# Simulacra for Obsolete Audio Effects

A reconfigurable open-platform effect pedal based on the Bela framework.

Lehel Török
52103627

**Master's Thesis**

in

Sound Design

Supervisor: Ao.Univ.Prof. Dipl.-Ing. Winfried Ritsch

FH JOANNEUM
Institut für Elektronische Musik und Akustik
Universität für Musik und darstellende Kunst Graz

Graz, 28. August 2023

## Abstract

The increasing accessibility and power of embedded single-board computers and prototyping platforms have resulted in the rise of various frameworks specifically designed for creating digital musical instruments and audio effects. These advancements have opened up possibilities for utilizing embedded computer technologies to replicate rare or outdated instruments and audio effects. This thesis centers around developing an open-platform guitar effects pedal using Bela's embedded computing framework, allowing guitarists to create and apply distinct audio effects on dedicated audio hardware. An extension cape's construction, design, and development process are documented, combining essential components like rotary encoders, switches, and audio inputs/outputs into a single circuit. A comprehensive software framework is established using Pure Data, providing all the necessary functionalities to utilize the device in a multi-effect configuration and incorporating a signal chain of outdated audio effects designed around a physical model of Hammond's Scanner Vibrato unit. Finally, benchmarking and discussions are carried out to evaluate the artistic and technical benefits of the device.

## Kurzfassung

Die zunehmende Zugänglichkeit und Leistungsfähigkeit von eingebetteten Einplatinencomputern und Prototyping-Plattformen hat zur Entstehung mehrerer Frameworks geführt, die speziell für die Erstellung digitaler Musikinstrumente und Audioeffekte entwickelt wurden. Diese Fortschritte haben die Möglichkeit eröffnet, eingebettete Computertechnologien zur Nachbildung seltener oder veralteter Instrumente und Audioeffekte zu nutzen. Im Mittelpunkt dieser Arbeit steht die Entwicklung eines plattformoffenen Gitarreneffektpedals unter Verwendung des Embedded Computing Frameworks von Bela, das es Gitarristen ermöglichen soll, Effekte auf dedizierter Audiohardware zu erstellen und anzuwenden. Es wird der Konstruktions-, Design- und Entwicklungsprozess eines Erweiterungs-capes dokumentiert, das wesentliche Komponenten wie Drehgeber, Schalter und Audioein- und -ausgänge in einer einzigen Schaltung vereint. Es wird ein umfassendes Software-Framework auf Basis von Pure Data vorgestellt, das alle notwendigen Funktionen enthält, um das Gerät in einer Multi-Effekt-Konfiguration zu verwenden und eine Signalkette mit veralteten Audio-Effekten enthält, die um ein physisches Modell der Scanner-Vibrato-Einheit von Hammond herum entwickelt wurde. Schließlich werden Benchmarking und Diskussionen durchgeführt, um die künstlerischen und technischen Vorteile des Geräts zu bewerten.

## Obligatory signed declaration

I hereby confirm and declare that the present Bachelor's thesis/Master's thesis was composed by myself without any help from others and that the work contained herein is my own and that I have only used the specified sources and aids. The uploaded version is identical to any printed version submitted.
I also confirm that I have prepared this thesis in compliance with the principles of the FH JOANNEUM Guideline for Good Scientific Practice and Prevention of Research Misconduct. I declare in particular that I have marked all content taken verbatim or in substance from third party works or my own works according to the rules of good scientific practice and that I have included clear references to all sources.
The present original thesis has not been submitted to another university in Austria or abroad for the award of an academic degree in this form.
I understand that the provision of incorrect information in this signed declaration may have legal consequences.

Date, Signature

## Acknowledgment

Above all, I would like to express my heartfelt appreciation to Athena for standing by me throughout our shared academic journey over the last seven years. Her presence and encouragement have been instrumental in navigating through numerous challenging times and attaining a multitude of my objectives on the academic front.

Moreover, I am deeply grateful to my parents for their unwavering support throughout my educational journey. I also want to express my sincere appreciation to my brother, who opened the door to the world of music and arts for me and has been a constant source of encouragement in all of my academic endeavors. I am also grateful to my grandfather, whose enthusiasm for electronics and do-it-yourself technology ignited my curiosity in this domain and motivated me to delve deeper into its exploration.

Furthermore, I am truly appreciative of my esteemed supervisor, Ao.Univ.Prof. Dipl.-Ing. Winfried Ritsch, whose guidance has been instrumental in shaping my academic path and the development of my thesis. His expertise has been indispensable to my academic journey, and I am extremely grateful for his mentorship.

Finally, I wish to express my appreciation to the Bela team for their outstanding contributions and the invaluable support they offer on the platform. Furthermore, I would like to express my gratitude to the maker community for generously sharing excellent code examples, which form a solid foundation for all of us to build upon.

# Table of Contents

## List of Figures

## List of Abbreviations

| | |
|---|---|
| ADC | Analog to Digital Converter |
| API | Application Programming Interface |
| ARM | Advanced RISC Machine |
| BOM | Bill of Materials |
| CAD | Computer-Aided Design |
| CPU | Central Processing Unit |
| DAC | Digital to Analog Converter |
| DC | Direct Current |
| DHCP | Dynamic Host Configuration Protocol |
| DIY | Do-It-Yourself |
| DMI | Digital Musical Instrument |
| DSP | Digital Signal Processing |
| GND | Ground |
| GPIO | General-Purpose Input and Output |
| GUI | Graphical User Interface |
| HDMI | High-Definition Multimedia Interface |
| I/O | Input and Output |
| I2C | Inter-Integrated Circuit |
| IDE | Integrated Development Environment |
| IP | Internet Protocol |
| LED | Light-Emitting Diode |
| LFO | Low-Frequency Oscillation |
| MCU | Microcontroller Unit |
| MIDI | Musical Instrument Digital Interface |
| O2O | OSC to OLED |
| OLED | Organic Light-Emitting Diode |
| OSC | Open Sound Control |
| PCB | Printed Circuit Board |
| PRU | Musical Instrument Digital Interface |
| Pd | Pure Data |
| RAM | Random Access Memory |

| | |
|---|---|
| RGB | Red, Green, and Blue |
| RMS | Root Mean Square |
| RTS | Ring, Tip, Sleeve |
| SCL | Serial Clock |
| SDA | Serial Data |
| SPI | Serial Peripheral Interface |
| SSID | Service Set Identifier |
| SoC | System on a Chip |
| TRS | Tip, Ring, Sleeve |
| UDP | User Datagram Protocol |
| USB | Universal Serial Bus |
| VCC | Common Collector Voltage |
| WAV | Waveform Audio File Format |

# 1. Introduction

## 1.1 Background and Motivation

Early audio effects, such as delays and reverbs, made use of both mechanical elements and analog electronics. [1] Delays were commonly achieved by running magnetic tape with the help of a motor, while reverbs involved driving a metal plate or spring with an input signal through an actuator and capturing the resulting sound using a pickup. [2] While these concepts produced desirable sounds, their size, weight, and user-friendliness posed challenges for musicians. However, the advancement of digital technology and digital signal processing (DSP) has reached a stage where audio effects can be created using DSP on a large scale, thus providing a compelling alternative for enthusiasts of analog audio devices. [3]

While these technological advancements have led to the digital emulations of many early audio effect units, there has been limited exploration of specific electro-mechanical audio effects, such as Hammond's Vibrato Scanner. This iconic chorus and vibrato effect showcased Hammond's dedication to finding an expressive and emotionally intense sound. It provided a distinct sonic character by infusing depth and movement into the sound of the Hammond tonewheel organ. The unit utilized a delay line with progressively increasing delays between audio filter stages. A scanner connected selected taps from the line to the output, gradually altering the phase shift and creating a vibrato effect. Mixing the dry signal with multiple outputs from the delay line produced a chorus effect. [4]

Only two digital reproductions of the Vibrato Scanner unit are currently known: a virtual effect plugin developed by Martinic (see Figure 1) and another designed explicitly for Logic Pro X (see Figure 2). These plugins accurately simulate the original hardware, closely preserving its unique characteristics. They offer the same three vibrato and chorus presets as the original design but provide improved flexibility, allowing users to adjust rate and depth parameters. The advantage of these replications is that they can be applied not only to organ sounds but also to guitar, synthesizer, or any other desired sound. However, as they work only on personal computers, their usage might be impractical for performance and limited to specific general-purpose operating systems that impose considerable audio latency.

