**Microprocessor Systems Final Project Report**

Audio Signaling

Shane Conaboy

Justin Yost

ECSE-4790 Microprocessor Systems

December 14, 2015

**Introduction**

After brainstorming several ideas, it was decided that the final project was to see what could be done with an audio signal coming from the headphone port of a computer or phone. One of the initial reach goals was to perform a Fast Fourier Transform (FFT), using the 8051 microcontroller, on the signal and output it to a series of LEDs to recreate the vertical bars found on some audio equipment that light up as the audio places, representing the spectrum. Unfortunately, this goal could not be accomplished in the allotted time period, however the lab became more of an exploration of the received audio signals and how to process them using the 8051.

**Procedure**

The very first part of this lab was taking a pair of headphones and stripping them to find the wires that were being used to transmit the audio signals. As suspected, a single pair of wires was found, and attempts were made to view the signal using an oscilloscope. It was quickly discovered that the wires were insulated and sandpaper was used to strip the insulation. To prevent shorting and ease of handling, solid-core wire was soldered onto the headphone wires, and heat shrink was used to keep everything together. Using a computer headphone port and an online tone generator to obtain a single frequency, the signal was successfully viewed on the oscilloscope.

The next step was to sample the signal using the ADC on the 8051 and process the data. The tone received from the computer was a sine wave centered about 0, meaning there would need to be a DC offset added to have the amplitude always above 0, much like lab 4. However, instead of using a voltage source to add a voltage offset, it was decided that summing amplifiers would be created using op amps. Knowing that getting the audio signal back out of the 8051 using the DAC would require another voltage shift to center the signal at 0 again, using op amps was the most logical solution since they could accommodate both positive and negative voltages as long as they are supplied with such. A general summing amplifier circuit is shown in Figure 1 and the output voltage can be calculated using Equation 1.



Figure : General Summing Amplifier Circuit

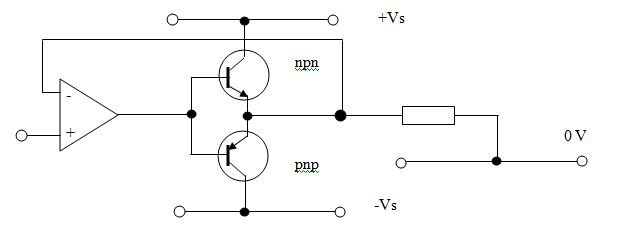
|  |  |  |
| --- | --- | --- |
|  |  | (1) |

The ADC was set up using the internal reference voltage of 2.4V, and it was determined that a DC offset of 1.2V would be used for the audio signal. A voltage divider was set up using the 5V power source, a 3.2k and a 1k resistor to obtain 1.19V at the output, as shown in Equation 2. The output of this voltage divider was buffered using a voltage follower so the load of the summing amplifier circuit would not affect the offset voltage value. The audio signal and the offset voltage were passed into the summing amplifier circuit, using 1k resistors for both Rin and Rf to obtain unity gain, with the output of the circuit fed into the AIN0.0 pin of the 8051.

|  |  |  |
| --- | --- | --- |
|  | 1.19V | (2) |

To verify that the signal could be received back from the 8051, it was output using the DAC set up with a reference voltage of 2.4V. To get the actual audio signal, the 1.2V DC offset needed be taken back out using another summing amplifier with a -1.2V input. The negative voltage was generated by **JUSTIN PUT STUFF HERE.** Instead of making this summing amplifier have unity gain, a 10k potentiometer was used as Rf with 1k resistors used for Rin, which allowed for a variable gain between 0 and 10. The potentiometer added the feature of volume control external to the computer, mimicking the functionality of a stereo.

An external speaker was wired to the output of the final summing amplifier to test if the signal could be successfully processed by the 8051. The online tone generator was used as a basic test to verify that both low and high frequencies could be achieved. After the tone generator, actual music was played from the computer and heard through the speaker. Testing the volume control revealed that the circuit functioned with little distortion at low volumes, however there would be extreme noise introduced as the volume was turned up using the potentiometer. Using the oscilloscope to view the output signal, it was found that the voltage level was peaking at approximately 0.3V. It was determined that since the speaker had such a low internal resistance, approximately 4 ohms measured by a multimeter, it was pulling an amount of current that the op amp could not supply. Several solutions to remedy this problem were researched, and the most popular circuit, shown in Figure 2, involved the use of transistors. Given enough time, this circuit would have been implemented and tested, however with limited time at this point, another common solution was chosen, which was to put a capacitor in series with the speaker. The capacitor would increase the impedance of the circuit, allowing more voltage and less current to be passed across the speaker.



NPN

PnP

Figure : Possible Speaker Driver Circuit

Seeing the similarity between the things done in this lab and lab 4, it was decided to use the Multiply and Accumulator (MAC) to create a filter and distort the audio signal. To create this filter, **JUSTIN INSERT MORE THINGS HERE.** The filter was then tested using both the online tone generator and actual music from the computer.

Unfortunately, at this point in the lab, time had run out and other features were unable to be implemented. Given more time, the FFT with LED display and different communication methods, such as IR communication, would have been added into the project.

**Challenges and Results**

* Each computer/device output a different voltage. The MacBook Pro output the highest voltage at max volume – approximately 2.5V peak-to-peak.
* Changing the volume on the computer affected only the voltage amplitude of the signal.

**Future Improvements**

**Conclusion**

**Appendix A: CAN Bus Transmit Code**

**Appendix B: CAN Bus Receive Code**

**Appendix C: RC Car CAN Arbitration IDs, Functions, and Value Ranges**