Design Document

Embedded Systems Digital Audio Loop Station

Max Diebold, Brahm Rifino
CS 461-401 Fall 2022
Oregon State University

Abstract

Audio sampling and its techniques are constantly evolving with new problems waiting to be solved. There is a need for a quick and cost effective way to sample audio both live and recorded. An embedded system using gloablly available hardware and software will be proposed that will provide a group or individual with a portable audio sampling and playback device. There is often a delay between the recording/sampling of audio and its playback. This system will allow users to immediately playback live recordings while also offering playback of saved audio files. Various sound effects and alterations tools will be given to the user to allow editing and modification of audio playback.

Table of Contents

Revisions

Iteration	Version	Date	Description	Notes
1	v1.0	11-3-22	Initial Design	
2	v2.0	11-21-22	Final Design	Updated appendices Pitch shift effect

Overview

In this document our group will be covering and informing of our initial design plans for the Embedded Audio Loop Station that we will be developing. Within this document we have considered and will be including information from past reports. These past reports have served as milestones during our initial brainstorming to where we are now. Our requirements document is serving as the basis of this initial design phase.

The purpose of this document is to address the primary goals, objectives, and functional areas of the audio loop station that will be developed. With that said, we will be focusing on the minimum viable product (MVP) during this initial design phase. The MVP serves as the backbone of the audio loop station and its design requirement. In future interactions of this document we may include changes and improvements to our audio loop device, so we will reflect detailed design changes, in detail, as needed. The design team and the project manager are the intended audiences of this design document.

Definitions

The following terms are defined:

- **Audio Loop Station -** A device that plays audio using a live recording or previoiusly stored audio files.
- **Data Storage -** Digitally recorded files that are saved in a storage system for future use.
- **Directory -** A directory is a unique type of file that contains only the information needed to access files or other directories.
- **Embedded Software -** Software that has fixed hardware requirements and capabilities. Created exclusively for the particular device that it runs on.
- **Formant -** A local peak in the spectrum of a sound that contributes to the sound's timbre, or character.
- **Monophonic Audio -** Audio that has been recorded from a single signal or channel such as a microphone or a single audio device/instrument.
- **Playback -** The act of reproducing a sound or video recording, especially in order to check a recording that is newly made.
- **Potentiometer -** Physically, this component serves as an adjustable voltage divider. In the context of this project, it would enable fine-grained control of a target parameter.
- **Quantization -** In the context of signal processing, the mapping of a continuous symbol to a discrete, finite set; often used to truncate a signal to a specific boundary.
- **Recording -** The action or process of recording sound or a performance for subsequent reproduction or broadcast**.**
- **Stereophonic Audio -** Audio that has been recorded using two audio signals or channels, producing a fuller and more live sound.
- **Tempo -** The speed at which a passage of music is played.

Introduction

The field of audio processing and sampling is continually evolving and growing. There is a consistent need for new and affordable technology. Current audio sampling and playback devices can be expensive and tedious to learn how to use. Often, these devices, such as the Akai MPC (MIDI/Music Production Center) [1], require supplemental knowledge on how to use and operate their software. Many musicians have made devices like the MPC popular with average and professional people alike wanting to learn and produce music using these devices. These sampling devices prices can often go into the thousands of dollars, this was especially true when they were first being introduced.

The E-Mu SP1200 [2] was a famous product that was produced by E-mu Systems. The SP1200 was revolutionary, it was portable and able to produce songs by itself, effectively creating a new way to produce audio tracks by then current and future generations of musicians. The SP1200 originally retailed for over $15,000 USD and offered a sampling rate of 26.04 kHz. A sampling rate of 26.04 kHz in today's world is considered low quality. The standard audio sampling rate today is 44.1 kHz [3], the quality that you would expect to hear from a CD or MP3 file. This paper will discuss an affordable and portable high quality audio sampler and looper. This device will use an embedded processor to record, select, and playback audio clips in real time.

A loop pedal, by definition, must be capable of looping recorded audio, though devices vary in their capabilities regarding the number of concurrent tracks. Devices are also varied with respect to auxiliary features: those that fall outside the definition of a loop pedal. The team creating this device has identified two main functional areas: one is concerned with the central capabilities of recording, storage, and playback; the other encompasses the auxiliary features, each of which will communicate with the first area but which will be more self-contained.

The end result of this project will be a physical product, so the user interface will be largely analogue; the general UI for the device will be outlined, and the specific controls that will be used to control each effect will be included in each module’s description. Out of the modules included in the research presented, at least three will be implemented in the final product.