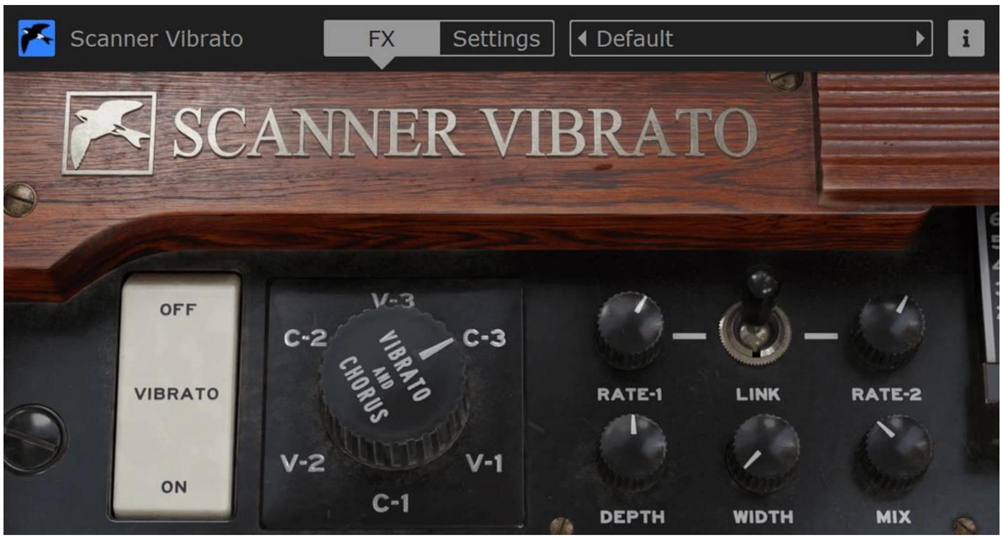


**Figure 1:** Scanner Vibrato virtual effect plugin by Martinic. [5]

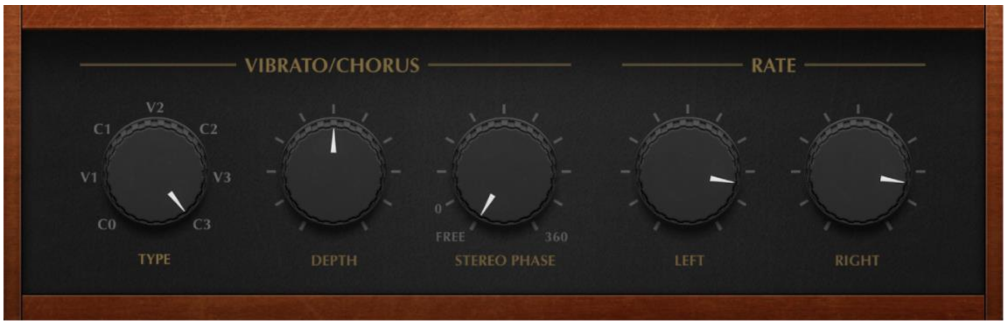


**Figure 2:** Scanner Vibrato virtual effect plugin for Logic Pro X. [6]

## 1.2 Simulacra

When specific instruments or technologies become rare or unavailable, they can be substituted with simulacra. [7] Unlike simulations, simulacra do not aim to imitate all specific behaviors or processes but instead focus on capturing the essential characteristics while not necessarily replicating the exact outward appearance. [8] In software design alone, the relevant term would be an emulator. However, given the inclusion of the physical interface, which is not replicated, the term simulacrum corresponds more accurately to the intended objective in this context. [9]

Utilizing modern embedded computer technologies to create simulacra presents a potential method for substituting rare or outdated instruments and audio effects. The expanding accessibility and growing power of single-board computers and prototyping platforms have led to numerous frameworks tailored explicitly for designing and building digital musical instruments and audio effects. When considering any particular project, the suitability of each environment is determined by various factors, including latency, workload measurements, documentation, and software implementation. Choosing an appropriate platform can be challenging, as it involves trade-offs between computing power, hardware connectivity, programming ease, and price. [10]

## 1.3 Scope of Project

This thesis assesses the practicality of employing embedded computer technologies to replicate rare or obsolete audio effects. The proposed approach involves developing a portable open-platform guitar effects pedal, leveraging Bela's embedded computing framework. To achieve this, a specially designed extension cape is implemented for Bela Mini, integrating essential components like rotary encoders, switches, and audio inputs/outputs onto a single circuit. Additionally, a comprehensive software framework is established using Pure Data, providing all the necessary functionalities to utilize the device in a multi-effect configuration and incorporating a signal chain of outdated audio effects.

Moreover, the project is intended to be open-source, ensuring that both software and hardware components are freely available to the public. This involves utilizing exclusively open-source technologies for developing all software and hardware aspects of the device and providing access to all essential data, such as schematics, CAD files, bills of materials, and the software framework, to the public.

# 2. Open-Source Ecosystem

## 2.1 Open Design Movement

Open-source software has gained significant attention as an alternative software development and distribution approach. It has proven successful in creating high-quality software at minimal cost. Open-source projects are characterized by a loosely connected community united by shared values of software freedom, allowing anyone to use, study, modify, and distribute the source code. This model provides various benefits, including flexibility, suitability, support, stability, and maintenance. [11]

While initially limited to the software industry, the open-source movement has expanded to other fields, such as open data, open education, and open hardware, driven by global digitization and widespread access to affordable internet services. An expanding community of DIY enthusiasts, inventors, artists, scientists, and individuals from various backgrounds is adopting personal manufacturing and collaborative networks to bring about transformative changes in their surroundings. This trend has been facilitated by the accessibility of affordable digital fabrication tools, open-source hardware, and a culture of knowledge sharing, collaboration, and support through virtual and physical communities. This movement signifies a shift towards peer production and the democratization of technology. [12]

## 2.2 The Maker Culture

The term “maker community” refers to a community that revolves around utilizing emerging technologies and practices to repurpose and modify existing materials, ultimately creating something new. Maker culture has its roots in the hacker culture of the 1960s-1970s and the DIY amateur radio movements of the early 20th century. These movements emphasized collaborative production and the development of open and customizable technologies. Makers embrace this alternative approach to technology production, made feasible by crowdfunding infrastructures and startups. Some maker communities focus on specific open-source hardware or software platforms, with Arduino being a notable example. In addition to the usual motivations for contributing to open-source or free software, the sustainability of this community is strengthened by the incorporation of expansion boards, resulting in a wide range of extensions and variations. [13]

## 2.3 Open-Source Hardware

Open-source hardware allows anyone to access, modify, create, and distribute the design specifications of a physical object. It is not limited to a specific object type and can be applied to various items like cars, chairs, computers, robots, and houses. Like open-source software, open hardware provides access to the source code, including schematics, blueprints, and CAD files. Users can modify the code, improve software functionality, and even change the physical design, which promotes accessibility and encourages sharing knowledge and modifications. Open hardware removes barriers to design and manufacturing, enabling more people to participate and collaborate in hardware development. [14]

According to the Open-Source Hardware Association, hardware must meet a set of criteria to be considered open-source. These include providing comprehensive design files and accessible documentation, clearly stating the open-source parts of the design, releasing software under an open-source license (if applicable), allowing modification and distribution of derived works, enabling unrestricted sharing of project documentation, requiring proper attribution to original licensors, avoiding discrimination, not restricting hardware use in specific fields, ensuring the license applies to all parties without additional licenses, granting rights to extracted portions of the work, avoiding limitations on non-derivative items, and remaining independent of specific technologies. Adhering to these requirements supports the principles of openness, collaboration, and innovation in the Open-Source Hardware movement. [15]

