Embedded Signal Processing System

ECE230

Intro to Embedded Systems

Final Project

Winter 2019-2020

Instructor: Dr. Zak Estrada

Nick von Bulow

Xinyu Shen

Feb. 24rd, 2020

Table of Contents

Introduction

Overview of Project Features

We created an embedded signal processing system with an LCD screen and an audio input and output. The purpose of this project is to demonstrate a mastery of the material we have learned throughout ECE230 (and ECE180).

The system has two modes: image and audio. In the image mode, the system will read an image from the SD card and display it on the LCD screen. In the audio mode, the system will take samples from the audio input. The system will then apply different filters to them depending on the user input, and users are able to listen to the filtered audio signal.

Criteria for ECE230 Final Projects

There are certain minimum criteria that we have to meet in our projects. Specifically, the requirements are listed below:

1. The project must use the PIC16f887 (or other PIC microcontroller with approval)
    a. Our project uses the dsPIC33CK256MP202 microprocessor which has been approved by the instructor.
2. The embedded circuit must include at least one sensor (input)
    a. Our project takes input from an audio source, push button, and SD card.
3. The embedded circuit must include at least one actuator (output)
    a. Our project outputs to the LCD, audio sink, LED, and SD card.
4. The embedded circuit must make use of at least one timer.
    a. Our system uses the PTG timer to trigger DMA transfers from the ADC to memory and memory to DAC.
5. The embedded circuit must make use of at least one interrupt.
    a. The PTG module sends an interrupt every 512 samples so we can swap the buffers and process the data.

User Guide

This user guide is intended to help the user operate our system.

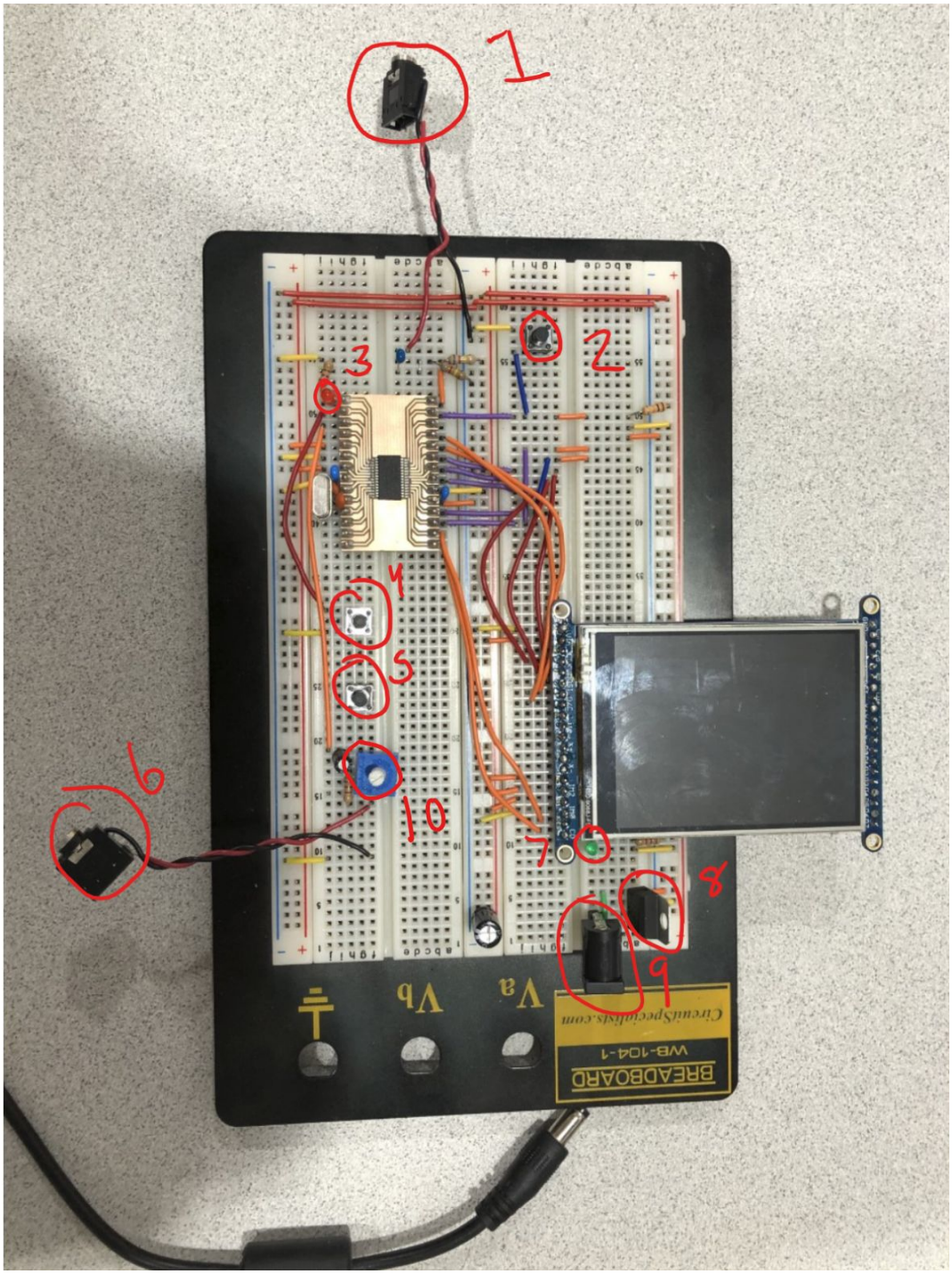

Figure 2.1 Photo of the whole system

Before you Start

Check all components so that they are complete and intact. Bad components may lead to malfunction of the entire system and possibly damage your own equipment, such as headphones. DO NOT touch the voltage regulator (#8) while the system is in operation. It WILL burn you.

Power Up

Simply plug in the system (#9). The green LED (#7) near the regulator (#8) indicates whether the power is plugged in. On initialization, the system boots into audio mirror mode, i.e. the audio detected on the DAC is played back on the output without any filter applied.

Switching Modes

With the power input (#9) in the lower right corner, there are three buttons: one in the upper right corner (the reset button, #2) and two in the bottom center. The button on the top (#4) is for switching between different equalizers and the button on the bottom (#5) is used for switching between audio mode and image mode. The red LED (#3) near the top indicates the mode. It turns off in image mode and goes on in audio mode.