Proposed Solution

System Architecture

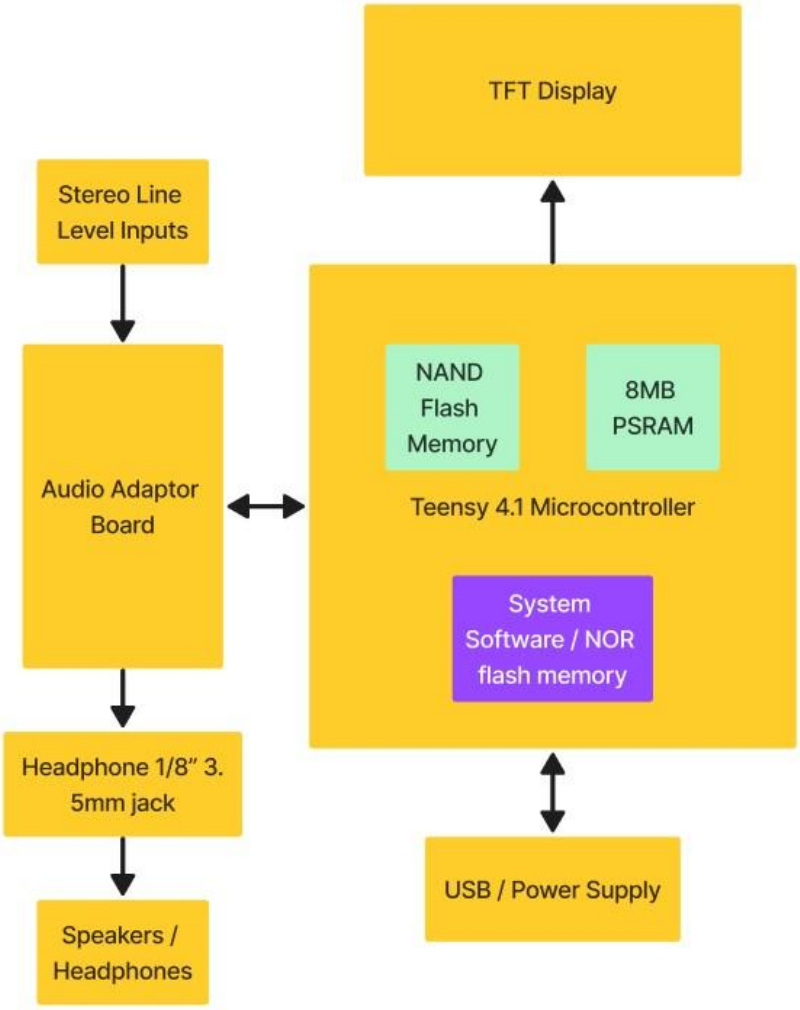


Fig.1. *Hardware diagram for the loop device*

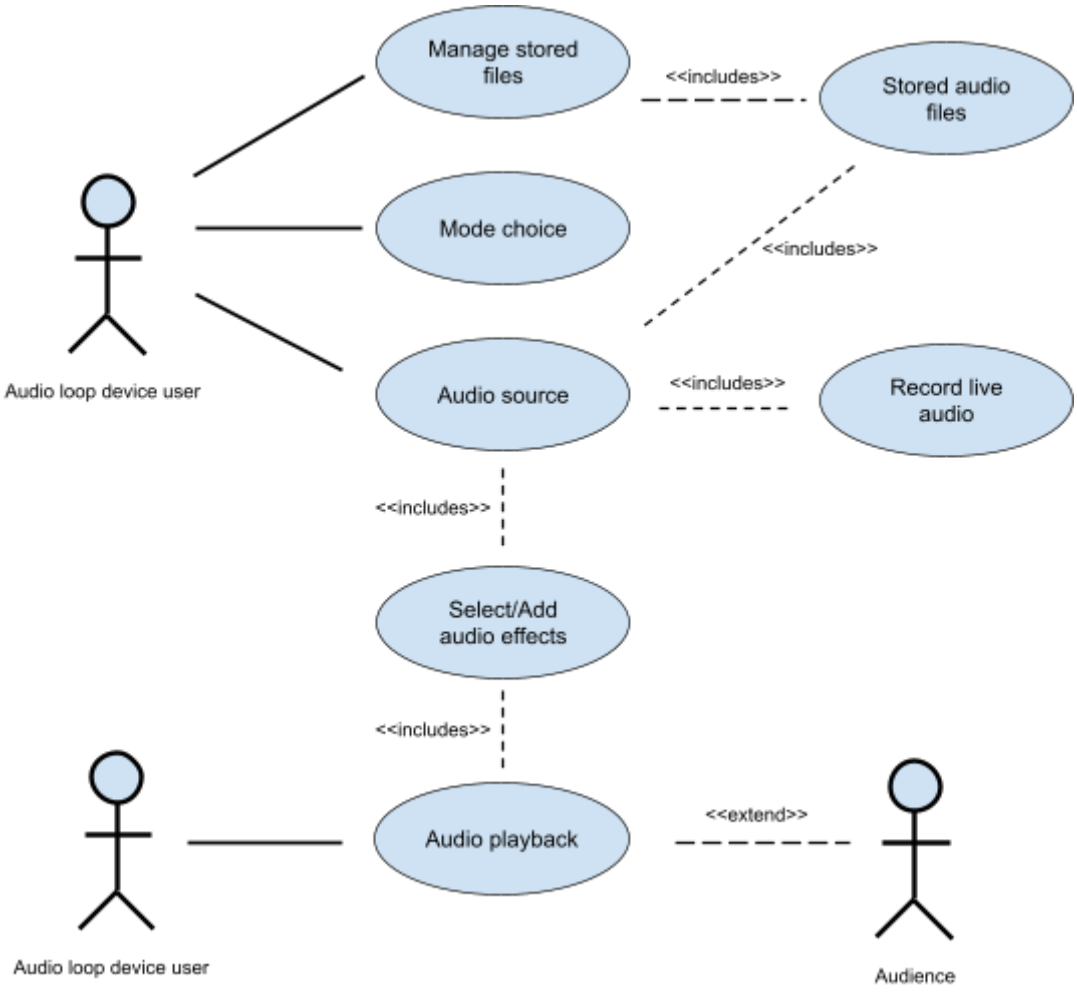


Fig. 2. *Use-case diagram for known users*

System Main Components

Playback

Audio playback is at the forefront of functionality implemented in the design of the device. The device will produce audio output through the playback of live recording and stored audio files. The playback function will be able to immediately reproduce a live recording, or a previously recorded audio recording that has been stored in the device's internal memory.

