**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

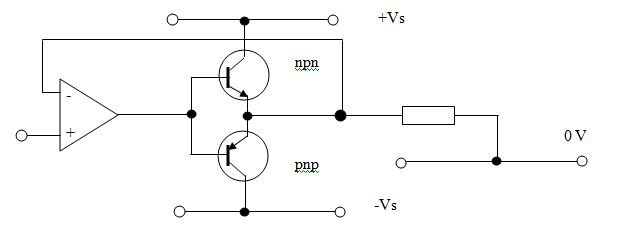
|  |  |  |
| --- | --- | --- |
|  |  | (1) |

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**

When using the oscilloscope to view audio signals received from different devices, it was discovered that each device resulted in a different max voltage output dependent on the set volume. The student Lenovo laptop had a max voltage on the order of tenths of volts, while the MacBook Pro had a max voltage of approximately 2V peak-to-peak. This was attributed to differences in the driving circuits of the laptops, and thus the MacBook was used for the remaining portion of the lab.

The next thing to be observed was how changing the volume on the computer affected the output of the signal. The volume clearly affected the amplitude of the signal, however it was not a one-to-one scale. The volume and signal amplitude appeared to have a logarithmic relationship, as observed when putting the volume at ¾ of the maximum value, the amplitude of the signal was at ½ maximum voltage. A logarithmic scaling would make sense since the loudness of sound is measured in decibels, which are computed on a base 10 log scale.

One of the more complex parts of the project was choosing an op amp to use to offset and possibly amplify the audio signal. There were several different op amps available through the Mercer Lab, some of which were specifically for audio amplification with a default gain of 20. Looking over the datasheets of several amplifiers, it was unclear as to how the audio amplifier would benefit over a normal op amp, so the **INSERT OP AMP NUMBER HERE** op amp, a classic op amp, was chosen for use. Even if the audio amplifier would have allowed for cleaner processing of the signal, the added complexity outweighed the benefits that it would provide within the scope of this project.

Initial testing of the circuit to input the audio signal into the 8051 and output through the DAC proved to be partly successful. The signal was able to be received by the 8051 through the ADC easily, and after solving some wiring issues, the output from the DAC was implemented as well. It was noticed that the final output signal was offset incorrectly, and this issue was traced back to the voltage divider circuit that provided the 1.2V offset at the beginning. This circuit needed to be isolated so the load from the summing amplifiers did not affect its behavior. After passing the voltage divider output through a voltage follower, the output from the DAC was a sine wave centered at approximately 0 as expected.

One puzzling phenomenon that was observed was when the gain on the last summing amplifier was increased using the potentiometer, essentially turning up the volume, the voltage levels measured across the speaker would peak at approximately 0.5V before saturating and cutting off parts of the signal, resulting in a large amount of noise at the output. It was initially unclear as to why this was happening since the supplied voltage to the op amps was approximately -5V to 5V, so the amplifiers should have easily handled a range of -2V to 2V. After some investigation, it was determined that since the speaker had such a low resistance, 4 ohms, the output from the op amp was essentially shorting to ground, causing a large amount of current to flow. The current must have been outside the specifications of the op amp, so power to the speaker was limited. Adding a capacitor in series with the speaker to increase impedance and decrease current flow allowed higher voltage levels, and thereby louder volume levels, to be reached. Several capacitors were tested to find which one provided a good balance between max volume and low noise output, and a capacitor of 47 μF was determined to be the best for the needs of the project.

After audio signals were successfully played through the circuit setup, introducing intentional distortion using a MAC-generated filter was attempted. With limited time available, a rudimentary low-pass filter was implemented using the MAC and tested using both the online tone generator and with music. The filter was seen to work using the tone generator with frequencies past approximately 15kHz, however when listening to music, no significant change was heard. Attempts were made to decrease the cutoff frequency of the filter, however with limited time and understanding of digital filtering techniques, the efforts were unsuccessful.

**Future Improvements**

This project was a great exploratory exercise into the workings of audio signals and some basic processing techniques. Unfortunately, the previous lab, the CAN Bus lab, took longer than expected to complete, cutting into the time allotted for this final project. Given more time, more refined digital filters would be implemented, as well as an FFT on the audio signal and some kind of display module. Also, a lot of time was spent choosing op amps and deciding how the final circuit would be implemented. If this project were to be repeated, more background research on different amplifiers, audio vs. traditional, would be done and the overall circuit would be reworked to obtain a better, more efficient design.