## 2.4 Open-source Maker Platforms

### 2.4.1 Microcontroller Platforms

Over the last few years, there has been a rise in the availability of open-source microcontroller platforms capable of audio processing and synthesis. Arduinos have played a crucial role in popularizing microcontrollers by increasing their accessibility through a high-level programming language, specialized libraries, and an Integrated Development Environment (IDE). The impact of the Arduino on computer music and the New Interfaces for Musical Expression community has been profound, resulting in the creation of numerous new music controllers and instruments. [16] Microcontroller platforms have convenient hardware sensor connections and reliable timing but limited computational capabilities. However, software libraries like Mozzi enhance the microprocessor’s capabilities, enabling it to generate complex sounds using standard synthesis units such as oscillators, samples, delays, filters, and envelopes. [17]

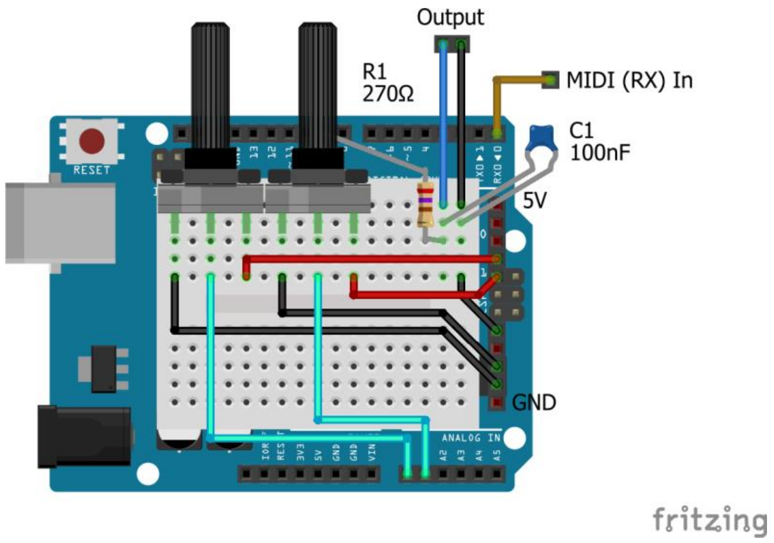


**Figure 3:** Arduino FM MIDI Synthesis with Mozzi. [18]

### 2.4.2 Embedded Linux Platforms

Simultaneously, the emergence of embedded Linux platforms, such as the Raspberry Pi and BeagleBone, has also opened up possibilities for audio signal processing and transformed the approach toward Digital and Augmented Musical Instruments. Thanks to these systems, users could process audio in a manner similar to desktop computers using user-friendly tools like PureData, ChucK, and SuperCollider. [19] However, the convenience of operating systems came at the cost of drawbacks, such as hardware and software complexity, audio latency, and efficiency. Moreover, none of these boards were initially designed with real-time audio applications in mind. They lacked adequate audio input/output capabilities, requiring external audio interfaces connected via USB [20] or additional audio codec subsystems. [21]

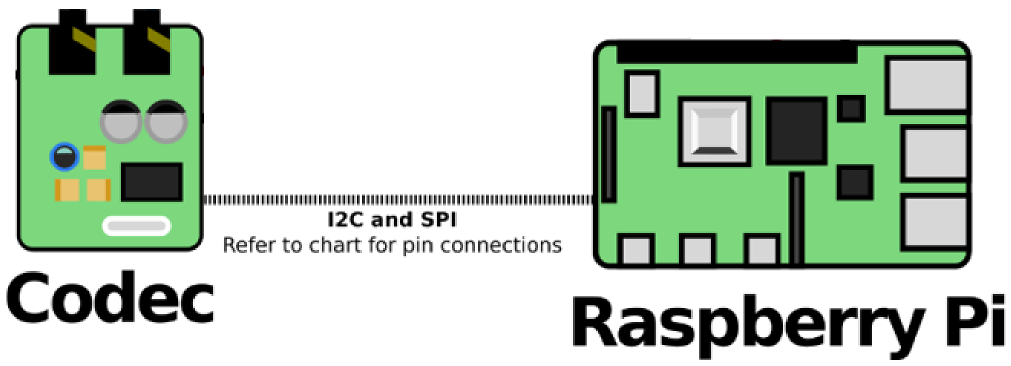


**Figure 4:** Raspberry Pi Audio Codec Subsystem. [22]

## 2.5 Specialized Hardware and Software Solutions

In response to these challenges, there has been a rise in specialized hardware and software solutions based on single-board computers and high-performance microcontrollers. These solutions come with integrated audio I/O hardware and tangible control elements similar to popular electronic instruments such as guitar pedals and synthesizers. Their design provides users with an intuitive interface and a familiar performance environment. These instruments are highly versatile and can easily switch between different roles, much like laptops. They work in tandem with specialized Linux distributions, where audio processing tasks are performed outside of the operating system. This action allows for smaller buffer sizes and reduced audio latency. Programming these instruments is a straightforward process, as users can transfer pre-built sketches created in platforms such as Max and Pure Data or written in C++ onto the device using a USB connection. Moreover, community-developed libraries containing pre-existing patches and scripts serve as a valuable resource for ensembles whose members may lack the inclination or ability to program their own software instruments. [23]

### 2.5.1 The OWL

The OWL (see Figure 5) is programmable hardware designed by Rebel Technology for real-time audio signal processing. It provides musicians with a versatile tool for creating and manipulating audio effects during live performances. The OWL features a powerful ARM Cortex M4 microcontroller with a floating-point unit for precise audio processing. It incorporates 1Mb of high-speed SRAM, offering expanded memory capacity for seamless audio data storage and manipulation. Equipped with a high-quality stereo audio codec supporting 24-bit resolution and a sampling rate of up to 96 kHz, the OWL ensures exceptional audio quality. [24]

Designed as a dedicated audio device, the OWL replaces laptops for on-stage audio processing. Musicians can program their own code using C++ to create custom audio effects, eliminating the need for complex computer music environments. Its tangible format resembles a traditional stomp box and seamlessly integrates into a guitarist's pedal board setup. The OWL embraces an open-source approach, providing an API for users to develop their audio effects applications. This approach encourages a community of users and programmers to freely share effects patches, collaborate, and enhance the device's capabilities. [25]

**Figure 5:** The OWL Pedal. [26]

### 2.5.2 Daisy

The Daisy hardware is based on the Daisy Seed, an embedded system with an ARM Cortex-M7 STM32H750 MCU processor. The Daisy Seed's MCU processor can operate at 480MHz and handle complex Digital Signal Processing algorithms. The built-in AK4556 Codec offers 24-bit AC-coupled converters, while internal signal processing operates at 32-bit floating point precision, at 48kHz or 96kHz sampling rates, with configurable audio block sizes ranging from 48 to single sample frames. This flexibility allows throughput latencies as low as 0.01ms while maintaining the capability to perform complex algorithms. In addition to audio functionality, the Daisy Seed supports non-audio analog pins for controlling knobs, switches, LEDs, and gate voltages. These pins are sampled at the same rate as audio, and some I/O pin applications, like gate outputs, operate with the same throughput latency as audio signals. [27]

Due to its small size, the Daisy Seed is ideal for breadboarding development projects and embedding within Digital Musical Instruments. ElectroSmith, the company behind Daisy, distributes various hardware configurations, including the Daisy Patch for Eurorack modular synthesizers, the Daisy Pod (see Figure 6) for stompbox pedals, and the Daisy Field for desktop applications. The development of Daisy firmware is supported by popular programming platforms, including Arduino, FAUST, PureData, and Max/gen~ through the Heavy framework and Oopsy software. [28]

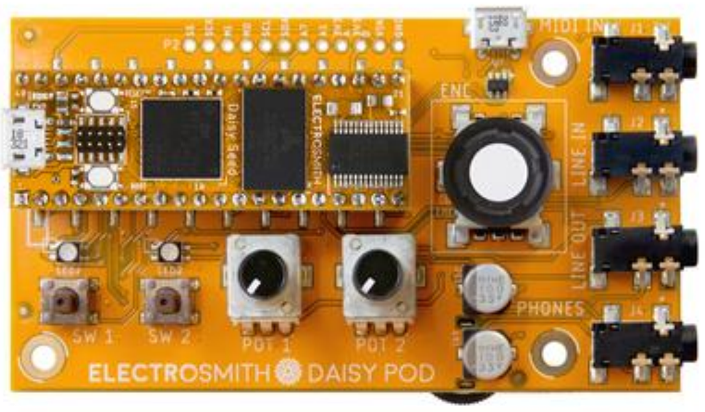


**Figure 6**: Daisy Pod breakout board. [29]

### 2.5.3 Satellite CCRMA

Satellite CCRMA (see Figure 7) is a widely used and well-supported platform that focuses on creating an efficient audio environment for embedded Linux computers. It consists of either a Beagle Board xM or a Raspberry Pi Model B, along with an Arduino microcontroller and a breadboard for electronics prototyping. The platform comes with preinstalled Linux audio applications like Pure Data, Faust, and ChucK, which offer various audio functionalities. Its main objective is to provide a compact and standalone embedded prototyping platform. Satellite CCRMA is a versatile tool that aids in designing musical interactions and facilitates the development of innovative musical instruments and sound installations. [30]