Audio Mode

When the system operates in audio mode, it will takes in input from an audio source (i.e. your computer) you connected to the input audio jack (#1) and outputs the processed music through the output device you connected to the output audio jack (#6). Pressing the top button (#4) will make the system switch between different equalizers.

The system starts with an normal equalizer where the output should sounds exactly like the input. It switches between the following equalizers in a loop: normal, popular, blues, classical, jazz, electronic, superbass, lowpass, highpass, middlepass.

The LCD screen will display the frequency spectrum of the corresponding equalizer. The seven bands correspond to 63, 160, 460, 1k, 2.5k, 6.5k, and 16k, in Hz. The picture at the end of the user guide is an example output of the LCD of the “superbass” equalizer.

Also, you can change the volume of the output by rotating the potentiometer (#10, rotate the white part, not the whole potentiometer!). Clockwise decreases the volume while counterclockwise increases it.

Image Mode

When the system operates in image mode, it will read a predefined file called “LOGO.BMP” which is a 16 bit 5-6-5 bitmap from the SD card and display it on the screen.

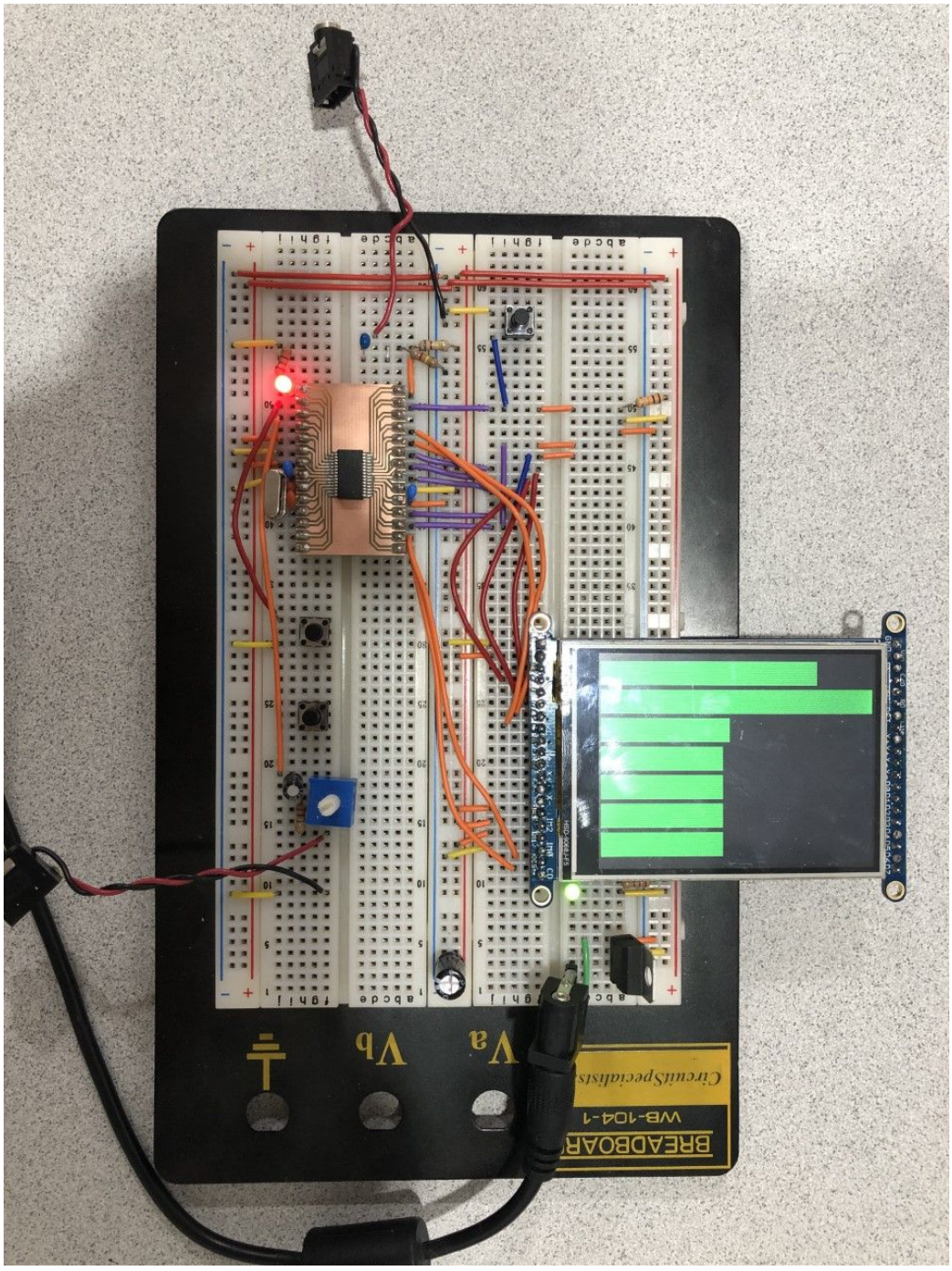

Figure 2.2 Sample output of LCD of superbass equalizer