**Conclusion**

This final project facilitated the application of the skills learned from all the previous labs, allowing a more open-ended approach when solving problems. Even though there was little time for preparation and implementation due to the longer CAN Bus lab, it was still a project that was interesting and educational. The basic goal of receiving an audio signal using the 8051 and outputting it to play music was successful, as well as the implementation of minor signal distortion. It was very satisfying to be able to hear actual music after the signal passed through the designed circuit and microcontroller. It was disappointing to run out of time at the end, but overall, the basic project goals were met and music was played through the setup.

**Appendix A: Circuit Schematic**

**Appendix B: Code**

//------------------------------------------------------------------------------------

// final\_project.c

//------------------------------------------------------------------------------------

// The program sampled an audio signal on AIN0.0, passes it through a low-pass

// filter, and outputs the resulting signal using the DAC.

//------------------------------------------------------------------------------------

// Includes

//------------------------------------------------------------------------------------

#include <c8051f120.h>

#include <stdio.h>

#include "putget.h"

//------------------------------------------------------------------------------------

// Global Constants

//------------------------------------------------------------------------------------

#define EXTCLK 22118400 // External oscillator frequency in Hz

#define SYSCLK 22118400

#define BAUDRATE 115200 // UART baud rate in bps

//------------------------------------------------------------------------------------

// Function Prototypes

//------------------------------------------------------------------------------------

void main(void);

void SYSCLK\_INIT(void);

void PORT\_INIT(void);

void UART0\_INIT(void);

void ADC\_INIT(void);

void INTERRUPT\_INIT(void);

void DAC\_INIT(void);

void TIMER\_INIT(void);

void MAC\_INIT(void);

// Global Variables

char start\_conversion = 0; // Flag to indicate when to start the ADC

unsigned int dig\_val = 0;

//------------------------------------------------------------------------------------

// MAIN Routine

//------------------------------------------------------------------------------------

void main(void)

{

unsigned char adcValH[5];

unsigned char adcValL[5];

unsigned int result;

unsigned int results[2];

unsigned short int analogval;

unsigned char analoghi, analoglow;

float VREF = 3;

WDTCN = 0xDE; // Disable the watchdog timer

WDTCN = 0xAD;

PORT\_INIT(); // Initialize the Crossbar and GPIO

SYSCLK\_INIT(); // Initialize the oscillator

ADC\_INIT(); // Initialize ADC0

DAC\_INIT();

MAC\_INIT();

adcValH[4] = adcValH[3] = adcValL[4] = adcValL[3] = adcValH[2] = adcValH[1] = adcValL[2] = adcValL[1] = 0;

result = 0;

results[1] = results[0] = 0;

while(1)

{

SFRPAGE = ADC0\_PAGE;

AD0INT = 0; // Clear the "conversion done" flag

AD0BUSY = 1; // Start A/D Conversion

while (AD0INT == 0); // Wait for the conversion to finish

analoglow = ADC0L; // Read the low byte

analoghi = ADC0H; // Read the high byte

analogval = analoghi<<8 | analoglow;

// Update the variables in the filter equation

adcValH[4] = adcValH[3]; // x(k-2) high byte

adcValH[3] = adcValH[2]; // x(k-1) high byte

adcValL[4] = adcValL[3]; // x(k-2) low byte

adcValL[3] = adcValL[2]; // x(k-1) low byte

adcValH[2] = adcValH[1]; // x(k-2) high byte

adcValH[1] = adcValH[0]; // x(k-1) high byte

adcValL[2] = adcValL[1]; // x(k-2) low byte

adcValL[1] = adcValL[0]; // x(k-1) low byte

results[1] = results[0];

results[0] = result;

adcValH[0] = analoghi; // x(k) high and low bytes

adcValL[0] = analoglow;

SFRPAGE = MAC0\_PAGE;

MAC0CF |= 0x08; // Clear MAC

// Load the MAC with the correct values and compute the filter equation

MAC0AH = 0x20;

MAC0AL = 0x00;

MAC0BH = result>>8;