The utilization of satellite CCRMA formed the basis for developing a wide range of musical instruments and installations, each designed with unique functionalities and interactive capabilities. Projects include independent musical instruments that can function autonomously and enable musicians to control and produce sound through embedded sensors. It was also employed in multichannel audio setups, utilizing an external sound interface, as well as in implementing real-time audio effects within stompboxes. Additionally, video-based interactive installations were implemented with the help of this platform, employing webcams and water to generate sound samples. [31]

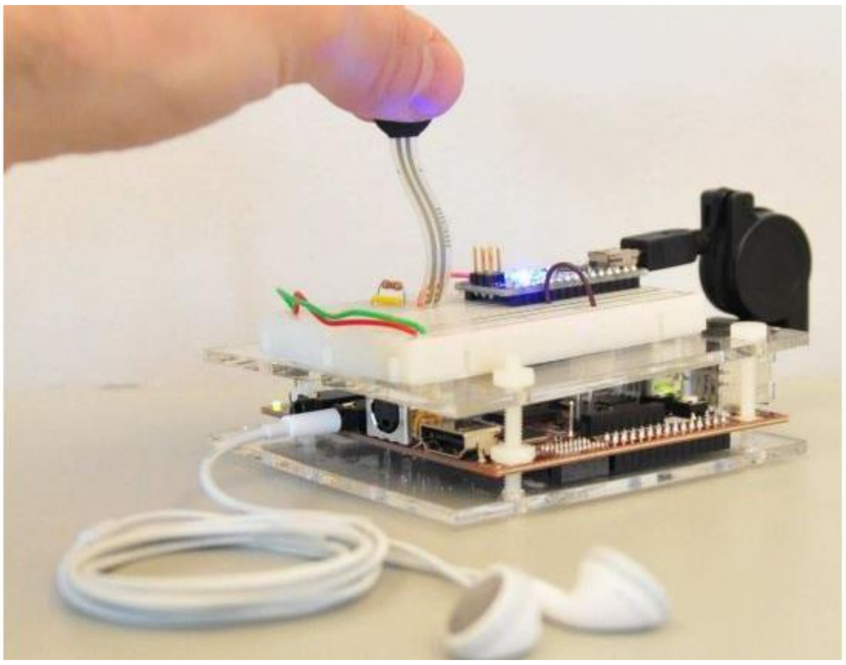


**Figure 7:** Satellite CCRMA example prototype. [32]

### 2.5.4 Organelle

The Organelle by Critter & Guitari (see Figure 8) is a compact synthesizer designed for sonic exploration and musical experimentation. Built around a Raspberry Pi embedded computer module, the Organelle can run the Pure Data programming environment, allowing musicians to synthesize, sample, apply effects, and everything in between. Powered by a 1.2GHz ARM Cortex A53 quad-core processor and 1GB RAM, running on a Linux operating system, it boots up in approximately 12 seconds. It delivers a sampling rate of 44.1kHz and 16-bit resolution, ensuring high-quality sound both on input and output. Its buttons, knobs, and keys are fully mappable, providing a customizable instrument that adapts to anyone's creative needs. [33]

The Organelle offers connections such as input and output jacks, USB ports, MIDI, an HDMI output port, and a footswitch jack, enhancing its versatility and integration capabilities. The makers of Organelle encourage community engagement by providing a platform for sharing their creations with others. This open-source approach allows for an ecosystem where musicians and programmers can collaborate, exchange ideas, and elevate the instrument's capabilities. With its user-friendly interface, powerful abilities, and community-oriented approach, the Organelle empowers musicians to unleash creativity and explore uncharted sonic territories, all within a compact and intuitive package. [34]

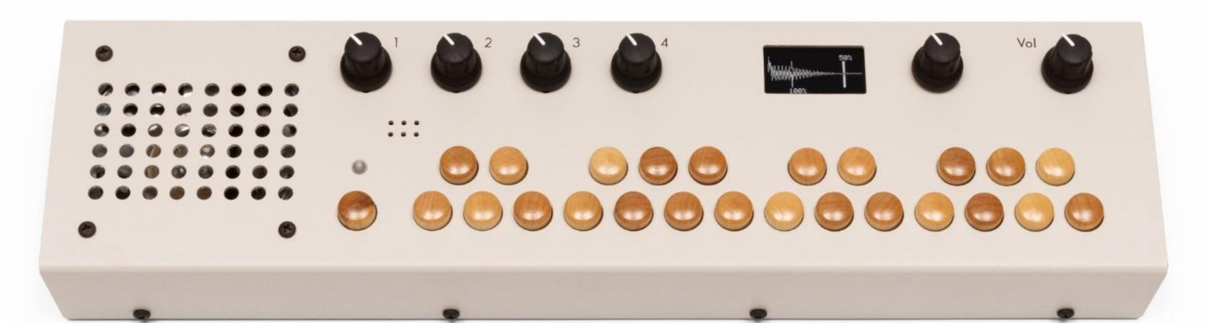


**Figure 8:** Organelle M. [35]

### 2.5.5 Prynth

The Prynth project combines hardware and software elements to create a platform for prototyping and developing sound synthesizers. The hardware comprises a Raspberry Pi and a custom cape with a Teensy microcontroller for analog and digital I/O functionalities. Instead of providing built-in audio capabilities, Prynth offers a pre-configured JACK API that allows users to utilize any audio interface in the system. The software is a custom headless Linux distribution that includes an IDE capable of running the SuperCollider programming language and handling onboard audio processing and mapping. This feature enhances usability and facilitates remote control and configuration of the instrument. [36]

The main objective of the Prynth project is to create user-friendly technologies and comprehensive guides for assembling musical instruments while fostering knowledge exchange and encouraging discussions among creators. The Prynth framework facilitates easy prototyping and instrument development by employing a plug-and-play approach for interfacing analog sensors with a sound synthesis engine. The provided documentation offers sufficient guidance for building instruments from scratch using the framework. Prynth significantly simplifies the instrument-building process, reducing setup time and integrating the sound synthesis engine of a DMI within the interface. [37]

