Wah Pedal Project

Eric Walden & CJ Tulowiecki



Contents

[Summary 3](#_Toc481843645)

[Objectives 3](#_Toc481843646)

[Hardware 3](#_Toc481843647)

[Platform 3](#_Toc481843648)

[Design 4](#_Toc481843649)

[Software 4](#_Toc481843650)

[Platform 4](#_Toc481843651)

[Design 4](#_Toc481843652)

[Test Results 5](#_Toc481843653)

[Lessons Learned 5](#_Toc481843654)

[Final Notes 6](#_Toc481843655)

# Summary

The project began with the purchase of a broken Dunlop Cry Baby “wah” pedal. This is an old analog pedal that had become the gold standard for “wah” pedals over time and are still widely used. However, the pedal was bought simply because it could provide us with a completely assembled pedal enclosure for our project. The analog circuitry (aside from the potentiometer and kill switch) was ripped out so that it could be replaced by embedded processing. A Teensy 3.2 with additional audio board extension provided the necessary processing power and audio fidelity within the given size constraint. Code was developed in Arduino to provide the “wah” effect. The Teensy’s IO was then connected to the potentiometer and kill switch so that the “wah” effect could be modulated or turned off completely. At this point the Teensy’s power regulator blew and we had to use an external regulator to power the board. The embedded system was installed inside of the pedal enclosure. From there experimentation with the filter sweep and resonance characteristics was performed to find the best sounding “wah” effect.

# Objectives

The objective of this project is to design and build a digital “wah” pedal effect that can be implemented into a guitar audio processing chain. A “wah” pedal consists of a filter that sweeps its cutoff frequency in response to the movement of the expression pedal by the user. A potentiometer will be used to sense the position of the expression pedal. Different filter models can be used that provide different sound characteristics.

# Hardware

### Platform

To be successful in this project, we needed a microcontroller small enough to fit inside the pedal enclosure with at least 3 ADC channels to sample the stereo audio in and the potentiometer voltage (2 if the audio is mono). We also needed 2 DAC channels to output the stereo audio waveforms (1 if mono). The last requirement was that both the ADCs and the DACs must have a minimum 16-bit precision and 44.1 KHz sampling rate, in order to have the proper audio fidelity that is up to the standards of today’s audio production.

The platform we selected meets all of these specifications: the Teensy 3.2. The Teensy 3.2 features a 72 MHz ARM Cortex-M4 microcontroller with 64KB of RAM and 256KB of flash memory and operates at 3.3V logic. The Cortex-M4 also has DSP instructions which provide real-time FFT spectrum analysis. The device dimensions are surprisingly small: 1.4” x 0.7”, smaller than a credit card, and it is compatible with the Teensy Audio Adaptor Board, which stacks right on top of the Teensy, providing the high-quality ADCs and DACs we require at a 16-bit resolution and 44.1 KHz sample rate. With a combined price of $35, this combo provides high performance, large memory, and plentiful resources to implement the wah pedal.

### Design

The hardware was designed to be easily assembled while providing impressive performance and feel. The purchase of a cry baby “wah” pedal enclosure took care of the external analog hardware needed to complete the project such as a potentiometer and kill switch. The Teensy audio board extension comes with 16 bit 44.1 kHz audio inputs and outputs with line level amplifiers which provided the board with enough fidelity and power to make a clean sounding effect. The Teensy board was small yet powerful enough to perform the desired filtering.

# Software

### Platform

The Teensy 3.2 only supports Arduino, so this became the software platform for the project. However, Arduino had advantages such as quick prototyping and intuitive data communication. Interfacing between the Teensy and the audio board

### Design

The overall structure of our design is an infinite loop that continuously polls the angle of the pedal (or the potentiometer voltage) and the stomp switch. If the stomp switch is in the active state, the voltage across the potentiometer is sampled by an ADC and mapped to a range of frequencies that were predetermined to act as the range of frequencies of which the cutoff could exist at. This value is then applied to the cutoff frequency of the filter, changing as the angle of the pedal changes. If the stomp switch is not in the active state, then the low-pass filter is “turned off,” leaving the input audio unchanged.

The design of our software takes use of the libraries included with the Teensyduino API. Most importantly, the Audio library. This contains numerous classes written in C++ to provide easy implementation of audio system designs that use the Teensy Audio Adaptor Board. There are four, most-important classes used to implement our design: AudioInputI2S, AudioOutputI2S, AudioConnection, and AudioFilterBiquad. AudioInputI2S and AudioOutputI2s allow the user to sample an input signal or output a signal using the ADC and DAC channels on the Teensy Audio Adaptor Board. AudioInputI2S allows us to connect the guitar input signal to the input of an ADC channel. Next is the AudioFilterBiquad class, which allows the user to implement a biquadratic cascaded filter, useful for all kinds of filtering. In our design, we use a method of the AudioFilterBiquad class called setLowpass, which configures a stage of the filter with a low-pass response, with the specified cutoff frequency and Q shape. Now our design has input through an AudioInputI2S connection and a filter, but they are not connected to each other. This is where the AudioConnection class comes into play. The AudioConnection class routes the audio data from the source object’s output to the destination object’s input. In our case, we use multiple AudioConnection objects to route the sampled data in from the guitar to the filter, and then routed out from the filter to the line out of the audio board.

# Test Results

Overall, programming and assembling of the hardware for the wah pedal was a success, with minor changes to hardware and software. One change that was made was changing the design to take mono audio input instead of stereo. In our initial design, we had the misconception that guitar audio waveforms were stereo, but performing a little research revealed that all guitars output mono audio, except for very rare case. The other main change was where the filtering would occur. In our initial design, we envisioned using the audio codec chip located on the audio board (SGTL5000) to process the audio and perform the required digital filtering. However, upon inspection of the codec chip’s datasheet, it was revealed that the primary use of the chip was its audio-quality ADCs and DACs, but lacked any real ability to perform processing. Therefore, its use in our design was to sample audio in and send it back out after the DSP was performed on the Cortex-M4.

# Lessons Learned

Valuable lessons were learned throughout the course of the project. One of which was that in order to keep project cost down you must research and determine how much processing power you need. While choosing the correct processor for the application, many professional audio DSP chips popped up that could easily perform the functions we required of it. However, these DSP chips cost at least one hundred dollars which was not an attractive price point. The Teensy was the perfect solution for our needs as it provided high quality minimal audio processing at a low cost.

Another lesson that was learned was that finding a used item that has many of the parts you need already assembled together is much better than finding each part individually, buying them new, then assembling them. The broken Cry Baby pedal proved to be of much higher quality than we could have hoped for in a generic pedal enclosure. It also cost much less.

Proper use of the repository became important as the project’s material continued to grow. Code management became much easier and efficiency was increased.

Another lesson that is fabrication related is do not solder a board while the power is on. This mistake was made and contributed to the destruction of the Teensy’s power regulator. If soldering on a powered board is required, ensure the board is first powered off.

# Final Notes

The most common way to program the Teensy board is using an add-on that allows the user to program the Teensy device using the Arduino IDE. The add-on is called “Teensyduino”. The installation file has been included in the project submittal, in the folder named “Source Code\WahPedal”. The latest version and a well written-guide to installing the Teensyduino add-on is located at the link below.

<https://www.pjrc.com/teensy/td_download.html>