MAC0BL = result;

MAC0AH = 0x20;

MAC0AL = 0x00;

MAC0BH = results[1]>>8;

MAC0BL = results[1];

MAC0AH = 0x10;

MAC0AL = 0x00;

MAC0BH = adcValH[2];

MAC0BL = adcValL[2];

MAC0AH = 0x20;

MAC0AL = 0x00;

MAC0BH = adcValH[1];

MAC0BL = adcValL[1];

MAC0AH = 0x10;

MAC0AL = 0x00;

MAC0BH = adcValH[0];

MAC0BL = adcValL[0];

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE;

SFRPAGE = MAC0\_PAGE; // Delay with any dummy command

result = (int)MAC0RNDH<<8 | MAC0RNDL; // Read the result from the rounding engine

SFRPAGE = DAC0\_PAGE; // Output the result through the DAC

DAC0L = result;

DAC0H = result>>8;

}

}

//------------------------------------------------------------------------------------

// SYSCLK\_Init

//------------------------------------------------------------------------------------

// Initialize the system clock to use a 22.1184MHz crystal as its clock source

void SYSCLK\_INIT(void)

{

int i = 0;

char SFRPAGE\_SAVE = SFRPAGE;

SFRPAGE = CONFIG\_PAGE;

OSCXCN = 0x67;

for (i = 0; i < 3000; i++); // Wait 1ms for initialization

while ((OSCXCN & 0x80) == 0);

CLKSEL = 0x01;

OSCICN &= ~0x80;

SFRPAGE = SFRPAGE\_SAVE;

}

//------------------------------------------------------------------------------------

// PORT\_Init

//------------------------------------------------------------------------------------

// Configure the Crossbar and GPIO ports

void PORT\_INIT(void)

{

SFRPAGE = CONFIG\_PAGE;

WDTCN = 0xDE; // Disable watchdog timer.

WDTCN = 0xAD;

XBR0 = 0x04;

XBR1 = 0x04; // Enable external interrupt

XBR2 = 0x40; //Enable crossbar

P0MDOUT = 0x01;// Set TX0 pin to push-pull, RX and Push Button to open drain

P0 = 0x06; // Inputs RX and Push Button set to high impedence

SFRPAGE = LEGACY\_PAGE;

IT0 = 0; // /INT0 is edge triggered, falling-edge.

}

//------------------------------------------------------------------------------------

// UART0\_Init

//------------------------------------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1

void UART0\_INIT(void)

{

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Mode 1, 8-bit UART, enable RX

SSTA0 = 0x10; // SMOD0 = 1

SFRPAGE = TIMER01\_PAGE;

TMOD &= ~0xF0;

TMOD |= 0x20; // Timer1, Mode 2, 8-bit reload

TH1 = -SYSCLK/(BAUDRATE\*16); // Set Timer1 reload baudrate value T1 Hi Byte

CKCON |= 0x10; // Timer1 uses SYSCLK as time base

TL1 = TH1;

TR1 = 1; // Start Timer1

SFRPAGE = UART0\_PAGE;

TI0 = 1; // Indicate TX0 ready

}

void ADC\_INIT(void)

{

SFRPAGE = ADC0\_PAGE;

AMX0CF = 0x00; // All input ports set to single-ended mode

AMX0SL = 0x00; // Select port AIN0.0 for inpit

ADC0CF = 0x20; // ADC clock rate < 2.5MHz, internal gain of 1

ADC0CN = 0X00;

REF0CN = 0X02; // Voltage reference from VREF0 pin; enable bias generator and disable reference buffer

AD0EN = 1; //Enable ADC

}

void DAC\_INIT(void)

{

SFRPAGE = DAC0\_PAGE;

DAC0CN = 0x80;

}

void INTERRUPT\_INIT(void) {

EA = 1;

EX0 = 1; //External interrupts

}

void TIMER\_INIT(void) {

TMOD = 0x01; // Timer0 in 16-bit mode

CKCON = 0x08; // Timer0 uses sysclk

TR0 = 1; // Enable Timer0

}

void MAC\_INIT(void) {

SFRPAGE = MAC0\_PAGE;

MAC0CF = 0x1A; //MAC mode, fractional mode, rounding will not saturate, reset MAC, shift right

}