Software Requirements Specification

**Project: TeensyAudio Wavetable Synthesis**

Authors: Ryan Mellmer

Connor Delaplane

Aida Keifer

Nicholas Craig

Josh Bucklin

Jonathan Jensen

Xuan Tang

Date: 2016-11-28

Version: 1.15

**Table of Contents**

[1.0 Introduction](#_vm2kuca7lyhx)

[1.1 Overview](#_92ebyw6sfv5)

[1.2 Glossary](#_ggngisx56073)

[2.0 Major Components of the Solution](#_4mx9ak7o8wrv)

[3.0 SoundFont Sample Extraction Script](#_tinnt1cuepby)

[3.1 Functional Requirements](#_x28yywhu4xde)

[3.2 Non-Functional Requirements](#_s5yvs3gz8wi8)

[4.0 Modeling of Wavetable Synthesis Library](#_89xo84svvvkt)

[4.1 Functional Requirements](#_tlx5m1fvwi53)

[4.2 Non-Functional Requirements](#_f9rtaos8p5)

[5.0 Wavetable Synthesis Library for Teensy](#_dq4hmprwvxx)

[5.1 Functional Requirements](#_jzrcnzcjuahn)

[5.2 Non-Functional Requirements](#_e2n7crcjghfp)

[6.0 Minimum Viable Product](#_gsu2ye6qk26s)

# 

# 

# 1.0 Introduction

### 1.1 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

### 1.2 Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Wavetable Synthesis | A form of musical synthesis relying on the modification of actual instrument audio samples. This is contrasted with other forms of audio synthesis that work on fundamental waveforms. |
| SoundFont (SF2) | A file format capable of storing instrument samples for use in wavetable synthesis. |
| Teensy | USB-based microcontroller development system distributed by PJRC. (website: pjrc.com) |
| Attack | Portion of the decoded sample array which plays immediately upon activation (e.g. piano key depression). |
| Sustain | Portion of the decoded sample array which can be looped indefinitely to produce a sustaining, consistent instrument tone. |
| Release | Portion of the decoded sample array which plays after the sustained section of a note, and will mark the natural end of a tone. |

# 2.0 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.

## 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.

# 3.0 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.

## 3.1 Functional Requirements

### User loads valid 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.

### User loads invalid SF2

Description: There are various different criteria under which an .sf2 will be rejected by the script:

* .sf2 depends on specific oscillators/effects in order to function correctly
* An individual note recording is too large to fit within the Teensy’s memory

Input: Invalid .sf2 (doesn’t conform to the above criteria)

Output: Error message explaining why the file won’t be processed.

### 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).

## 3.2 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.

# 4.0 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.

On a high level, this library will take extracted samples encoded in a byte array as input and, through programmatic interaction with its API, produce output compatible with the Audiostream.h API of the Teensy Audio library.

## 4.1 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.

## 4.2 Non-Functional Requirements

### Implementation in C++ as a class or set of classes

### Description: Implementation of the Wavetable Synthesis on the Teensy library will occur entirely in C++. Primarily this will facilitate interoperability with the Audiostream.h API.

### **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.

# 5.0 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.