While live tracks are being recorded for playback, the user will also have the option of loading stored audio for playback. Additionally, effects can be applied to any of the loaded tracks. A key feature of this device is that once audio recording is finished, playback will immediately replay the recording, along with the other tracks that contribute to the loop. Audio

playback will be reproduced at 16 bit or 44.1khz sample rate (44,100 samples/second) which equates to the quality you get from a CD or MP3 audio track. The device will be capable of playback up to 4 simultaneous mono or 2 stereo audio files, at a minimum.

Recording

Simultaneous recording and playback will be possible on the loop pedal. There will be various modes incorporated into the device for its functionality. The user will be able to record live audio through one onboard electret microphone, or a line input may be used for guitars, other microphones, etc. After recording of the audio is completed the device will immediately playback the audio to the user via headphones or speakers. The user will also be able to store the recording in the onboard memory. File storage of recordings and audio files will allow the user to access and playback files of their choice.

Software and Storage

Our audio loop device will run on an embedded system that operates system software, which will process audio inputs and outputs. The user will interact with this software, which communicates with the device's hardware. Storage will be accomplished via onboard memory that will be accessible from the device’s embedded software. The files will be stored in a main directory or in separate directories that we specify. From these directories the user will be able to select audio files for playback.

Reverse Play

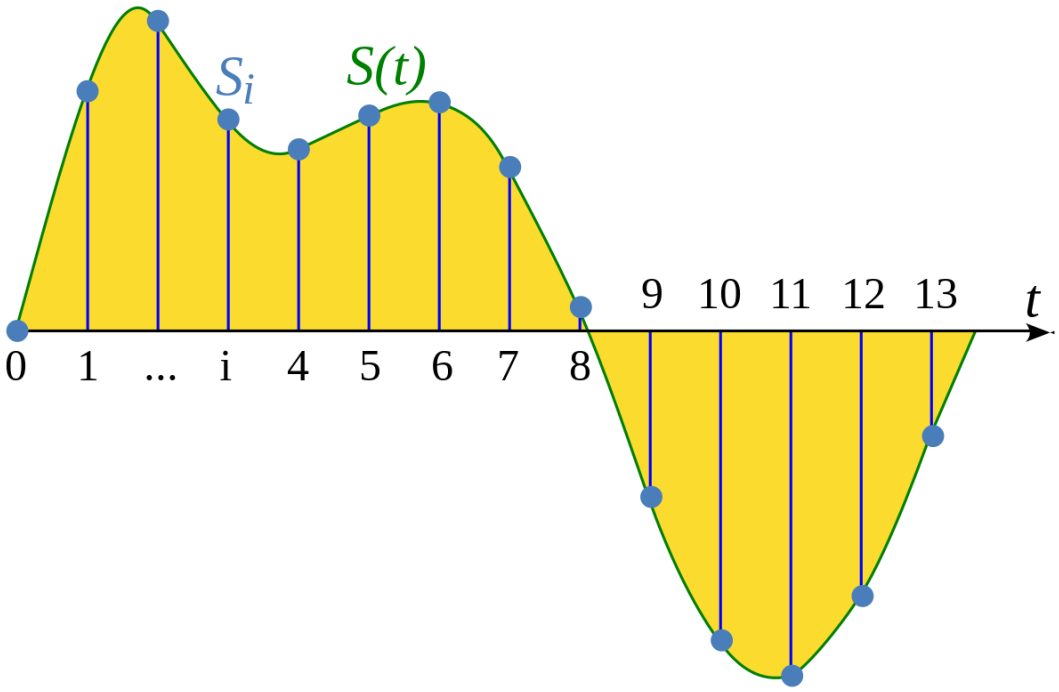


Fig. 3. *Sampled sound wave*

Reversing a recorded audio signal is a relatively straightforward task. When an analogue signal is converted into a digital signal, a discrete number of samples are taken, depending on the sampling rate specified or available. These samples can then be stored as a stream of bytes for further manipulation. Because this effect requires an awareness of all of a track's samples, it is not possible to apply the effect in real time.

The algorithm for this function will proceed as follows:

- Copy the sampled signal into an "undo" buffer file in flash memory, where it will remain until overwritten.
- Treating the signal as a stream of 4-byte samples, place the file cursor at the end of the file stream, then read the stream in reverse into a new file in flash memory. On the next loop, this new file will be read, and the previous file pointer will be used for writing.

Overdub and Replace Modes

The default behavior when recording a new track is to clear the memory associated with the track completely before writing to it. If overdub mode is activated, however, incoming samples will be combined with existing samples, then written to the memory locations previously held by the existing samples. In replace mode, as the name implies, incoming samples will

replace the values previously stored in the memory locations. Either mode, but not both, can be activated on an empty or filled track, but the behavior will be identical to the default for the track's first recording. Overdub mode incorporates the new samples into the existing samples via additive synthesis, which itself is based on the Fourier Transform: any sound signal can be recreated through the addition of other signals, in particular sine waves. This transform is computationally expensive, so the computation and track rewrite will not occur in real time. Instead, the new signal will be passed through alongside the original signal during the initial recording; the next loop will contain the mixed signal.

