**Final Project**

# Teensy based flanger

MUMT 307: Music  Audio Computing II

Winter 2021
Schulich School of Music
McGill University

Last updated : April 26, 2021

Siméon **Le Grand** 260999667

## Contents

## 1 Objectives

The objectives of this project are fairly straightforward. The main objective is to use Teensy microcontroller as the brain for a guitar pedal. This microcontroller can implement any effect that is programmed into it. In this case, the effect that was implemented is a flanger. The flanger effect is programmed in C++. The Teensy microcontroller is programmed using the Arduino IDE. The latency caused by the effect should not be noticeable by the guitarist when using the guitar pedal.

## 2 Hardware

The hardware used for the guitar pedal consists of the Teensy microcontroller, the audio adapter board, an input buffer and an output buffer. The Teensy 4 microcontroller and audio adapter boards were purchased from PJRC, the makers of these boards. The audio adapter board is made exactly for the Teensy 4 board. This allows the user to record and output 16 bit samples instead of the 10 bits of the Teensy 4. It also allows control of the SGTL5000 codec chip. This chip is used as the ADC/DAC. It then sends this data to the Teensy 4 by I2C communications protocol. The software treatment is done in the Teensy. This is where the flanger effect is applied to the sound.

The input and output buffers are simple op amp circuits that are meant to do a few things. First of all, these circuits are there to isolate the Teensy from the rest of the circuits. In this case, a guitar, guitar pedals and a guitar amplifier. These buffers set the impedance [Ω] of this guitar pedal. Input impedance is 1MΩ and output impedance is 1kΩ. They also set amplification levels to match guitar signal level to line level which is what the Teensy audio adapter board is expecting. A volume knob is also implemented in the output buffer.

It should be noted that the input and output buffer circuits were developed with other students as part of a group project for a class at Polytechnique Montréal.

