients in order to minimize the error function–ie, the noise that makes it through and is not cancelled out. We run the input from the external microphone through this filter, and play the filtered result out of the speaker with the aim of cancelling noise.

2 MATERIALS

Several materials will be used in order to construct a prototype ”headphone” to test on:

- 2x SPH0645LM4H microphone breakout boards from Adafruit will be used to collect noise from inside and outside the ear cup. These communicate via I2S, which will remove the need for ADCs between the microphone and the FPGA, and minimize potential for noise in the wires from the FPGA to the earpiece.
- Speaker
- Amplifier? We could put this inside the cup, or we might be able to use the onboard amplifier on the FPGA–I’m not sure.
- Cup
- Foam
- Wire

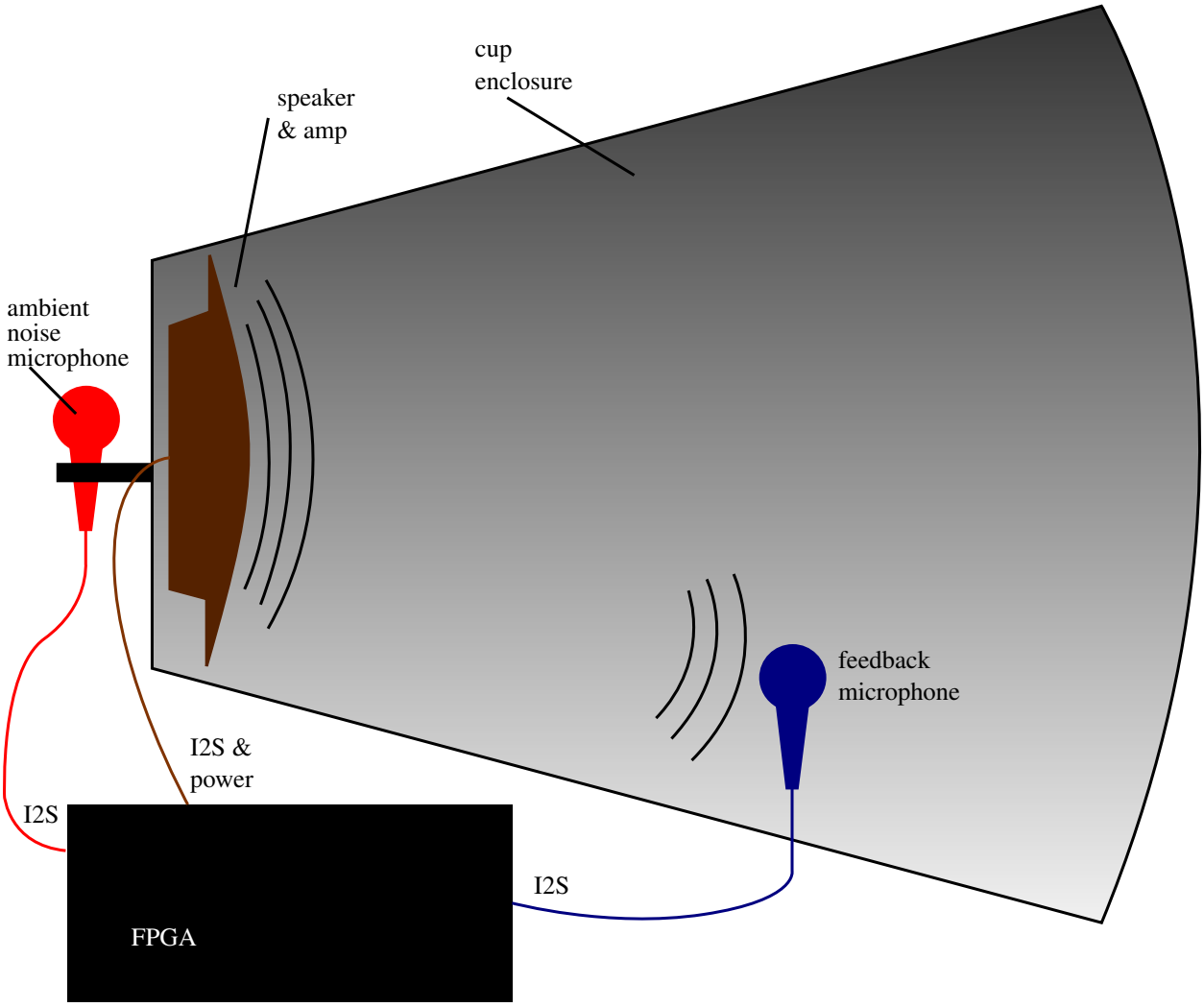

Figure 1: The peripherals that will be included in our project

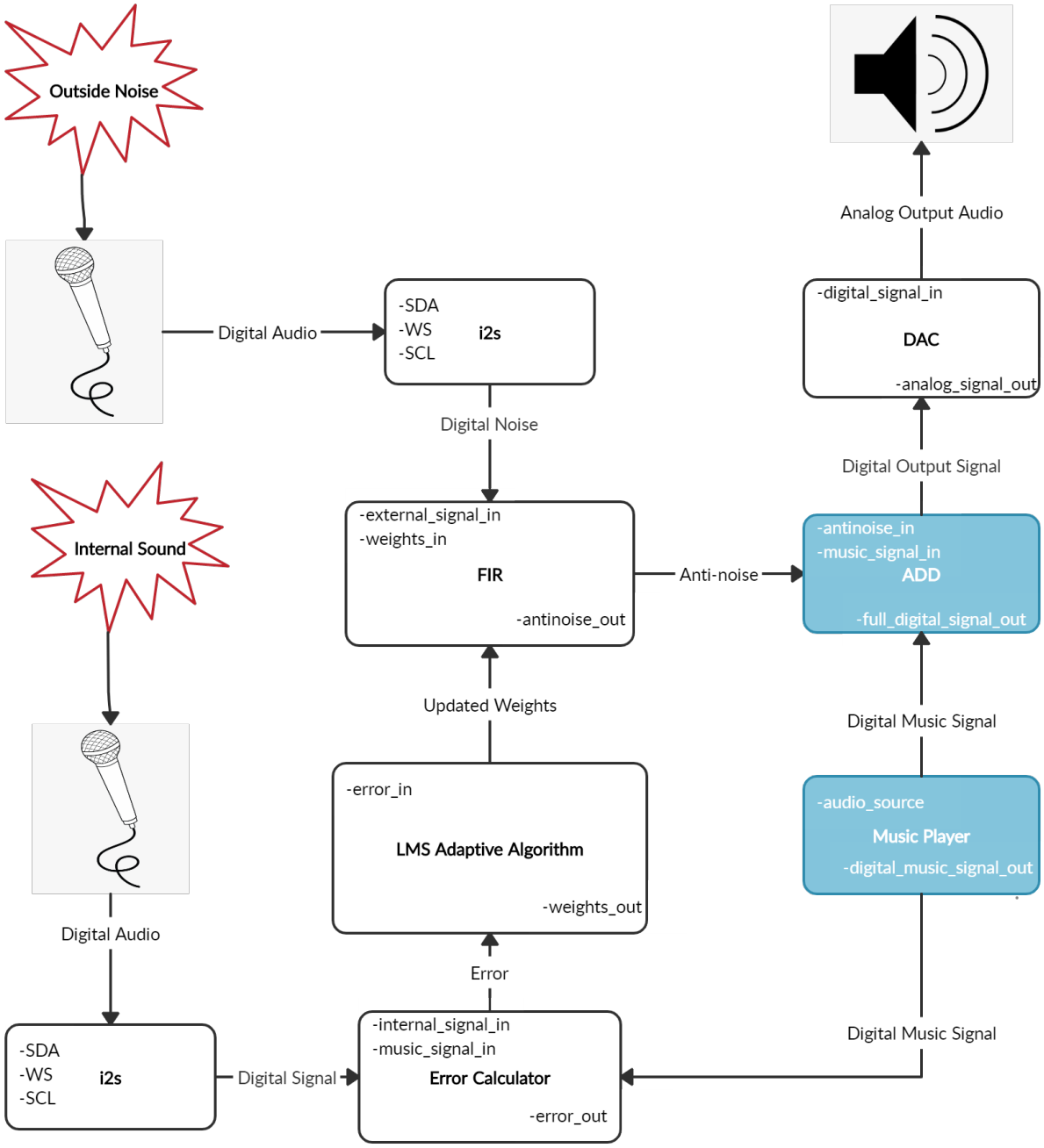

Figure 2: The high-level block diagram for our system.

3 MODULES

3.1 I2S - *IP*

This module will parse the incoming data stream from each of the two microphones. Can be implemented with IP.

Inputs

- SDA - Microphone Data
- SCL - Audio Clock

Outputs

Level of Performance

- 476 LUTs
- 1722 FFs
- 1 36k BRAM

3.2 FIR FILTER - *OWNERS*

The FIR Filter module convolves the external ambient noise with a set of filter coefficients in order to produce an anti-noise signal. This anti-noise signal is later used to cancel out the ambient noise that makes it into the the ear cup. These coefficients are set continuously by the LMS adaptive algorithm module.