Loop Trimming

When using a loop pedal, the track first recorded often serves as a "base" for the rest of the tracks, but this isn't always the case. For example, each track could be set to loop over itself, without staying within the bounds of other tracks: this will be the behavior if loop trimming is not enabled on a track. If the effect is enabled, the behavior will also depend on whether overdub or replace effects are activated: if so, the track will loop to its beginning, in time with the basetrack, and will mix with or replace, respectively, the samples stored there. It will be possible to change the track referenced as the base track.

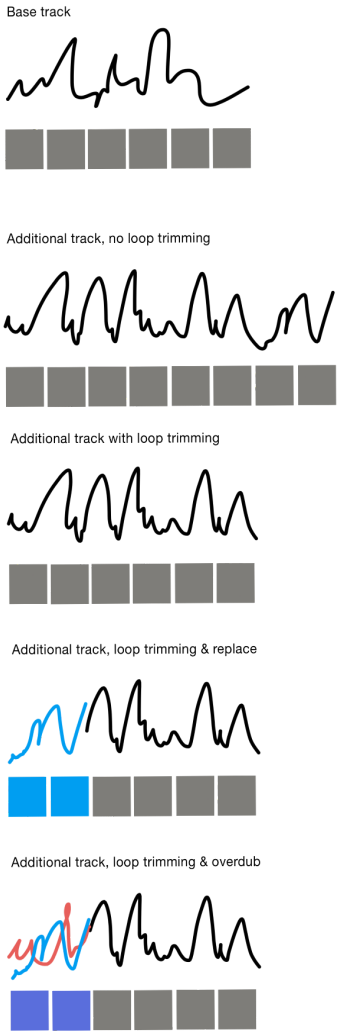


Fig. 4. *Combinations of loop trimming, overdub, & replace*

Slip

With the slip effect, a user can offset the start of a track in relation to another. The behavior that follows, like other effects, depends on whether loop trimming is active: if so, any part of the track that extends beyond the length of the base track will be dropped, or simply not played back; otherwise, the full signal will be played back. Optionally, a “wrap” option can be implemented, which will fill emptied space at the front with samples that occurred beyond the duration of the base loop, if loop trimming is active. It is not possible to apply this effect in real time.

This effect would best be implemented by manipulating the cursor position of the file's read stream, alongside the use of a "delay" variable. This delay will dictate how much empty space to add to the front of a track if there is no wrapping, or it will dictate how far back from the end of the track to place the cursor if there is wrapping. It will also affect subsequent recordings in overdub and replace modes. In the case of overdub mode, the previous, delayed track will be mixed with the incoming signal into a new file in flash memory, while replace mode will only perform exclusive writes.

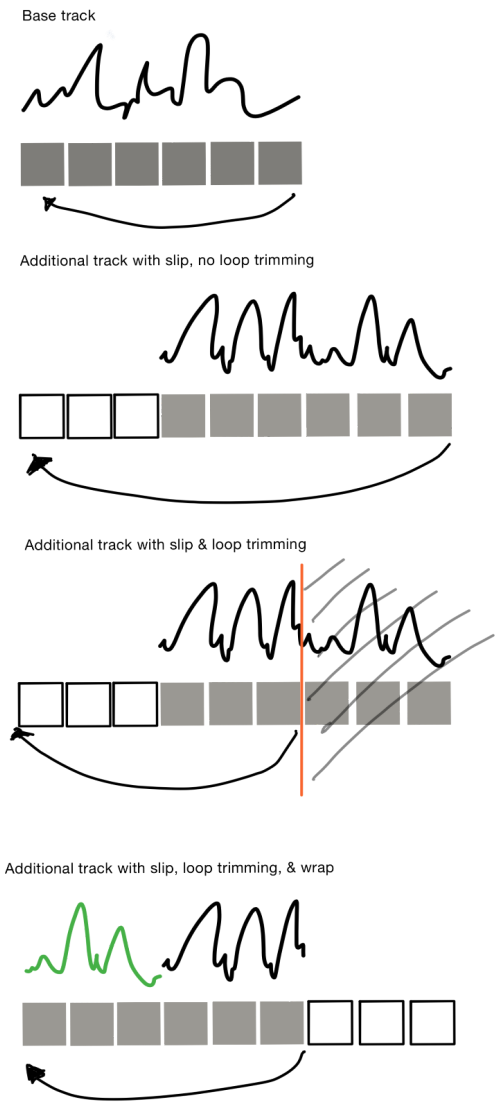


Fig. 5. *Combinations of slip, loop trimming, & wrap*

Pan

Panning an audio signal involves adjusting the relative strengths of the signal sent to the left and right audio channels. This implies a stereophonic signal, which can be accomplished by a) recording a track with two microphones, or b) duplicating a monophonic signal. Our loop device will feature an onboard electret microphone as well as a line input. While it is possible to plug a microphone into the line input, an adaptor and a preamp will likely be needed, and the sound quality will differ from the electret microphone to such a degree that the setup would be ill-suited to stereo recording.

Duplicating a monophonic track will be the more common scenario. The strength of each channel is controlled individually, but the standard approach is to control the total gain of the stereo signal with one control, usually a knob, then to control the relative strength of each channel with another bipolar knob. The device may only have a single knob, in which case a button will be used to select the knob's target parameter. This effect will be available in real time.

