Wavetable Synthesis on Microcontroller Requirements

# Overview

The primary aim of this project is to produce a solution which is capable of running wavetable synthesis on a Teensy 3.2 microcontroller board. This document will define the requirements for the solution, in particular aiming for the following goals:

* Requirements should be defined so they can be easily confirmed and completed
* Individual requirements should be as independent as possible, to facilitate scheduling
* Requirements for individual parts should fit into a process of iteration

# Major Components of the Solution

## Script/program to extract samples from SoundFont file

A script, written in Python, which will take a SoundFont file and produce a C++-formatted byte array of instrument samples for easy use in wavetable synthesis the Teensy.

## Script/modeling of wavetable synth program

A script, written in Python, which models the core functionality and API design of the Teensy wavetable synthesis solution.

## Audio library portion on the Teensy

A wavetable synthesis library written for the Teensy 3.2 which is capable of taking the previously mentioned instrument samples and producing musical output based on interaction with its API.

# SoundFont Sample Extraction Script

The following is a description of the behavior of a utility that will to unpack SoundFont files and generate byte-array encoded samples which will be used by the Teensy wavetable synthesis library.

## Functional Requirements

### User loads SF2

Description: To set the stage for processing SoundFont samples, the user must provide a .sf2 for the utility.

Input: Some arbitrary .sf2 file.

Output: Confirmation to user that the input file is compatible, otherwise a notice that the file has not been accepted.

### List contents of SF2

Description: Display to the user the contents of the loaded SoundFont in some selectable format.

Input: Verified .sf2 file.

Output: List of the instruments within the inputted SoundFont.

### User select instrument from list

Description: User selects which instrument(s) they would like to use.

Input: some user input corresponding to the SoundFont sample they would like to use.

Output: Confirmation to user of a successful selection.

### Parse sample into Attack, Sustain, Release

Description: Prepare the sample in some format that is appropriate for the wavetable.

Input: Successful instrument selection from user.

Output: 1 .cpp file with 3 unsigned int arrays holding contents or dumping pure wav bytes to disk for the 3 logical parts of a sample (Attack, Sustain, Release).

## Non-Functional Requirements

Size of the sample output size will be constrained such that it can be loaded onto the Teensy 3.2. This will be an easily configurable value within the script to accommodate usage on other microcontrollers which may have more onboard memory to utilize.

# Modeling of Wavetable Synthesis Library

This section covers the modeling of the wavetable synthesis library that will be implemented in Python. The purpose of this modeling will be as a test vehicle to:

* Better facilitate validation and verification of synthesis algorithms.
* Provide a test bed that is not significantly constrained by memory or processing resources.

## Functional Requirements

Below are itemized requirements for this portion of the project containing a description, and completion criteria.

### API that mimics that of the Teensy wavetable synthesis library

Description: The core library of the Python module will implement, as faithfully as reasonable, the core API structure of the Teensy library. More specifically, this means that there will be a general parity in class names and associated methods.

Concessions should be made as necessary, but primarily only to meet constraints of the Python programming language relative to C++. This item refers to API-related functional requirements of the Teensy library.

Completion Criteria: For each major class or method in the Teensy library, there will be a parallel class or method in the Python library.

### **Core functionality that mimics Teensy wavetable synthesis library, implemented with both** floating point and integer math

Description: Processing functionality, inferred by the sharing of the API between the Python library and the C++ library, will be implemented in the Python library as in a manner that allows the same input and produces the same output, at least for integer math, as the Teensy library. A floating point version of the synthesis will also be implemented, where the primary difference should be simply floating point arithmetic versus integer arithmetic.

Completion Criteria: Input allowed and output produced by the Teensy library, as detailed by its requirements section, will be available and consistent in format. Validation of this may occur, at least for output, manually by a listener.

### WAV file and real-time audio output

Description: The Python library will be capable of producing audio output that can be sent to either the speakers of the computer it is running on, or dumped locally to a WAV file.

Completion Criteria: User capable of selecting output type, and Python library producing output as per requirement F.2.

## Non-Functional Requirements

R.1: Automated testing facilities

Description: Where appropriate and possible, unit test will be added to validate basic functionality of the methods written in Python.