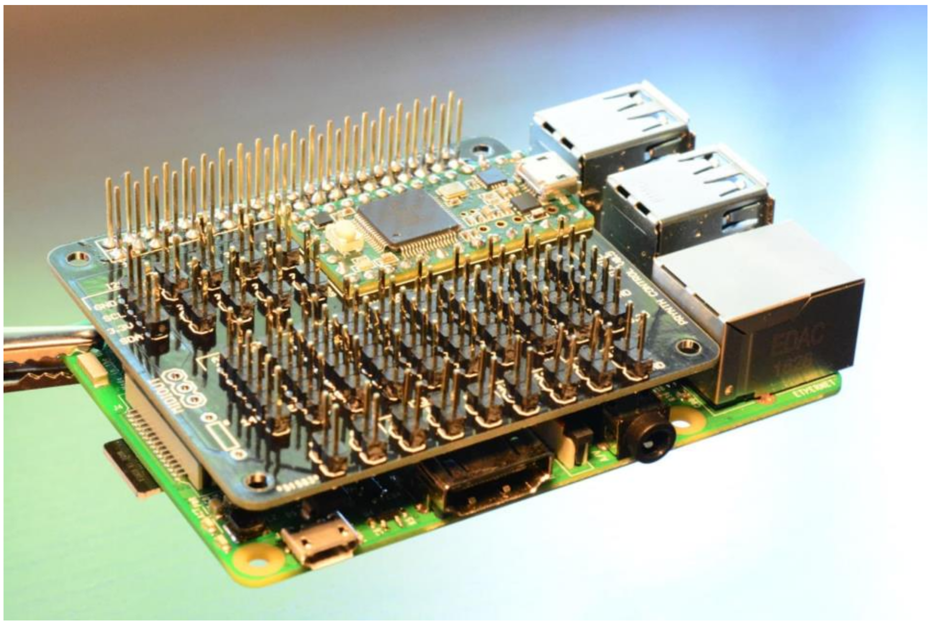


**Figure 9:** Prynth Control Board Attached to Raspberry Pi. [38]

# 3. The Bela platform

## 3.1 Bela

The Bela platform is a comprehensive hardware and software environment that combines advanced audio and sensor processing features. It was developed by the Augmented Instruments Laboratory at Queen Mary University of London to fulfill the requirement for a robust embedded instrument platform. The hardware is built on the BeagleBone Black single-board computer with an integrated hardware extension known as the Bela cape (see Figure 10). This expansion module provides essential functionalities for audio, analog, and digital input/output, significantly enhancing audio-rate analog input and output capabilities. It also enables efficient low-latency audio processing, making it an excellent choice for various applications. Bela supports the execution of code compiled from multiple programming languages, including C++, Csound, SuperCollider, Faust, and Pure Data patches. [39]

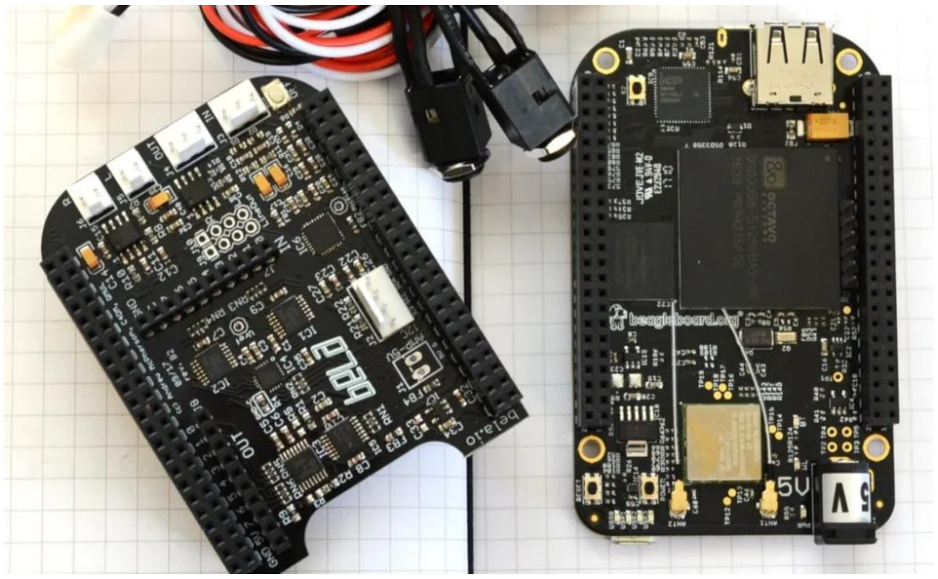


**Figure 10:** Bela cape and BeagleBone Black. [40]

### 3.1.1 Hardware Components

The Bela system comprises two main hardware components: the Beaglebone Black and the Bela cape. The Beaglebone Black is based on the Texas Instrument Sitara AM3358 system-on-chip (SoC), integrating a single-core ARM Cortex-A8 1 GHz CPU. It also incorporates two 200 MHz Programmable Real-Time Units (PRUs), which are microcontrollers optimized for real-time operations. The Bela cape provides stereo audio input and output, 16-bit analog I/O with eight channels, and sixteen digital GPIO pins (see Figure 11). This combination of hardware components forms the foundation of the Bela system. [41]

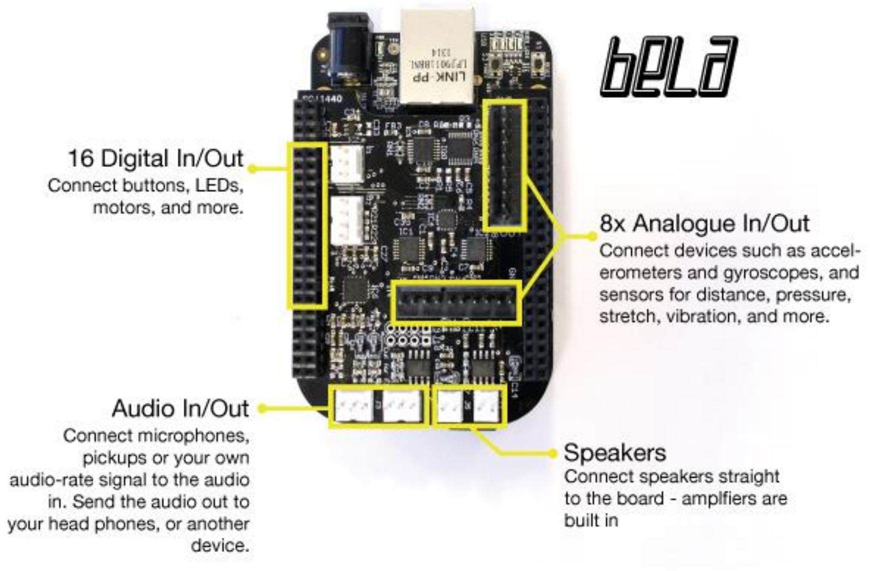

16 Digital In/Out
Connect buttons, LEDs, motors, and more.
8x Analogue In/Out
Connect devices such as accel-erometers and gyroscopes, and sensors for distance, pressure, stretch, vibration, and more.
Audio In/Out
Connect microphones, pickups or your own audio-rate signal to the audio in. Send the audio out to your head phones, or another device.
Speakers
Connect speakers straight to the board - amplfiers are built in

**Figure 11:** The Bela Hardware. [42]

### 3.1.2 Software Environment

The Bela software environment, running on Debian Linux with Xenomai9 real-time kernel extensions, optimizes performance by enabling the direct transfer of audio and sensor data to the hardware, ensuring low-latency audio processing (see Figure 12). Jitter is eliminated by automatically sampling analog and digital channels at audio rates. Combining Xenomai with the Programmable Real-Time Units (PRUs), the Bela system achieves reliable hard real-time performance with round-trip latencies of less than 100 µs for analog I/O and less than 1 ms for audio I/O. The high-priority threads of Xenomai effectively utilize CPU time, resulting in uninterrupted audio processing even with other Linux services running in parallel. [43]

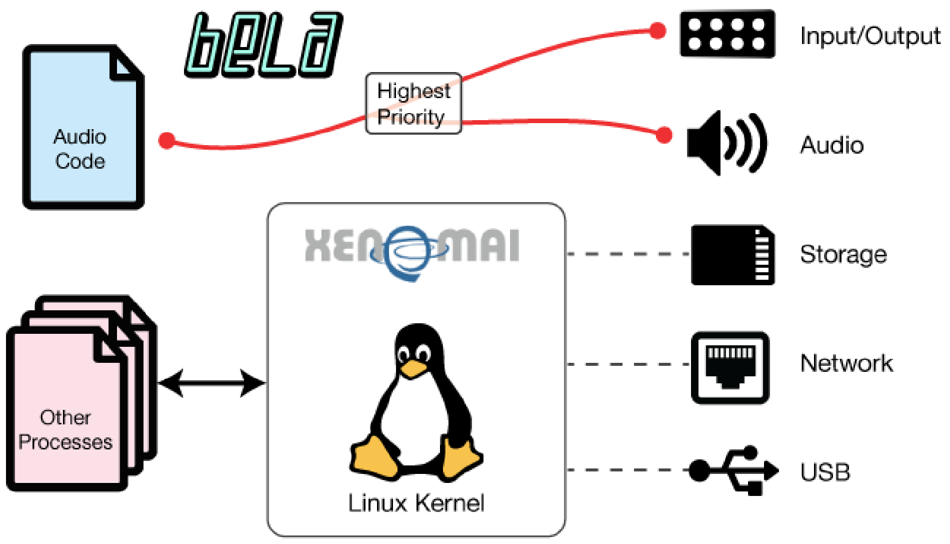
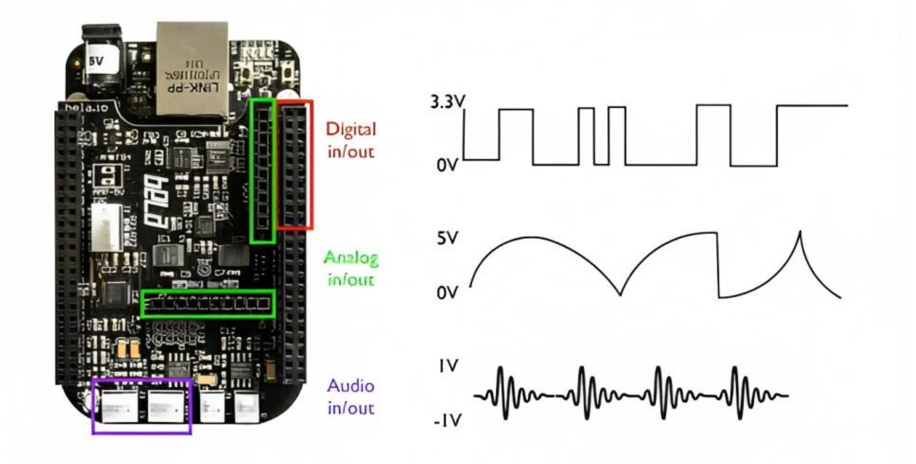