Inputs

blah

Outputs

blah

Complexity

blah

Level of Performance

blah

Testing

blah

3.3 LMS ADAPTIVE ALGORITHM

This module uses the LMS adaptive algorithm in order to update the coefficients of the FIR Filter for better noise cancellation. This algorithm takes advantage of the steepest descent optimization method, which analyzes the impact of the previous filter coefficients on the observed error and computes appropriate adjustments. This update is accomplished using the following equation, where $b_k(n)$ is the FIR coefficient for a given k values and a given timestep n, Λ is a predefined step size or learning rate that defines how quickly the filter weights change, and $f(n)$ is the value of the noise signal from the external microphone at a timestep n:

$$b_k(n+1) = b_k(n) + \Lambda e(n) f(n-k)$$

$$k = 0, 1, 2, \ldots, M-1$$

Inputs

blah

Outputs

blah

Complexity

To implement this operation, we will need to store the last M noise inputs from the external microphone along with the full last set of filter coefficients. This will require two BRAMs of size $w_S M$ and $w_C M$ respectively, where $w_S = 16$ is the bit depth of our samples from the external microphone, and w_C is the bit depth of the coefficients.

Level of Performance

blah

Testing

blah

3.4 ERROR CALCULATOR - *OWNERS*

The error calculator computes the difference between the output of the internal microphone (the sound within the ear cup) and the intended final audio. If we reach our stretch goal, the intended audio will be the music being inputted into the system. Otherwise, the goal will be silence. This difference or "error" will be fed into the LMS modules for error correction.

Inputs

blah

Outputs

blah

Complexity

blah

Level of Performance

blah

Testing

blah

3.5 DAC - *OWNERS*

The DAC (Digital-to-Analog Converter) module will be used to transform the final digital output audio into an analog signal that can be played through the speaker.

Inputs

blah

Outputs

blah

Complexity

blah

Level of Performance

blah

Testing

3.6 MUSIC PLAYER - *OWNERS*

This module outputs the source music/audio in digital form (the user should ideally hear a version of this audio with minimal corruption). With the possible extension of Bluetooth-based input audio, this module would be modified to receive its input from a Bluetooth receiver that would be interfaced with the FPGA.

Inputs

blah

Outputs

blah

Complexity

blah

Level of Performance

blah

Testing

blah

3.7 ADD - *OWNERS*

If we reach our stretch goal of the music player integration, we will need a module to merge the output of the FIR—the signal that will cancel the incoming noise—with the music to play. This module will take two signals as an input, and output their sum. It will be very simple.

Each module
-inputs and outputs
-complexity
-level of performance
(eg. number and type of arithmetic operations, size of internal memories, required throughput)
-how it will be tested

Teamwork Distribution
-team member works on which module
-most modules should have a single designer
-testbenches should be different members