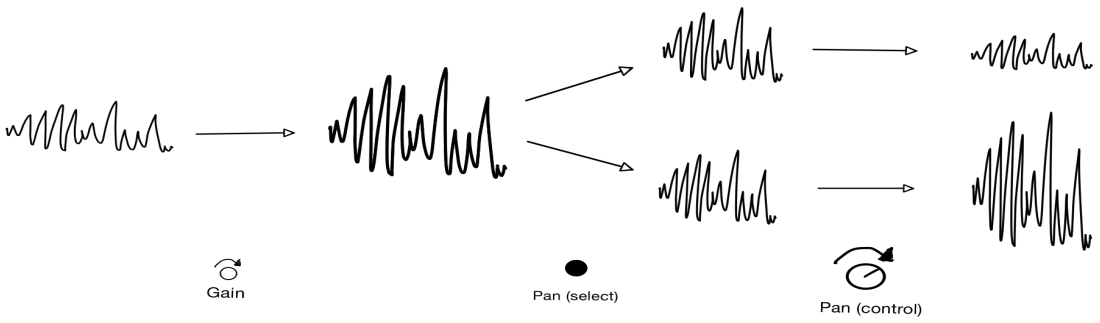


Fig. 6. *Behavior of the pan effect*

Pitch Shift

A variety of techniques exist to alter a sound's pitch without changing its tempo, and they fall into two main categories: time domain methods and frequency domain methods. The former tend to be easier to understand and implement, they take less processing power, and they

naturally preserve the sound's formants. However, they deliver poor results when applied to polyphonic sounds, i.e. sounds consisting of multiple fundamental frequencies [4]. Frequency domain methods require frequency analysis that is relatively expensive, computationally, so testing will need to be done to determine if the effect can be applied in real-time. Although the task used to require specialized DSP hardware, it is now feasible on most average systems, but it still might be beyond the capabilities of the Teensy microcontroller. If it cannot be accomplished in real time, the effect will instead be applied to the next traversal of the loop.

Out of the frequency domain methods, the most reliable technique discovered in available literature is the "phase vocoder" technique. Figure 6 shows a high level diagram of the stages involved in this technique.

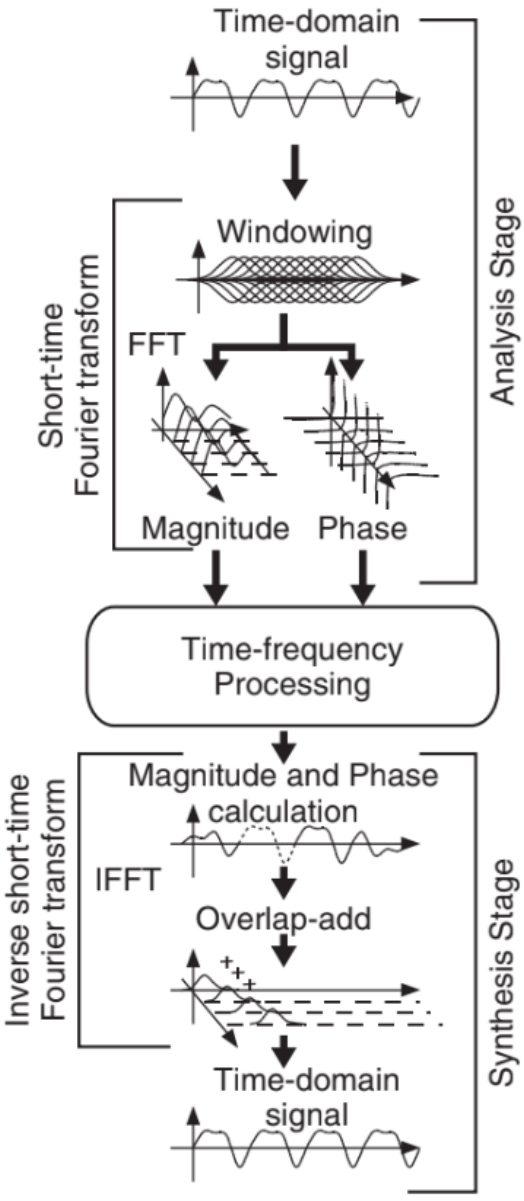


Fig. 7. *Phase vocoder pitch shifting*
Source: [5]

A given segment of a signal is transformed by first analyzing it with overlapping “windows”. An implementation of the fast Fourier transform is then applied to each window to decompose the section into its partial sinusoidal waves, which are each described by magnitude and phase. Finally, the actual pitch shifting is accomplished by scaling the individual waves and performing the process in reverse [6].

The phase of each sinusoid is important because of the discrete number of windows involved in the analysis and resynthesis: in a complex signal, waves will not fall neatly into frequency bins, so the differentiation of the phase is used to calculate the deviation from the true bin frequency. The windows overlap to accommodate a wide range of phase offsets.

This technique will exhibit a degree of latency with most systems, so it will not be possible to apply the effect at the time of recording. However, it may still be possible to apply it in “real-time” in the sense that a signal can be shifted mid-loop: in this case, a short (at least 150 ms) buffer time would be required before the shifted signal would start to sound.