**Figure 12:** The Bela Software. [44]

### 3.1.3 Audio and Sensor Processing

The Bela system delivers high audio and sensor performance (see Figure 13), making it suitable for real-time responsive applications. It enables audio sampling at 44.1 kHz and each of the eight sensors' ADC and DAC channels at 22.05 kHz, allowing for capturing fine details and precise temporal profiles within sensor signals. The Bela system achieves low-latency and highly responsive audio applications with buffer sizes as small as two audio samples. This performance is a notable advantage compared to typical general-purpose operating systems that often require 32 samples or more audio buffer sizes. Furthermore, the Bela platform offers alternative sensor ADC and DAC formats. Instead of sampling eight channels at 22.05kHz each, users can sample four channels at 44.1kHz or two at 88.2kHz. [45]

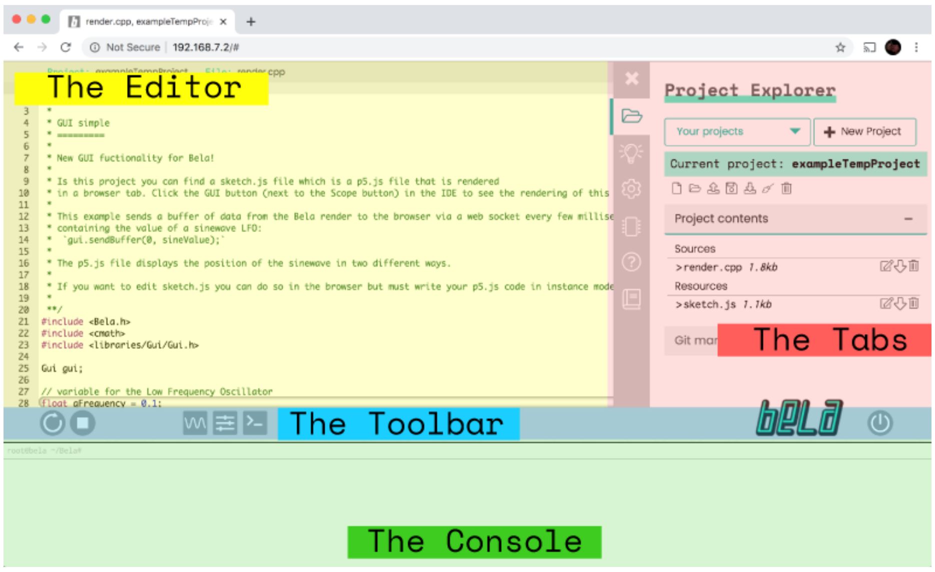


\

**Figure 13:** Audio and Sensor Processing on Bela. [46]

### 3.1.4 Integrated Development Environment

The platform is built to offer a seamless plug-and-play experience for Digital Musical Instruments (DMIs) designers. The Bela Integrated Development Environment (IDE) provides easy access to parameters frequently adjusted during the instrument design process, enhancing the workflow for DMI designers. [47] It is a user-friendly interface offering features such as a text editor, project and file manager, interactive pin diagram, API reference, and examples showcasing different techniques and fully functional digital musical instruments. It also offers an oscilloscope tool for visualizing time and frequency domain signals and a graphical user interface for creating virtual controls and visualizations for any project. [48]



```
 *
 * GUI simple
 * =========
 *
 * New GUI fuctionality for Bela!
 *
 * Is this project you can find a sketch.js file which is a p5.js file that is rendered
 * in a browser tab. Click the GUI button (next to the Scope button) in the IDE to see the rendering of this
 *
 * This example sends a buffer of data from the Bela render to the browser via a web socket every few millise
 * containing the value of a sinewave LFO:
 *  `gui.sendBuffer(0, sineValue);`
 *
 * The p5.js file displays the position of the sinewave in two different ways.
 *
 * If you want to edit sketch.js you can do so in the browser but must write your p5.js code in instance mode
 *
 **/
#include <Bela.h>
#include <cmath>
#include <libraries/Gui/Gui.h>

Gui gui;

// variable for the Low Frequency Oscillator
float aFrequency = 0.1;
```

```
>render.cpp 7.8kb
>sketch.js 7.1kb
```

```
root@bela ~/Bela#
```

**Figure 14:** Bela Integrated Development Environment. [49]

## 3.2 Comparison to Similar Platforms

Compared to microcontroller-based platforms, such as Daisy or OWL, Bela offers greater processing power while providing real-time performance. It also shows the advantages of a whole Linux operating system and remains minimally affected by system load. Unlike Organelle or Prynth, primarily designed as programmable sound synthesizers, Bela is more versatile and serves as a general-purpose solution. Unlike the CCRMA Satellite, which operates on a Raspberry Pi and relies on standard Linux audio drivers, the Bela software utilizes Xenomai extensions on the BeagleBone Black CPU. This approach guarantees consistent and reliable performance, even achieving sub-millisecond latency, despite other processes on the board. Additionally, Bela provides a more significant number of audio and sensor input/output options compared to commonly available Raspberry Pi hats. Subsequently, due to its ample processing power and extensive I/O capabilities, Bela fully meets the requirements of a self-contained device suitable for embedding in standalone Digital Musical Instruments or sounding objects. [50]

# 4. Building the Pedal

## 4.1 Idea

The concept for the pedal was to achieve maximum compactness, resembling a stompbox pedal like the OWL. However, incorporate fully customizable buttons and knobs like the Organelle interface. Its design should include visual indicators to provide feedback on parameter adjustments and the selection of effects, enabling a versatile multi-effect configuration. Moreover, the pedal should be compatible with any pedal board configuration by supporting standard guitar pedal power supplies. Additionally, the hardware should be open-source, making all essential data, including schematics, CAD files, bills of materials, etc., accessible to the public.

### 4.1.1 Considerations

To ensure maximum compactness, the pedal is built around the Bela Mini, which is based on the PocketBeagle and is significantly smaller than the original Bela board. Despite its smaller size, the Bela Mini still offers the same processing capabilities as the standard Bela board based on the BeagleBone Black. However, unlike the normal Bela, the Bela Mini does not include analog sensor outputs, line-level audio outputs, and ethernet connection (see Figure 15). [51] Furthermore, to maintain the project's open-source nature, all the data and information essential for reproducing the project are published on platforms like GitHub and similar repositories.