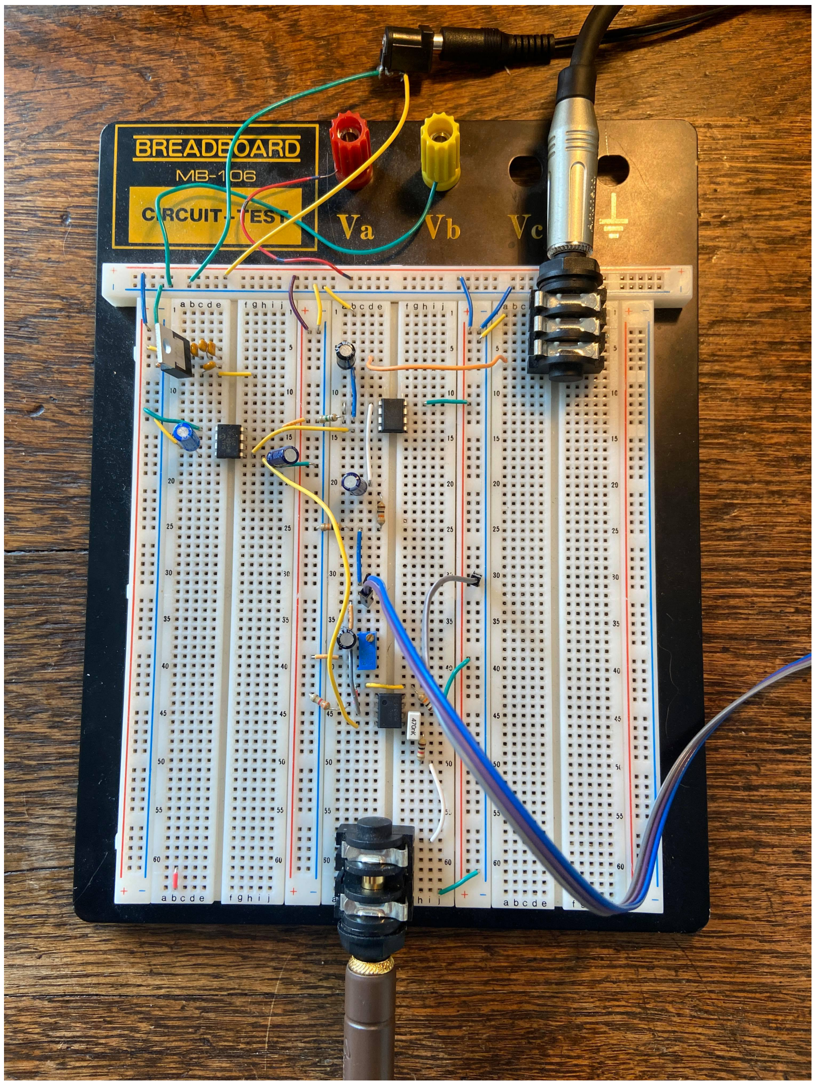
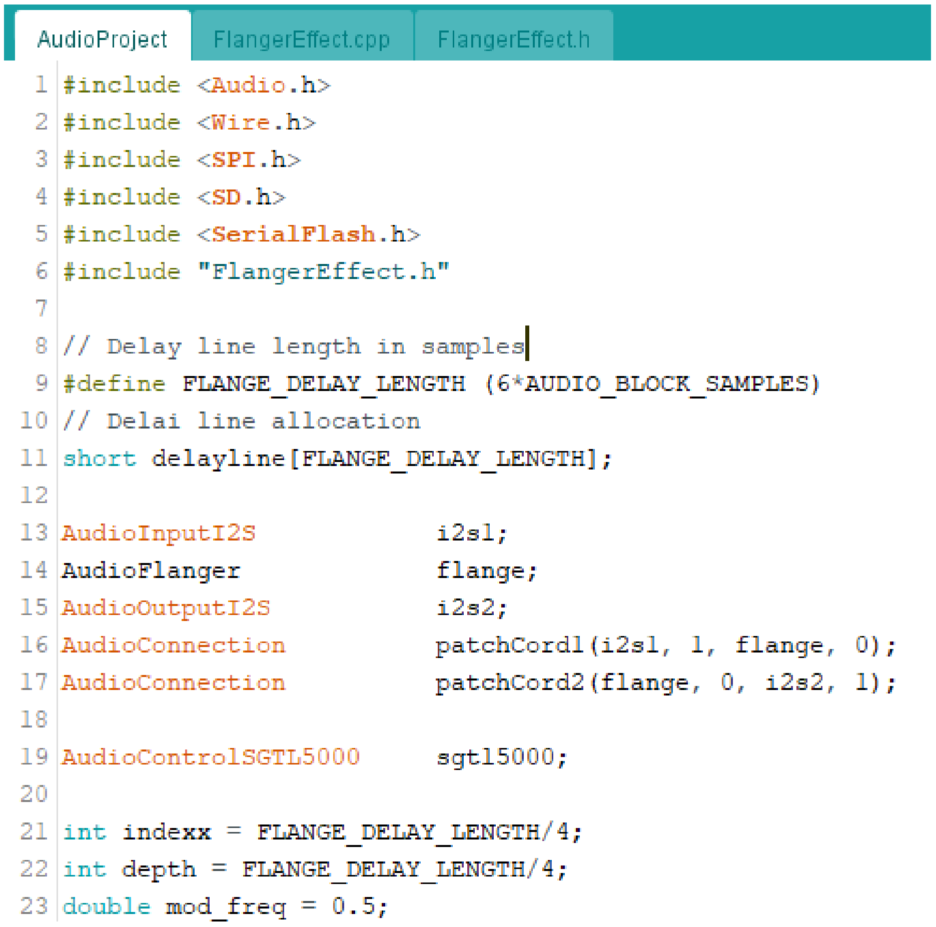


Figure 1: Analog circuit of the guitar pedal

## 3 Arduino IDE

Teensy microcontroller use the open source Arduino IDE to program its boards. This way the boards can be programmed with relative ease by the user without having to worry about the intricacies of programming such a board. Teensy boards have a lot of libraries that make programming and using them much easier. There are the "Audio.h" and "AudioStream.h" libraries that are very useful for audio. They allow the user to create patch chords between different objects much like in MAX/MSP.


AudioProject FlangerEffect.cpp FlangerEffect.h

```
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>
#include "FlangerEffect.h"

// Delay line length in samples
#define FLANGE_DELAY_LENGTH (6*AUDIO_BLOCK_SAMPLES)
// Delai line allocation
short delayline[FLANGE_DELAY_LENGTH];

AudioInputI2S            i2s1;
AudioFlanger             flange;
AudioOutputI2S           i2s2;
AudioConnection          patchCord1(i2s1, 1, flange, 0);
AudioConnection          patchCord2(flange, 0, i2s2, 1);

AudioControlSGTL5000     sgtl5000;

int indexx = FLANGE_DELAY_LENGTH/4;
int depth = FLANGE_DELAY_LENGTH/4;
double mod_freq = 0.5;
```

Figure 2: Arduino IDE : Audio project

As can be seen in the previous picture, an audio input and audio output are defined by using the AudioInputI2S and AudioOutputI2S objects. These are then routed to the AudioFlanger object via the AudioConnection objects. The AudioConnection objects are the equivalent of the patch chords in MAX/MSP.