Hardware Design and Implementation

Circuit Diagram

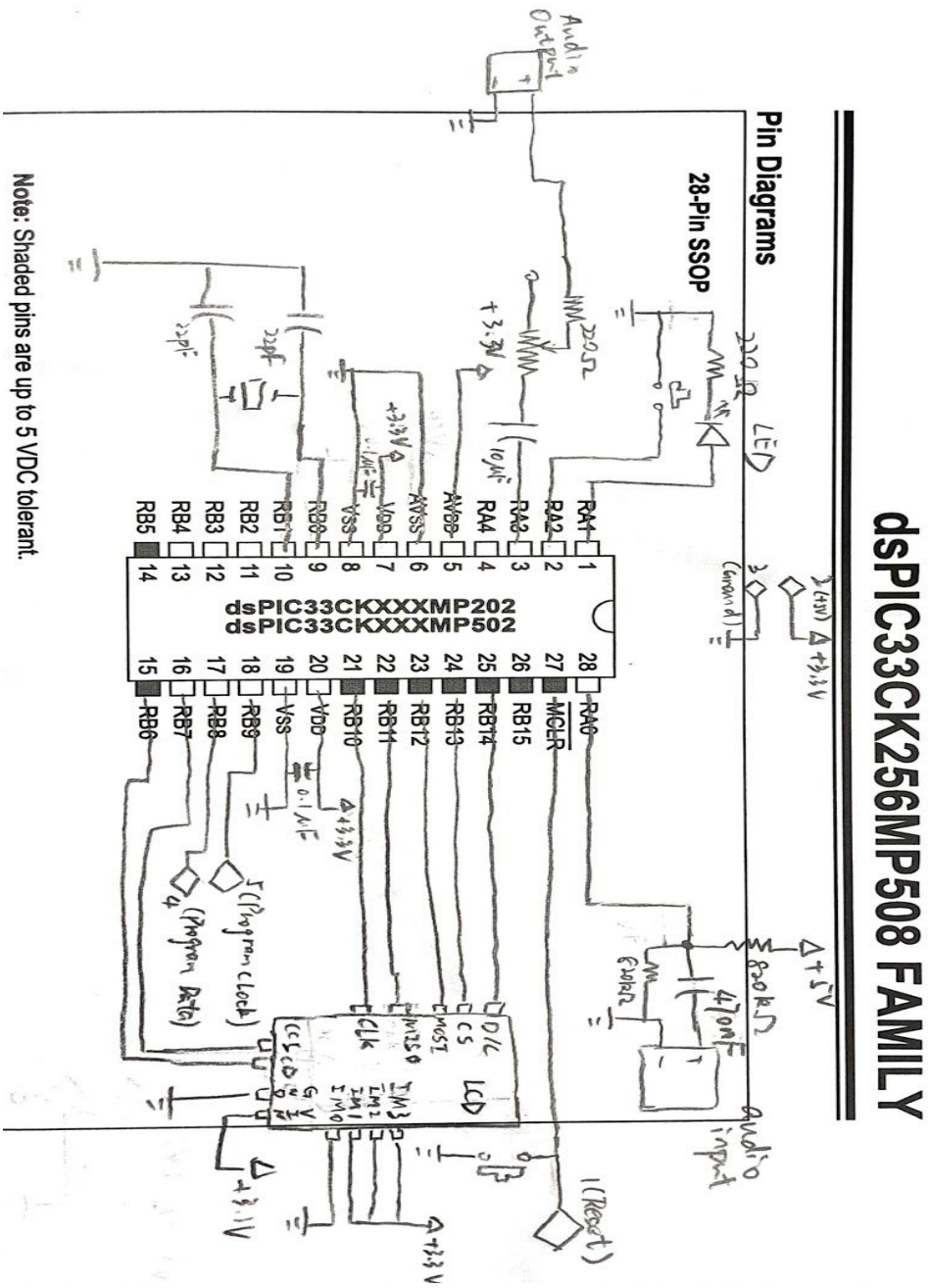

Figure 4.1 Overall Circuit Diagram

Pin Diagram and Assignments

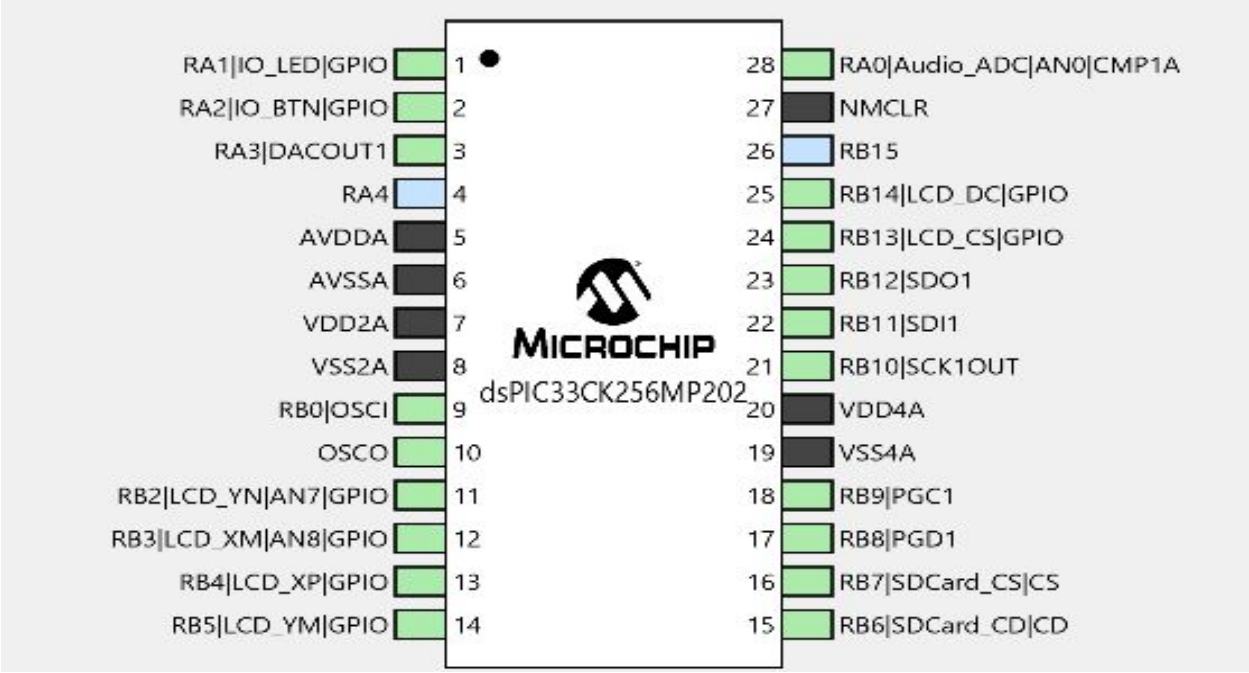

Figure 4.2 Pin Diagram

Table 4.1 Pin assignments for the system

Module	Direction	Pin Name	Description
Clock	in	OSCI	8MHz crystal oscillator, which is then internally multiplied up to 200MHz by a PLL.
	out	OSCO	
LCD	out	LCD_DC	Connected to the LCD D/C pin, which indicates whether data or command is sent to the LCD.
	out	LCD_CS	SPI Chip Select for the LCD
	out	SDO1	SPI MOSI. This is used for sending data to the LCD and the SD card.
	in	SDI1	SPI MISO. This is used for receiving data from the LCD and the SD card
	in/out	SCK1OUT	SPI clock for the LCD and SD card

SD Card	out	SDCard_CS	SPI Chip Select for the SD card
	in	SDCard_CD	SD card detect. Indicates whether a card is inserted or not.
Audio	in	Audio_ADC	The audio input of the system. Can be connected to computers or other audio input signal generators.
	out	DACOUT1	The audio output of the system. Can be connected to speakers, headphones, or other audio output devices..
Other	out	IO_LED	Connected to an indicator LED.
	out	IO_BTN	Connected to a button. The button is used for switching between different modes and filters of the audio output.