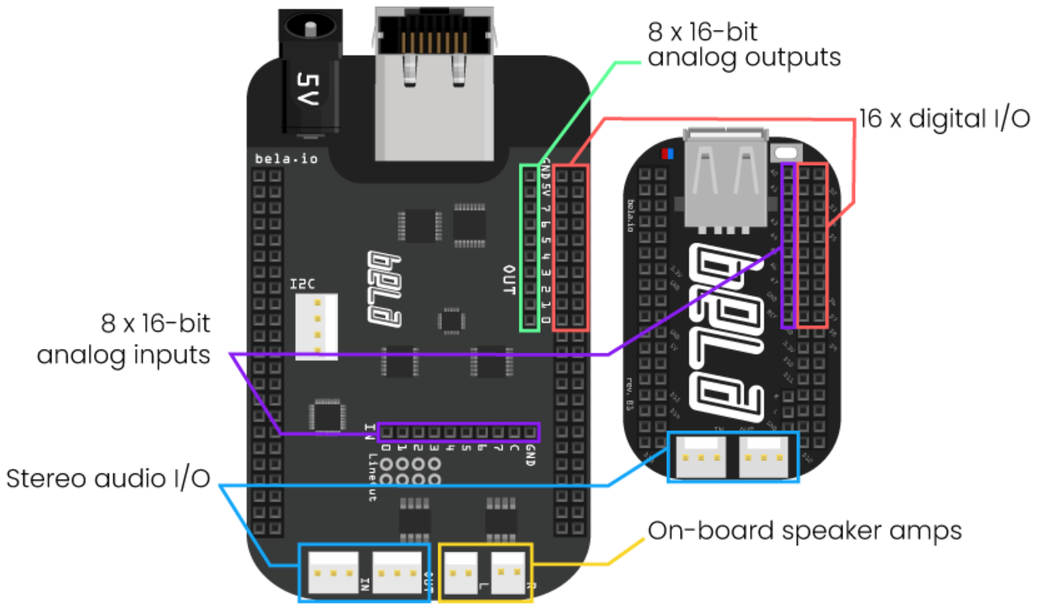


**Figure 15:** Comparison between Bela and Bela Mini. [52]

### 4.1.2 First Prototype

The first prototype was set up on a breadboard, connecting all the sensors via jumper cables to the Bela Mini. It incorporated four analog potentiometers that allow the control of one effect at a time and an on-off-on momentary toggle switch to skip back and forth between effects (see Figure 16). However, as the analog input values, such as potentiometers, are handled at the sampling rate and received continuously on Bela, they impose their initial value on the chosen parameters, making it impossible to retain the values. Consequently, the utilization of analog potentiometers in a multi-effect setting becomes inefficient.

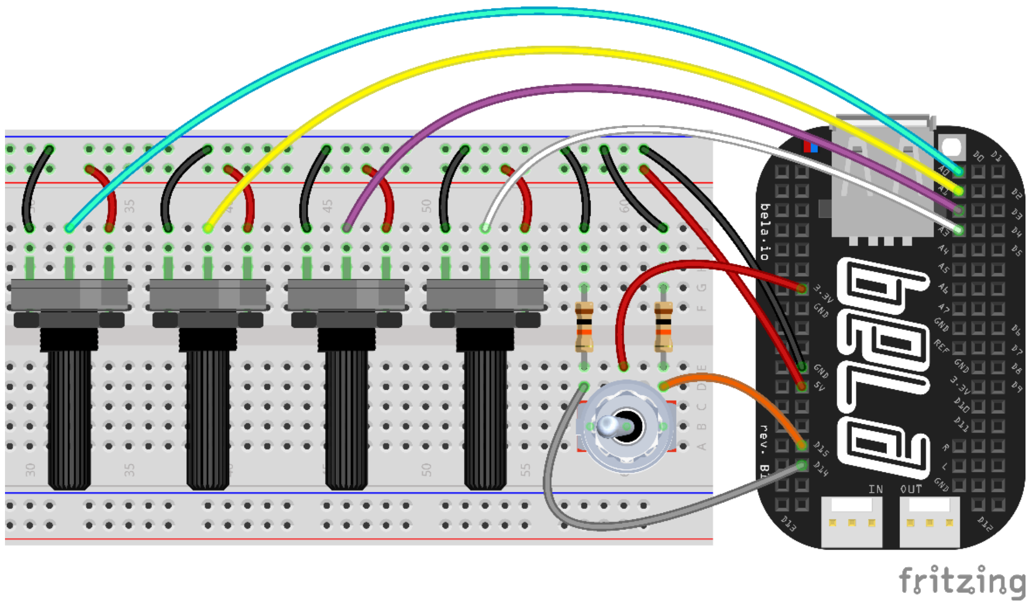


**Figure 16:** First Prototype.

## 4.2 Rotary Encoder

Rotary encoders are a viable alternative to potentiometers in digital audio equipment like audio mixing boards, sound processing equipment, instrument amplifiers, and guitar effects pedals. They offer significant advantages over potentiometers, including higher resolution due to generating digital signals that can be accurately measured and counted. In addition, rotary encoders generate digital signals that can be stored and retrieved, making them suitable for a multi-effect setting and extending individual control functionalities. [53]

### 4.2.1 Working Principle

A rotary encoder is an electro-mechanical device that detects and measures rotational motion by dividing a complete revolution into a limited number of discrete positions. The unit features three pins labelled A, B, and C. Inside the encoder, there are two internal switches. One connects pin A to pin C, while the other links pin B to pin C. These switches can be in either an open or closed state for each encoder position. As the encoder is rotated, clockwise or counterclockwise, the internal switches are activated and change their states. Consequently, the electrical connection between the pins opens or closes depending on the direction of rotation. As the encoder rotates, it produces a series of square wave pulses, which help determine the total number of cycles and their direction (see Figure 17). [54]

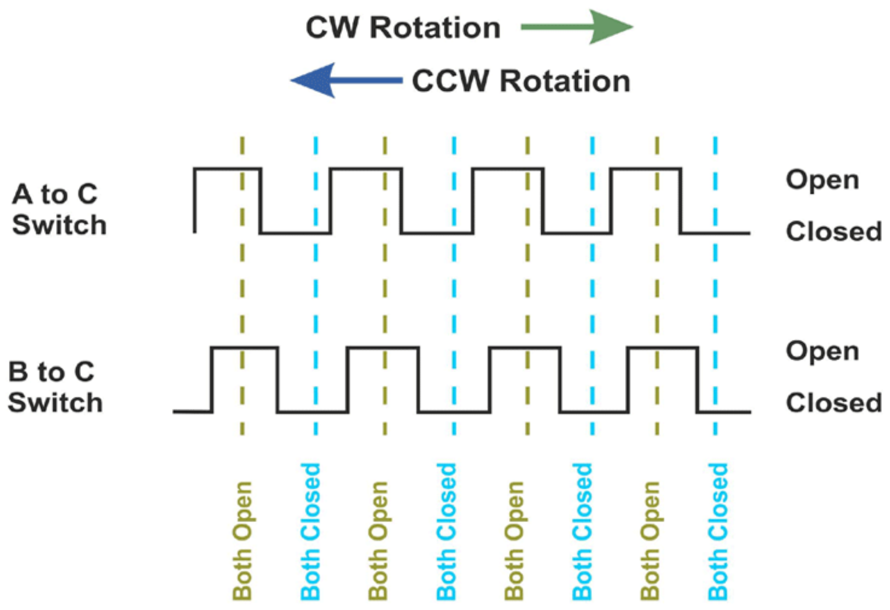


**Figure 17:** Rotary Encoder Working Principle. [55]

### 4.2.2 Encoder Switch

Moreover, the encoder is equipped with an additional switch. This switch is separate from the rotational function and is activated by pushing down the encoder shaft. The switch has two pins located on the opposite side of the encoder, typically referred to as S1 and S2. Within the switch, two metal components are linked to each of its terminals. These metal contacts do not touch in the unpressed state, resulting in an open circuit. However, when the shaft of the encoder is pressed, the two metal pieces come into contact, closing the circuit and enabling the flow of electricity. This mechanism establishes the two distinct states: 0 for the unpressed state and 1 for the pressed state, which can be read using Bela's digital pins. [56]

### 4.2.3 Pull-up Resistors

Bela Mini provides 16 digital pins that can be configured as either inputs or outputs. When configured as digital inputs, these pins can read up to 3.3V when on or 0V when off. Connecting an encoder, including the switch to the digital inputs of Bela, requires the addition of pull-up resistors in conjunction with pins A, B, and S1, while pins C and S2 are connected directly to the ground (see Figure 18). The purpose of these resistors is to maintain a stable reading of 3.3V when the circuit is open, which would otherwise fluctuate randomly between the minimum and maximum values. [57]