To start the flange object, a few parameters must first be defined. There is first of all the memory for the delay line. There is also the delay length, index, depth and modulation parameters. What these do will be explained in the section regarding the implementation of the flange effect. To start the effect, the start function must be called in the setup phase of the start up. The previously mentioned parameters are called in the start function. This begins the flange effect. In the setup section, there are also multiple other calls to the SGTL5000 to properly set it up for further use. These are mostly self explanatory by there name.

## 4 Flanger effect header file

The flanger effect header file defines the parameters of the flanger effect the is implemented in the .cpp file of the same name. First of all, the AudioFlanger object inherits the AudioStream library. This allows our object to use the properties of the AudioStream.h library.After that, there is the object constructor that also constructs the AudioStream object with one channel. There then is the initialization function called start. This allocates various parameter values to their respective variables of the flange object.

There then is the update function. This is where the bulk of the computation occurs. This is where the flange effect is actually implemented in the .cpp file. This function is called every 128 samples or approximately every 2.9 milliseconds.

Private variables are defined in the private section of the header file. These variable are mostly self explanatory by their name. inputQueueArray is a pointer to an audio_block_t structure that is defined in C. This is used in the constructor call of AudioStream.

## 5 Flanger effect .cpp file

This file implements the actual code that creates the flanging effect on the audio stream. There is the arm_math.h library that is included. This library is included for the arm_sin_q15 that can be called. This is needed to calculate the sin wave that circulates in the samples to add these to the current sample.

The first function to be implemented is the start function. This function is used in the setup() function that is called at power up of the board. The start function allocates parameters to the private variables of the AudioFlanger object.

The following function to be implemented is the update function. This is where calculations for the flanger effect takes place. First of all, an audio block is allocated in the memory. This block receives the 128 samples of input. Before doing the calculations, we first verify that it is pointing to some data. Once the operations of the flange are complete, the audio block is outputted to the following object on channel 0 via a patch chord. In this case, the following object is the I2S output. The memory for the audio blocks is then released to be reused. This is done via the release() function.

To implement the flanging effect, a sine value is calculated with the input being the modified delay rate. This delay rate is modified to be within 0 to 0.999 range. This sine value is then multiplied with the modulation depth variable only to be added with the delay offset and then subtracted from the circular index value. We then add or subtract the average delay length based based on the negative value or not of the buffer index. Interpolation is then applied to the values of the delay line. This is to make sure not to have any discontinuities in the signal. All these operations are done for every sample in the audio block.

## 6 Issues

The first issue to present itself when doing this this project was the steep learning curve that was using the Teensy microcontroller and its audio libraries. These are documented but it is a lot of information to digest all at once since everything is interconnected. The biggest issue was the creation of a new audio object. This was overcome by consulting the documentation available on the PJRC (Teensy parent company) website.

Another issue that was quite challenging was the use of a circular buffer on the delay line. This was troublesome to implement but was finally resolved by a few easy lines of code that simply verify if index has surpassed the average delay and if so makes the circular buffer equal zero.

## 7 Shortcomings

A shortcoming of this project is the inability to modify most parameters of the flane effect. Only the modulation frequency can modified. The delay length and depth cannot be changed without the effect creating weird unwanted noise.

Another shortcoming was the implementation of a amplifier of the delay line before combining it with the current sample. This is represented a the notch depth of the filters that are created by the flanging. However, this could not be implemented. A simple multiplication should have sufficed but this did not give the expected results.

## 8 Future development

The first development that should be undertaken is the use of potentiometers to modify the parameters of the flanger. This can be done fairly easily by connecting a potentiometer to an analog pin of the Teensy 4. Then, in the loop function of the AudioProject file, a function would verify if the value of the potentiometer has changed. If so, the value would then be scaled and a function would be called to modify the value of the flange object. This would mean a new function that only contains private variables that need updating.

Another further development is to modify the code to allow multiple parameters to be modified. Right now, only the modulation frequency can be modified without causing any problems.

## 9 Sources

PJRC website for audio connections.
`https://www.pjrc.com/teensy/td_libs_AudioConnection.html`

PJRC website to create new audio objects.
`https://www.pjrc.com/teensy/td_libs_AudioConnection.html`