Software Design and Implementation

Audio Mode

The following figure is an overview of how the system flows when it is operating in the audio mode.

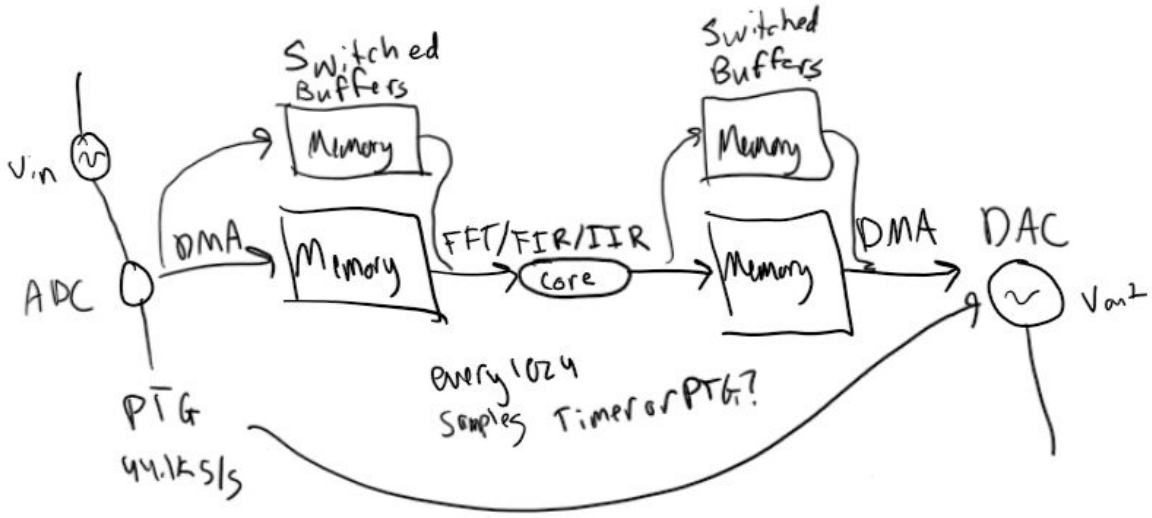

Figure 5.1 Software Flow for Audio Mode

LCD Module

Our system uses an ILI9341 LCD screen to display pictures from SD card and frequency spectrum of the audio outputs. We adapted parts of the Adafruit library for the LCD to create our own for the LCD and add several functions that we use in our project. We also preload an image (LOGO.bmp) to the SD card on the LCD so when one enters image mode, the LOGO will shown on the LCD.

PTG Timer and Interrupt

Our system utilizes the Peripheral Trigger Generator (PTG) ADC conversions, which in turn trigger DMA transfers to the memory. The PTG module works like a finite state machine which can be seen from the diagram and code.

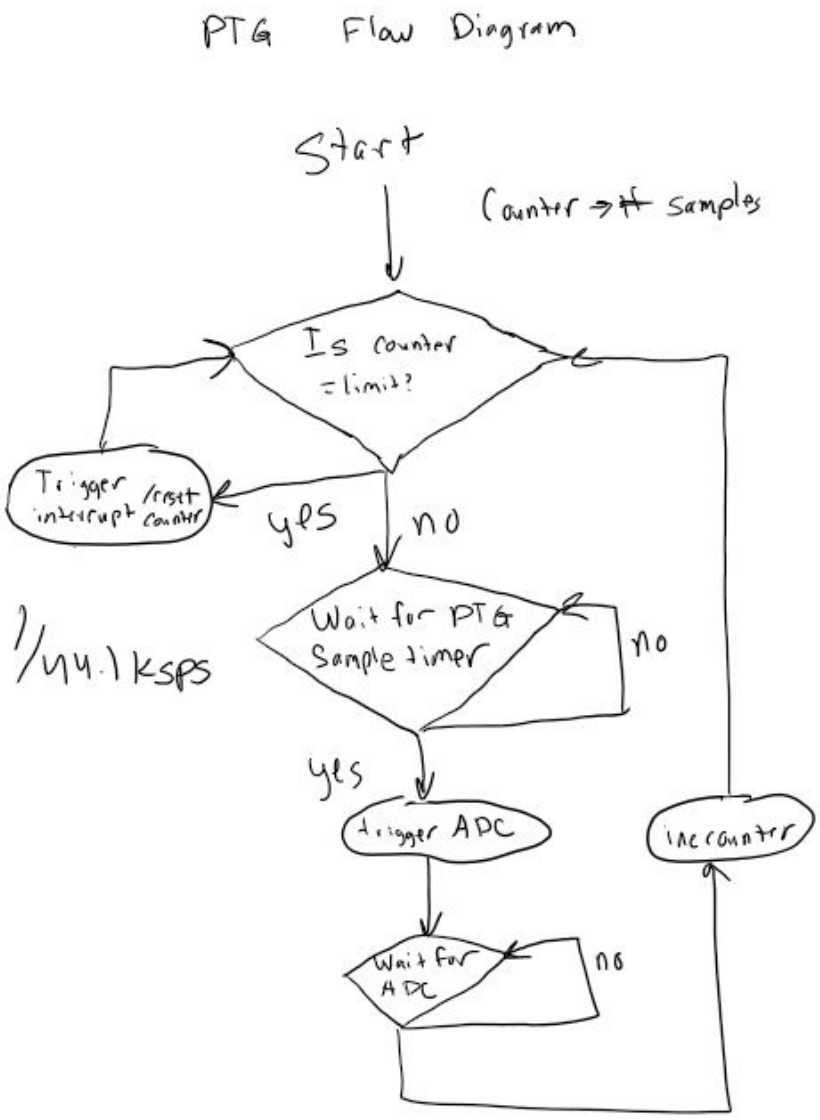

Figure 5.2 PTG Flow Diagram

```
PTG_STEP0 = PTGJMPC0 | 0x4;     // Jump to STEP4
PTG_STEP1 = PTGIRQ | 0x0;      // Generate PTG IRQ 0
PTG_STEP2 = PTGCTRL | 0xb;     // Wait for the software trigger (positive edge, PTGSWT = 0 to 1)
PTG_STEP3 = PTGJMP | 0x0;      // Jump to STEP0
PTG_STEP4 = PTGCTRL | 0x8;     // Wait for PTG Timer0 to match PTGT0LIM
PTG_STEP5 = PTGTRIG | 0xc;     // Trigger for ADC Sample Trigger
PTG_STEP6 = PTGWHI | 0xd;      // Trigger Input ADC Done Group Interrupt
PTG_STEP7 = PTGJMP | 0x0;      // Jump to STEP0
```