Technologies

Embedded devices do not run in a hosted environment, so there is no code pre-loaded onto the device. This means that there are no interfaces available for interaction with systems, as there are no systems to interact with. Correspondingly, the standard libraries available in most programming languages will not be present, and most modules in the libraries will not work properly if they are included, due to the lack of a runtime [7]. For this project, the members of the team considered two languages: Rust and C/C++. Rust offers a high degree of memory safety, but many of the abstractions that provide this safety reside in its standard library. There is a “core” library, a subset of the standard library, that can be used in programs for embedded devices, but it is relatively young and slim, so much would still need to be implemented from scratch. On the other hand, most of the libraries built for the Arduino microprocessor are a subset of C++, and can be used on the Teensy. PJRC, the creators of the Teensy, have also released a number of libraries of their own, dubbed “Teensyduino”, specifically for the device [8]. The Arduino and Teensyduino libraries are both accessible from the Arduino IDE, which also includes a loader for flashing the program onto the microcontroller.

For this project, we aim to limit the number of libraries we import. However, we do intend to make use of utilities that aid the program in accessing peripherals and in manipulating basic data structures. For the former, the Teensyduino library will provide the best fit [9], while the base Arduino functions will fulfill much of the latter [10].

Logic and Algorithms

- **System Software -** The main function of the system will directly drive the actions of device peripherals to effect specific functionality.
    - **Menu Algorithm -** This function will serve as the middle ground between human interaction and device functionality

        - Choose files - lists available files
        - Record - calls the record algorithm
        - Playback
            - Add effects to playback
        - Backtrack option
        - Choose / Enter option
    - **Playback Algorithm -** This algorithm will playback live recordings or selected audio files from memory storage.
        - **User input -** The user will select an audio file from memory storage for playback.
        - **Get file name -** Obtain the path or file name of the file to be played.
        - **Open file -** Check if file exists, if so then open file.
        - **Read file -** Read file in chunks of bits. Size of chunks is determined by hardware limits.
        - **Write to buffer -** Writes bits to a file in flash memory or a separate working file in SD memory.
        - **Close file -** Close the original file in SD memory.
        - **Read from buffer -** Read from the working file and process bits “just in time” for audio playback.
        - **Close file -** Close the working file when the session is finished, and free allocated buffer memory.
    - **Recording Algorithm -** This function will record live audio from an input and store the file in onboard memory. While a loop as a whole is being edited, flash memory will be used. When the loop is saved, the tracks can be mixed down into a file that will be saved to SD storage. When each recording is finished, the recorded track plays back in real time.
        - **User input -** The user will start recording
        - **Create new file -** Create a new audio file that will hold the bits of the recording. The file will be initially created in flash memory.
        - **Process audio input to buffer -** Writes chunks of bits to buffer. Size of chunks is determined by hardware limits.
        - **Write to file -** Write buffer data to the temporary file.
        - **Close file -** When the loop session is complete, mix the temporary track file with other track files into a single file in SD memory, and free allocated buffer memory. The file is now saved to a path in onboard memory that may later be retrieved.
        - **Finish recording -** Once recording is finished, the loop can be played back or incorporated into later loops.

System Dependencies

- **System Software and libraries**

    - **Main system software**
        - Arduino IDE
        - C++
    - **Software libraries -**
        - Arduino
        - Teensyduino
- **System Hardware**
    - Teensy 4.1 microcontroller
    - ILI9341 320x240 Color TFT
    - Teensy Audio adaptor board
    - 8 Megabyte PSRAM chip
    - 128 or 256 Megabyte NAND Flash memory
    - Solderless Breadboard
    - SD Memory Card

Fig. 8. *Teensy 4.1 board*

Fig. 9. *ILI9341 320x240 Color TFT*

Fig. 10. *Teensy 4.1 audio adaptor board*

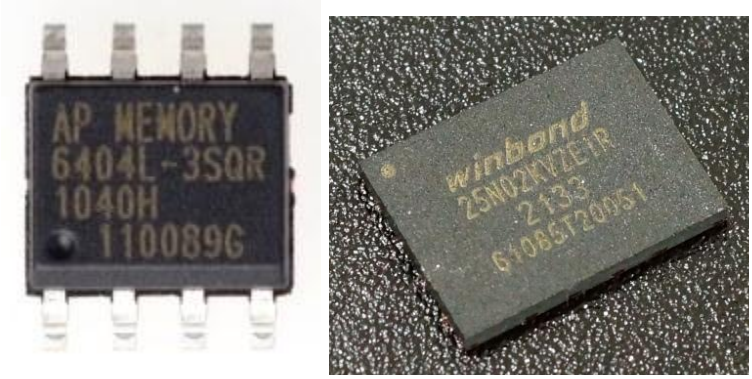


Fig. 11. *8 megabyte PSRAM chip (left). 256 Megabyte NAND Flash memory (right).*

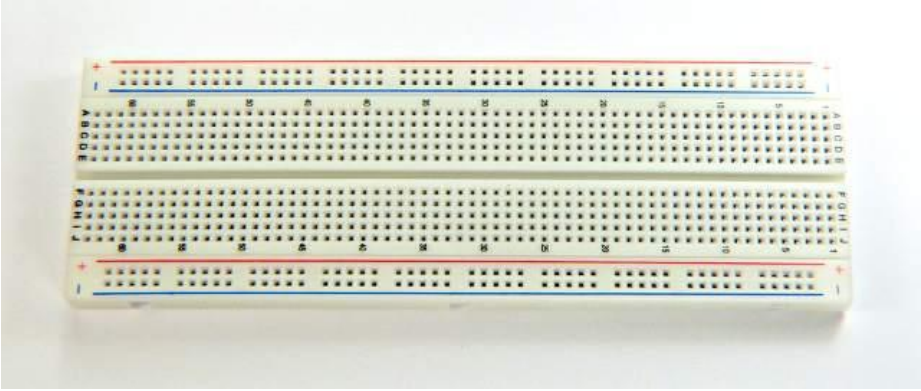


Fig. 12. *Solderless breadboard*

Release / Deployment

As this is not a web application or a software that will be available on multiple platforms, it will not be deployed widely. The team will use GitHub for software version control and releases, but deployment will only be possible on the Teensy 4.1 hardware.