Completion Criteria: Testing results, including functional coverage of the Python library should be readily runnable and viewed by the developers.

# Wavetable Synthesis Library for Teensy

This section outlines the requirements for the library features that will be created for use with Teensy & TeensyDuino. Where appropriate interface details are given.

Note that in order for this new library to work it utilizes certain features of the pre-existing Teensy Audio Library. Namely, Audiostream.h.

## Functional Requirements

This section describes what the new library will be able and where appropriate describes how the interface for the feature should behave by giving input/output pairs.

### Load audio data from the file produced by SF2 extraction process.

Description: In order to make audio data available for other library features (interpolation, tremolo, vibrato) the .cpp file that contains a const uint array from the SF2 extraction will be loaded and stored for repeated use by other functions within the library.

This data will be stored in a way that makes the attack, sustain and release phases of the audio sample available separately.

Input: A .cpp file that was generated by the SF2 extraction script.

Output: The three phases of the audio sample attack, sustain and release will be loaded into memory.

### Play silence in the event of read error.

Description: If the audio data can’t be loaded from a file or converted properly (interpolation, tremolo, vibrato) then the memory for the the audio samples will be filled with 0’s for silence.

Input: File with invalid data.

Output: Memory for storing attack, sustain and release is filled with 0’s.

### Create interpolated audio data on the Teensy.

Description: Given a sample of audio data it will be converted to audio data for a different pitch using an interpolation algorithm. The exact algorithm to be used is to be determined in design/prototyping, and will largely be determined by the method which produces the highest-quality sound while still performing at a high enough rate to keep up with audio playback.

The interface to accomplish this should take in one audio sample at a time. This allows each phase of an sample to be interpolated separately.

Input: A buffer of audio data that has been loaded from memory.

Output: A buffer containing the interpolated audio data changed to a new pitch.

### An audio sample’s sustain data can be looped.

Description: When playing the attack, sustain and release phases of an audio sample the sustain phase of the sample can be looped repeatedly.

Input: Trigger the library to play a note with a duration longer than that of the raw sustain data.

Output: An unbroken output buffer filled with looping sustain data as long as the predetermined audio sustain length.

### Audio data can be modified with a tremolo effect.

Description: Given a sample of audio data it can be modified to have a tremolo effect by using a tremolo algorithm. This effect can be coupled with any other effect, including interpolation.

Input: A buffer of audio data that has been loaded from memory.

Output: A buffer of audio data that has been modified for tremolo.

### Audio data can be modified with a vibrato effect.

Description: Given a sample of audio data it can be modified to have a vibrato effect by using a vibrato algorithm. This effect can be coupled with any other effect including interpolation.

Input: A buffer of audio data that has been loaded from memory.

Output: A buffer of audio data that has been modified for vibrato.

## Non-Functional Requirements

### **Entirety of code and sample data must fit within the Teensy’s limited 64k on-board** memory.

Description: The Teensy 3.2 DAC has 64k of on-board memory. The audio library that is produced, as well as the base instrument sample arrays must be as memory efficient as possible to conserve memory. The produced functionality should be able to run fully-functional with this amount of memory.

### Audio interpolation must be fast, and without notable clipping.

Description: The purpose of this wavetable synthesis project is to allow users to create usable musical instruments, so the audio that is produced from the library must be free from noticeable audio clipping, and should perform in real-time as to prevent any noticeable input lag for the user. Instruments need to be responsive for their players, so this library should be held to the same standard.

### Library must be well-documented and must expose all functions necessary to utilize all features of the library.

Description: This library will mainly be used by developers who will look to use it in their own projects. To make their lives easier, the library we produce must be well documented, and has to expose all functions necessary to take full advantages of all of the libraries features.

# Minimum Viable Product

The minimum viable product will tentatively be set as a combination of the above three components, with the following limitations:

* An attack, sustain, and release sample for a single instrument will be extracted from a SoundFont file and stored in a .cpp-based byte array
* The byte array will be loadable by both the modeling program, and the Teensy based wavetable synth library
* Both the modeling program and the library will play a single scale, using the extracted samples

This will give us a good baseline from which to add in additional functionality as per the requirements listed above in this document.