Figure 5.3 PTG Code Snippet

DMA

The DMA controller is not explicitly triggered by our code. Rather, it listens for triggers from the ADC module, and at that point, it will copy the value in the ADC buffer to the next address in memory, all without interrupting the actual CPU. This results in much more efficient code that can still reliably sample at a given rate. Upon a PTG interrupt, the CPU core will reset the DMA controller's address counter to the next unused buffer and then do the ADC input and DAC output.

Filter

We did not reinvent the wheel when it came to implementing the FIR filters. Microchip have created a highly optimized DSP library themselves for the dsPIC series which we opted to use. We used MATLAB to design the bandpass filters used to change the sound effects. There are a total of 7 filters, corresponding to the 7 frequencies mentioned before in the user guide. These filters are then combined with different gain for each to create different equalizers. Below we have placed frequency response plots of each of the frequencies and the sum of all these filters:

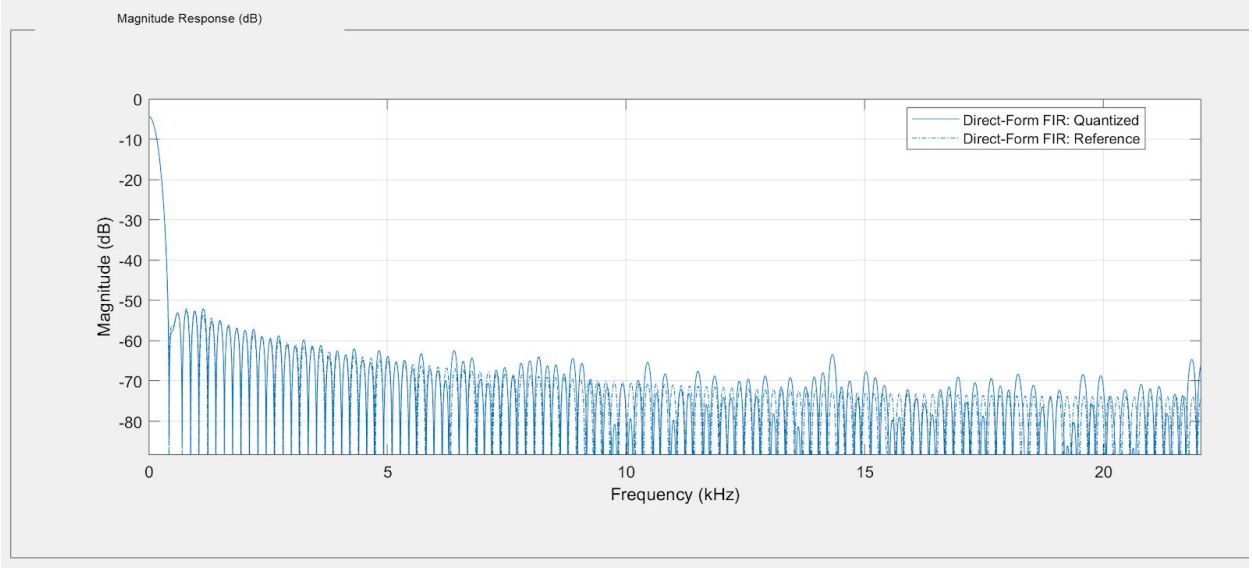

Figure 5.4 63Hz bandpass filter

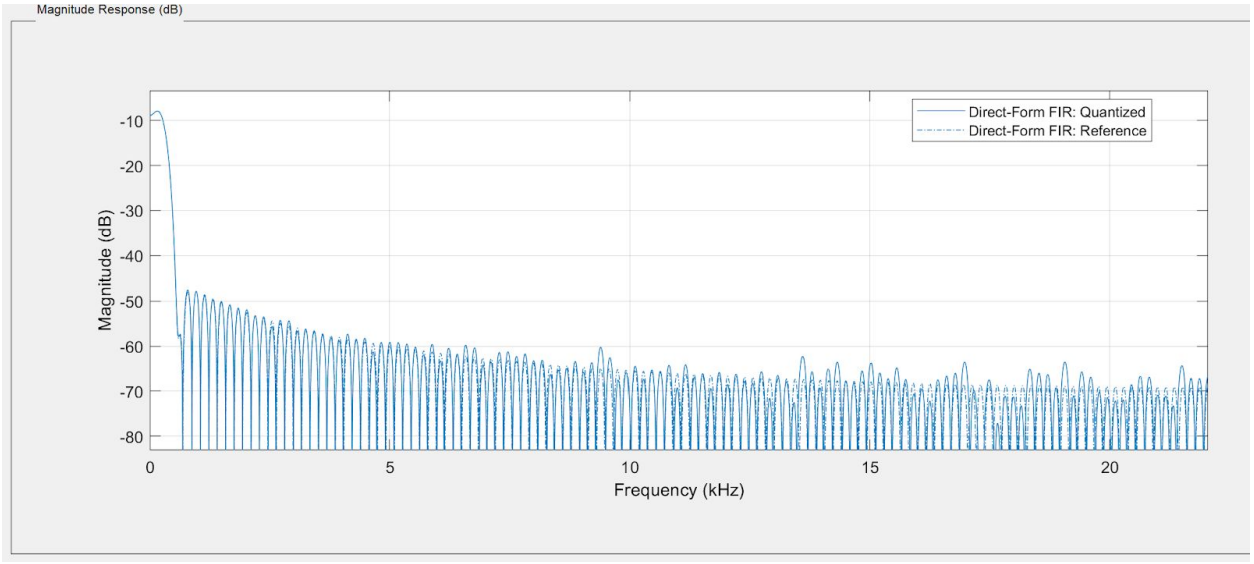

Figure 5.5 160Hz bandpass filter

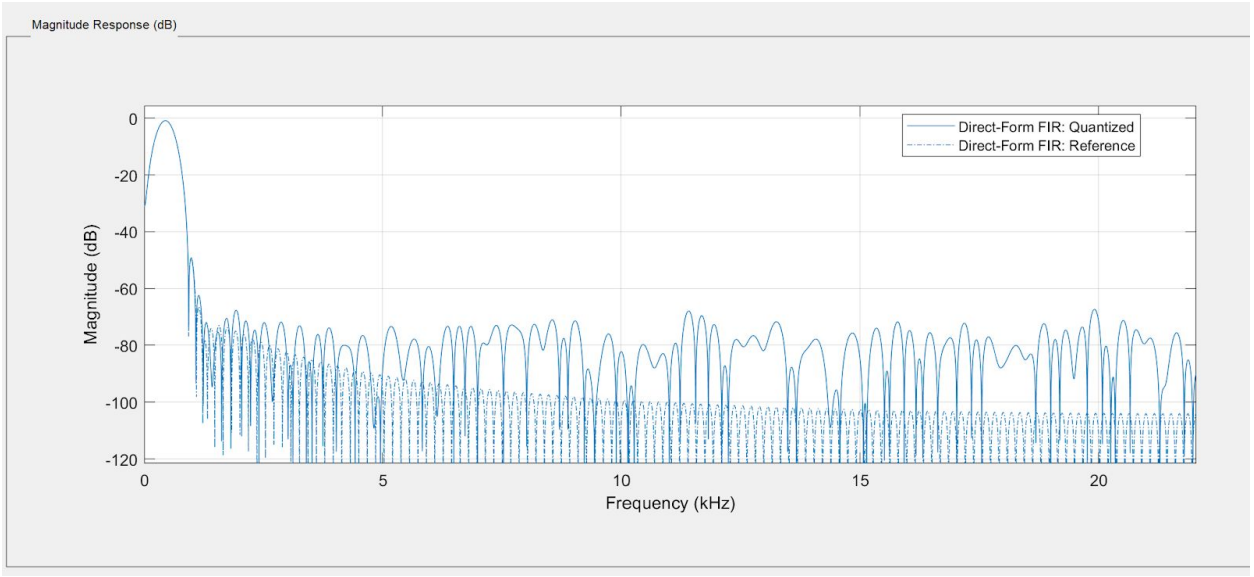

Figure 5.6 460Hz bandpass filter

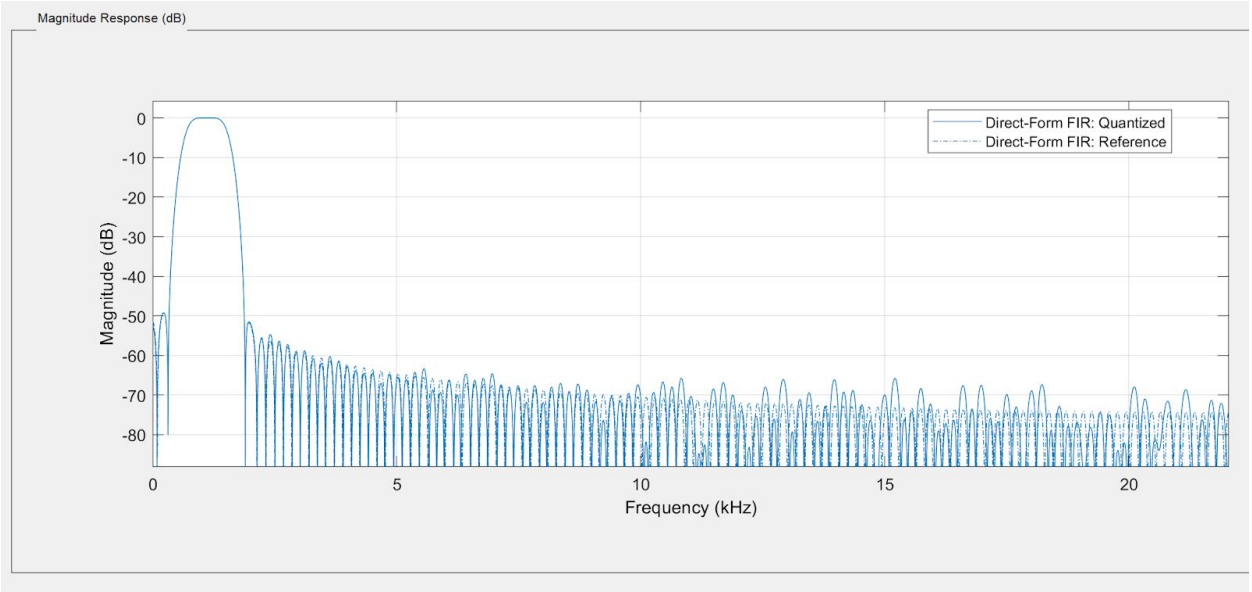

Figure 5.7 1kHz bandpass filter

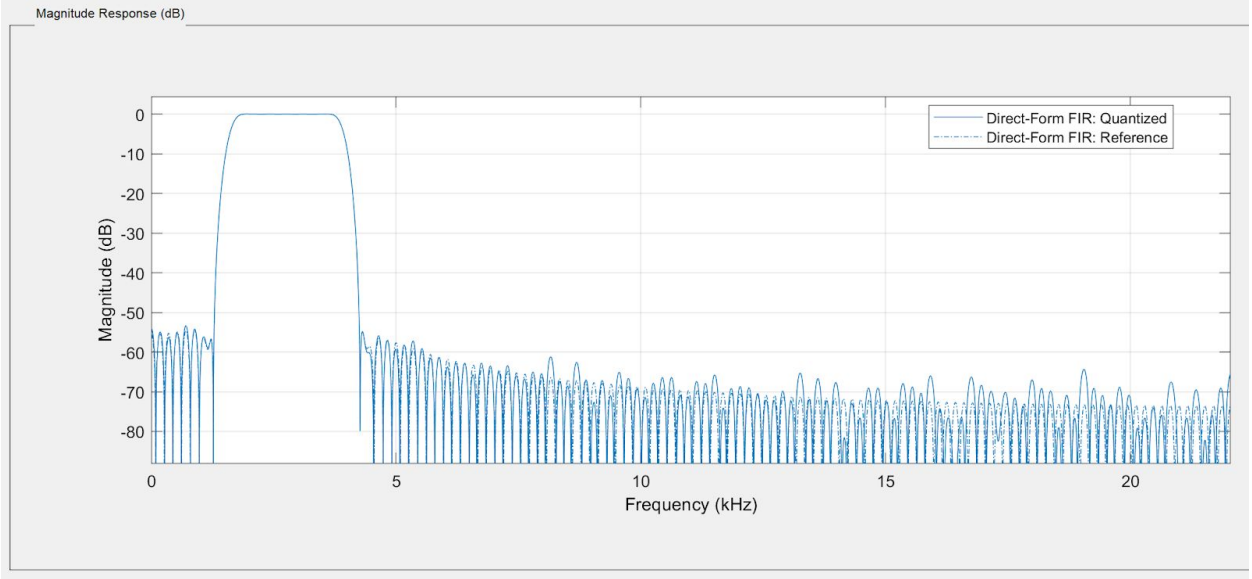

Figure 5.8 2.5kHz bandpass filter

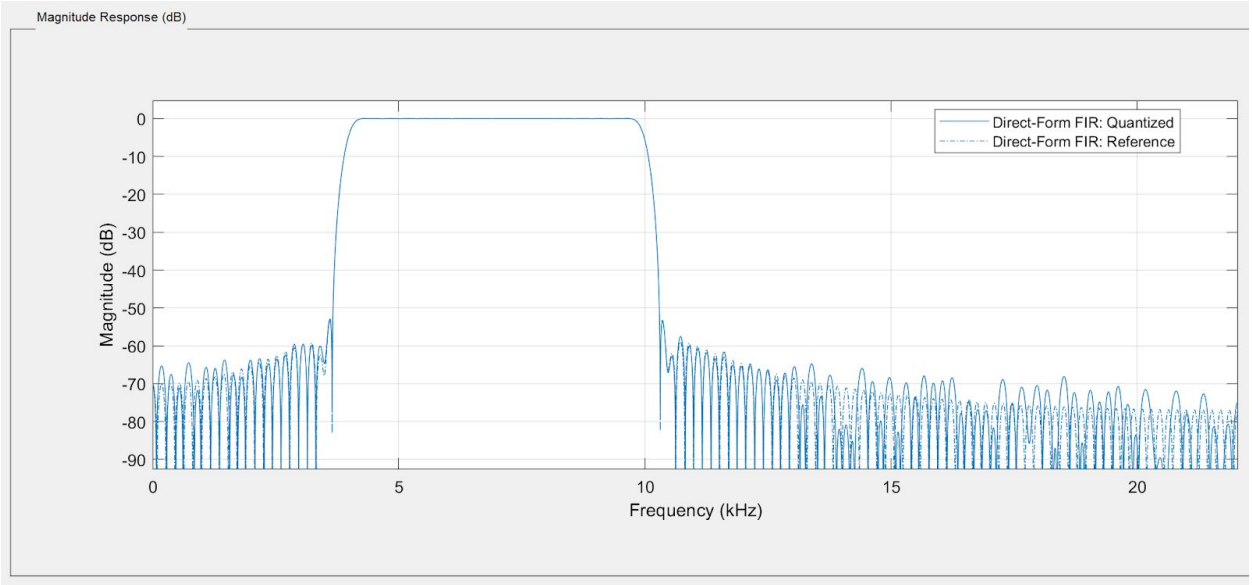

Figure 5.9 6.5kHz bandpass filter

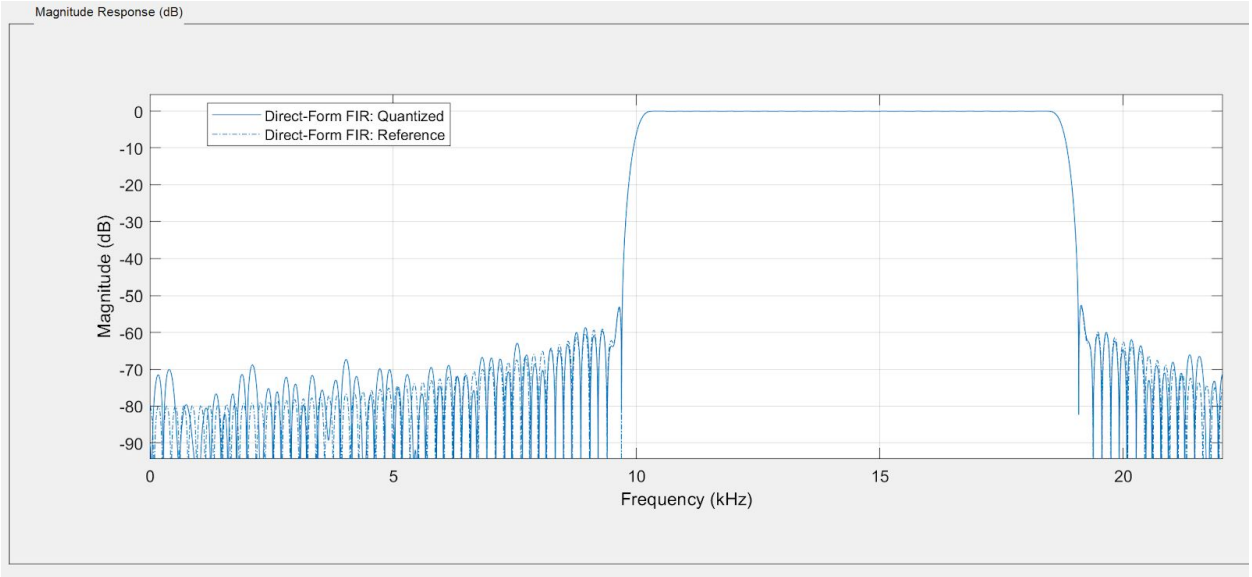

Figure 5.10 16kHz bandpass filter

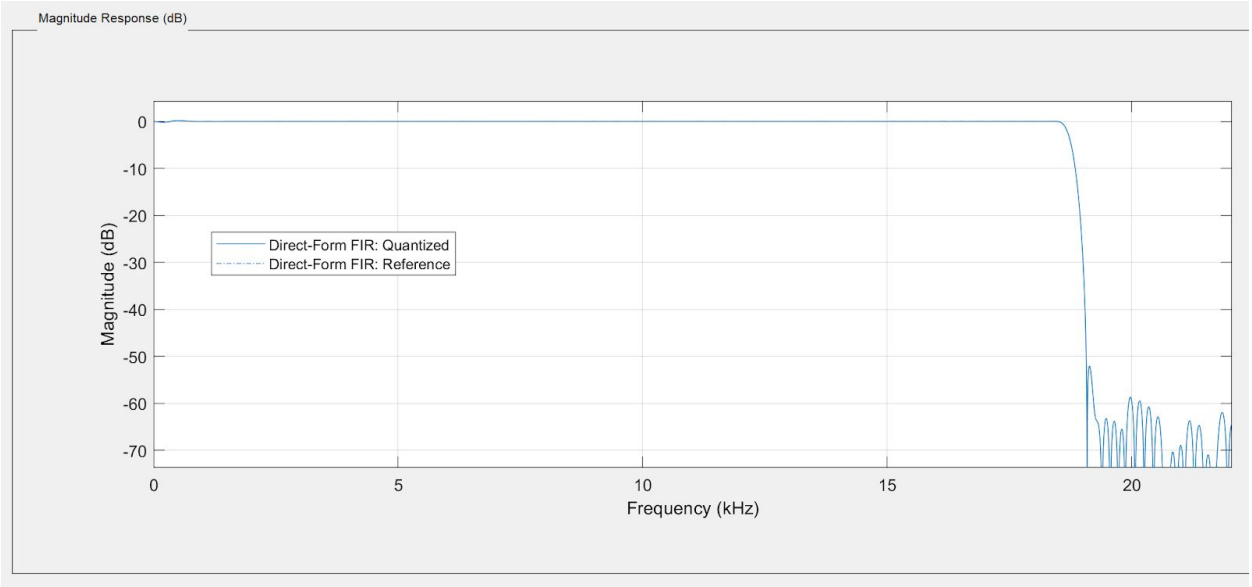