To load the software onto the device, there are two documented approaches, according to the Teensy vendors: the recommended approach is to use the Arduino IDE's loader which, as of version 2.0, is capable of connecting to Teensy boards; otherwise, a CLI tool is available, *teensy_loader_cli*, which is assisted by a configuration file. Both achieve the same result. Because the deployment is so system- and user-specific, both methods will be documented for product users.

Test Plan

This project is heavily dependent on the particular peripherals and capabilities of the hardware, so each update will require a new build and deployment to the device. However, debugging information will be available through the IDE through the serial connection to the device, so the standard testing techniques will still be available. Additionally, a GitHub workflow has been created to confirm that code pushed to the repository compiles successfully for the device. Due to the low-level environment of the project, unit testing should be the first priority, then white box testing may be used on an emulator before black box testing is carried out.

Risk Assessment

- **User Error - Risk Level: Low.** An unknown portion of this project is how well a new user will be able to operate and explore the functionality of the audio loop station.
    - Mitigation: User error would likely be due to incorrect navigation through the devices menus, incorrect audio in/out (not plugged in, reversed, etc.), power cable unplugged, and other related issues. These errors are not a major concern and can easily be remedied through a "walkthrough" and hands on lesson of how to operate the audio loop device.

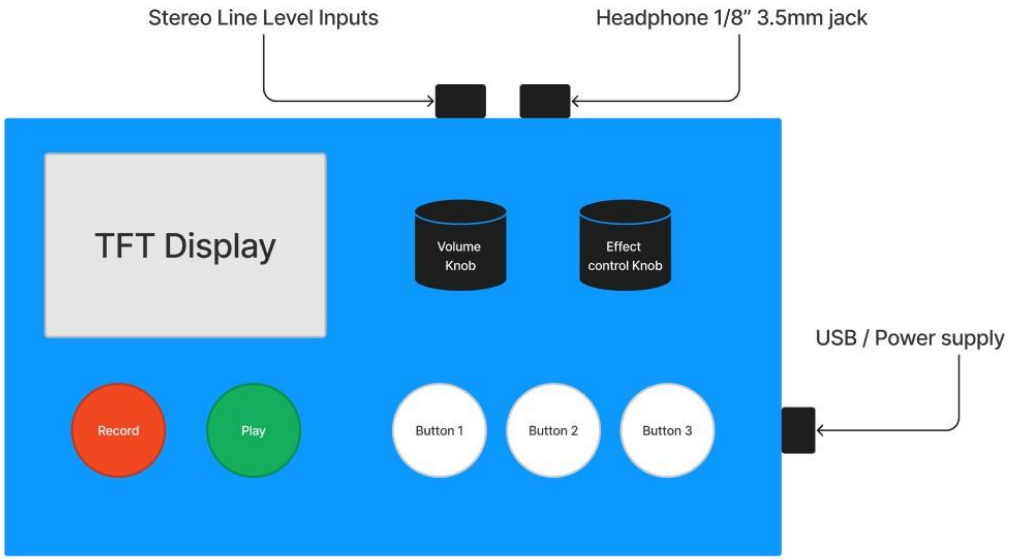
Appendices

Appendix I - Project Phases

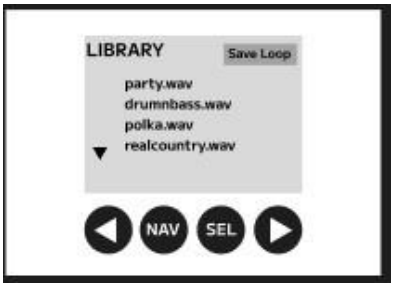
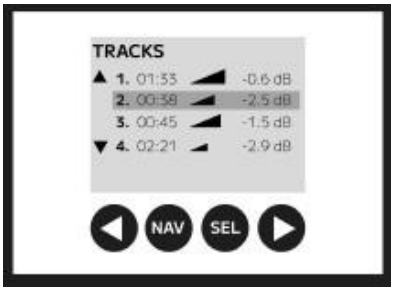
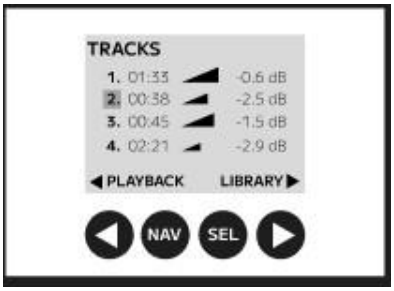
- **Phase 1 - Fall Term:**
    - Discuss team standards
    - Research project and problem at hand
    - Define the problem - Problem Statement (individual)
    - Gather and research project requirements
    - Discuss software, hardware, and tools for project design
    - Created requirements document
    - Identified functional requirements
    - Preliminary Design of project (individual)
    - Discussion of preliminary design
    - Created Design document
    - Final Design Document
    - Client Verification