Figure 5.11 All pass filter which is the sum of the above filters

Test Plan and Test Results

LCD Module

The LCD module did not work when we first tried to write images to it. We connect our circuit to the digilent board and use the logic analyzer to look at the D/C, MOSI, MISO, CS, and CLK bits. We first found out that the data bits were lost and after we fixed that, we found that the D/C bit is not correct when we are trying to send a command. It turned out that we needed a delay after we set the D/C bit so that there is enough time for us to send the actual command data bit.

Once the timing issue was corrected, the LCD worked without flaw (except that it was very slow, so we cranked up the clock speed and SPI speed).

PTG Module

We utilized the Microchip Code Configurator to help generate the essential header files and the configurations. We were unable to trigger the PTG interrupt in the first place. We read the documents and tried with some different configurations of the PTG and eventually got it to work.

This is verified by adding a breakpoint in the interrupt function and runs the program to see if the interrupt is actually triggered.

ADC/DAC

These are associated with PTG and DMA. We implemented these after we verify that the PTG interrupt and the DMA works correctly. We use the function generator in the D115 lab as an audio input and use the oscilloscope to capture the input and output waveforms of our system.

The conversion between analog and digital values are relatively simple and we managed to get them working. We can see on the scope that the output is the same as the input with some delay that we expected.

We change the input type over different functions (sine, square, ramp, etc.) and they all work correctly. Concluding that the ADC/DAC works correctly up to this point.

Get Current Working Buffer

We want to use several buffers so that when the audio output is using one of the previous input buffer, we can write to another buffer to speed up the processing. There were issues when we tried to switch the buffer but it jumps around, converting the wrong buffer.

We added breakpoints to our code where we switched buffer and monitors the current buffer count for both the input and output buffer. We intended to use the input buffer and output buffer reversely but it was not. It was using the same buffer for both inputs and outputs. We later found that we are missing “+1” and “-1”s for calculating the input and output buffers.

After changing the code, we try running the system with the same configuration and setup as the ADC/DAC test (with function generators and oscilloscope). We verify that the outputs are still the same as the input with some delays. We concluded that getting the current working buffer works correctly now.

Filter

Initially when implementing the FIR filter, we needed to make sure that the code didn’t crash. The dsPICs have dual-port memory that allow the CPU to access two locations in memory simultaneously, but in different banks. At first we had some trouble finding documentation about how to initialize the “X” and “Y” memory for use by the DSP code. However, once the variables were correctly allocated and initialized as per the documentation, we had no further issues with the FIR filter.

Integration

After finishing all parts of our system, we need to do an integration test. We connected the input audio jack to our computer and the output jack to a speaker. We powered up the system and played a song from the computer. The output from the speaker sounds like the original song. We then switched between the different equalizers and listened for any differences between the outputs. There are slight differences between the equalizers but the effect is greater when it comes to the lowpass/highpass/bandpass equalizers as they create the most drastic change.

We then tried switching between the two modes. The logo image loads correctly when we switched to the image mode and the audio mode operated correctly when we switched back.

We concluded that our system worked correctly as a whole.

Bill of Materials

Part	Quantity	Source	Price
dsPIC33CK256MP202	1	Digikey	$3.00
3.3 V Regulator	1	Parts Room	$0.15
SD Card	1	Pre-Owned	$0.00
LCD Screen (ILI9341)	1	Pre-Owned	$0.00
0.1 µF Capacitor	2	Lab Kit	$0.00
10 µF Capacitor	1	Lab Kit	$0.00
0.47 µF Capacitor	1	Lab Kit	$0.00
22 pF Capacitor	2	Lab Kit	$0.00
Crystal Oscillator	1	Lab Kit	$0.00
Potentiometer	1	Lab Kit	$0.00
220 Ω Resistor	2	Lab Kit	$0.00
820 kΩ Resistor	2	Parts Room	$0.30
Audio Jack	2	Parts Room	$1.50
Misc Parts (Wires, LED, pushbuttons)	-	Lab Kit	$0.00
		Total:	$4.95

References and Acknowledgements

During the development of the embedded signal processing system, we refer to the following documents:

- dsPIC33CK256MP508 Family Data Sheet
- Microchip DSP Library Reference (included with XC16 compiler)
- Adafruit ILI9341 Arduino Library (https://github.com/adafruit/Adafruit_ILI9341)

Thank you to Mark Crosby for designing and soldering our dsPIC breakout board for us!