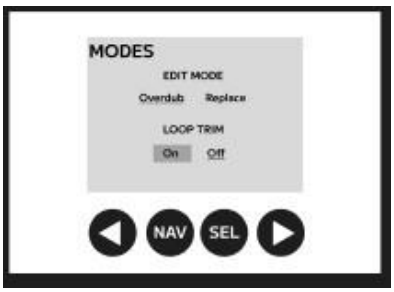
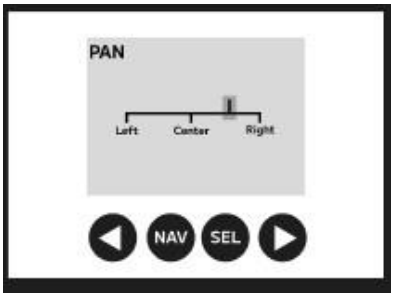
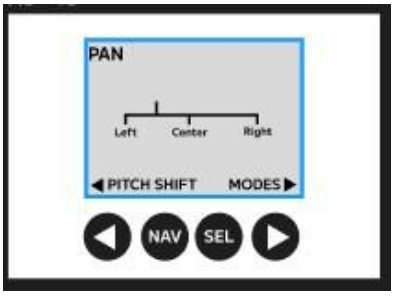
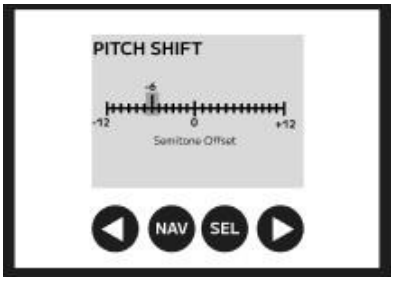
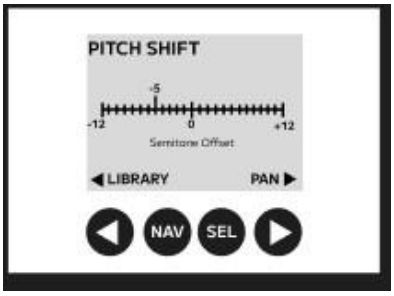
Appendix II - UI/UX

- **Prototype I -**
    - Users will be able to interact with the device through the display screen. There will be several buttons assigned to record, play, and more (button 1, 2, & 3). A volume knob (potentiometer) will be used for volume control as well as an effect knob for strength of effect. Buttons 1 - 3 may be used for added device functionality.



- **TFT Display Screens -** The below table shows the available screens and their descriptions

Screen	View 1	View 2
Library This screen shows the available files that can be used for playback, recording, and effects.		
Tracks Shows the tracks that can be used for simultaneous playback		

Modes This screen allows the user to choose options for the active track in playback		
Playback Shows the user what modes are enabled and the active track that is being used for overdub/replace		
Pan Allows the user to change the speaker balance from left to right		
Pitch Shift Allows the user to change the pitch of the active track		
Reverse Shows the user what track is being reversed, the name of the reversed track, and the playback of the the reversed track		

References

[1] Caleb J. MurphyCaleb J. Murphy is a singer-songwriter and music producer based in Austin, “Mono vs. stereo: Mixing tips for pro results,” *Musician on a Mission*, 23-Apr-2021. [Online]. Available: https://www.musicianonamission.com/mono-vs-stereo/. [Accessed: 26-Oct-2022].

[2] “What is data storage?,” *CDW*. [Online]. Available: https://www.cdw.com/content/cdw/en/articles/datacenter/what-is-data-storage.html. [Accessed: 27-Oct-2022].

[3] “Sample rates and audio sampling: A guide for beginners | adobe.” [Online]. Available: https://www.adobe.com/uk/creativecloud/video/discover/audio-sampling.html. [Accessed: 14-Oct-2022]

[4] T. Royer, “Pitch-shifting algorithm design and applications in music,” 2019. http://kth.diva-portal.org/smash/get/diva2:1381398/FULLTEXT01.pdf

[5] U. Zolzer, *DAFX : digital audio effects*. Chichester, West Sussex, England: Wiley, 2011.

[6] S. Bernsee, “Pitch Shifting Using The Fourier Transform | Stephan Bernsee’s Blog,” *Zynaptiq.com*, May 30, 2008. http://blogs.zynaptiq.com/bernsee/pitch-shifting-using-the-ft/ (accessed Nov. 21, 2022).

[7] “no_std - The Embedded Rust Book,” *Rust-embedded.org*, 2022. https://docs.rust-embedded.org/book/intro/no-std.html (accessed Oct. 28, 2022).

[8] “Teensyduino: Using Arduino Libraries with Teensy USB development board,” *Pjrc.com*, 2022. https://www.pjrc.com/teensy/td_libs.html (accessed Oct. 28, 2022).

[9] PaulStoffregen, “PaulStoffregen/Audio: Teensy Audio Library,” *GitHub*, Sep. 16, 2022. https://github.com/PaulStoffregen/Audio (accessed Oct. 28, 2022).

[10] “Arduino Reference - Arduino Reference,” *Arduino.cc*, 2022. https://www.arduino.cc/reference/en/ (accessed Oct. 28, 2022).