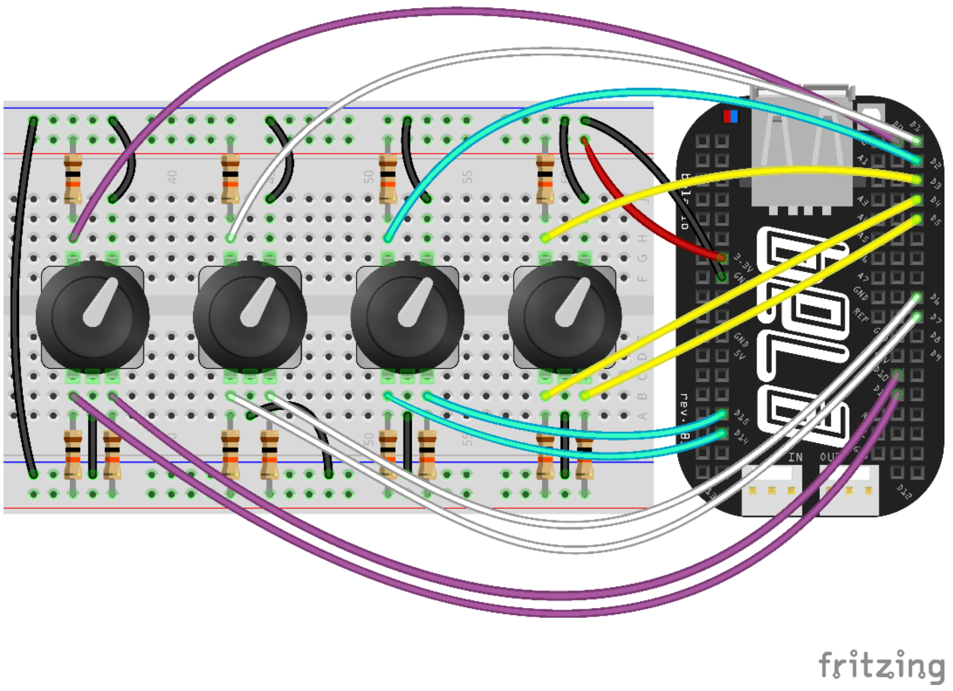


**Figure 18:** Rotary Encoder Connection.

### 4.2.4 Encoder Circuit

The circuit (see Figure 19) has been designed to produce a low signal when the switches are closed and a high signal when the switches are open. When they are closed, a ground connection is established at pins C or S2, which is then transmitted to the A and B pins or the S1 pin correspondingly, resulting in a low output. On the other hand, when the switches are open, a 3.3V supply input and 10k pull-up resistors are utilized to generate a high output, ensuring that both A, B and S1 are set to high. [58]

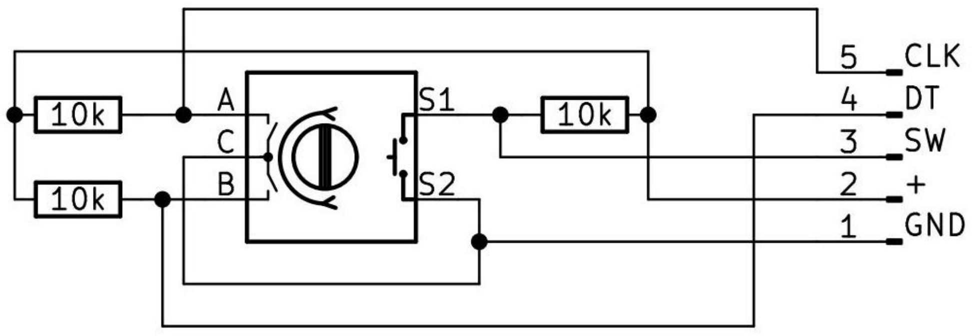


**Figure 19:** Rotary Encoder Circuit.

## 4.3 Momentary Footswitch

In addition to the four rotary encoder switches, two additional momentary footswitches were added to the circuit (see Figure 20). Unlike latching footswitches, which have two states and function as on/off switches, momentary footswitches act like push buttons, remaining active only as long as they are pressed. Latching footswitches are extensively used in analog effect pedals, primarily for bypass. Momentary switches, on the other hand, prove to be more advantageous in digital signal processing applications since their functions can be assigned through software rather than being predetermined in the hardware. This flexibility allows a single switch to be allocated for various functions, including bypass, tap tempo, and even looper functionalities like record, play, stop, and clear. [59]

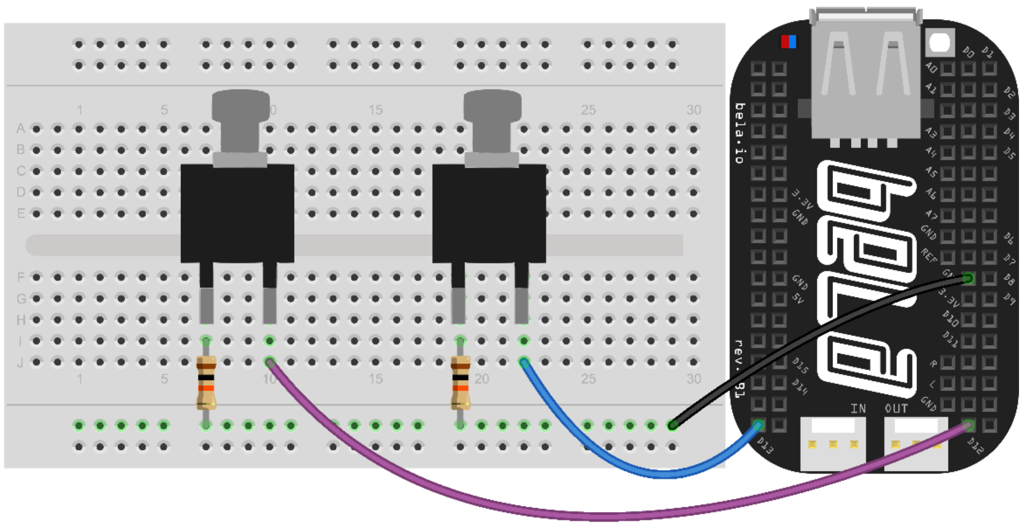
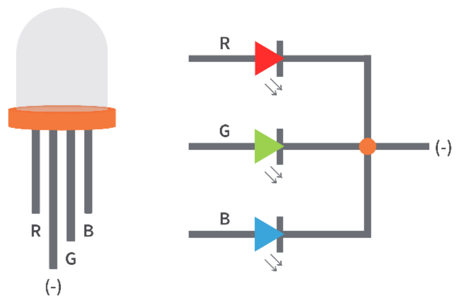


**Figure 20:** Momentary Footswitch Connection.

## 4.4 LED

The initial proposal for the visual indicator involved utilizing a four-pin RGB LED (see Figure 21 (a)) to showcase the chosen effect's color. However, this became problematic to implement as the Bela Mini version lacked the necessary analog outputs to drive such an LED. Another RGB solution could have been a NeoPixel LED (see Figure 21 (b)) that is usually controlled via a Serial Peripheral Interface (SPI) bus. This communication type can be set up with the Bela Mini, but it requires sacrificing either the audio input or output, making it an unreasonable solution for this project.



(a) Common Cathode RGB LED. [60]

(b) NeoPixel RGB LED. [61]

**Figure 21:** Common Cathode and NeoPixel RGB LEDs.

The final decision fell on implementing a bicolor red and blue 5mm LED that can also produce purple by alternating between the two colors very quickly. By achieving three different colors, this LED is versatile enough to provide simple visual feedback for different states and changes. To operate the LED, a voltage of 3.3V needs to be supplied to both of its anodes while its common cathode is connected to the ground. As mentioned earlier, Bela Mini provides 16 digital pins that can be configured as either inputs or outputs. When configured as digital outputs, these pins can output a voltage of 3.3V when turned on or 0V when turned off. To prevent the LED from receiving excessive current and subsequently burn out, a 240Ω resistor is connected in conjunction with its cathode (see Figure 22). [62]

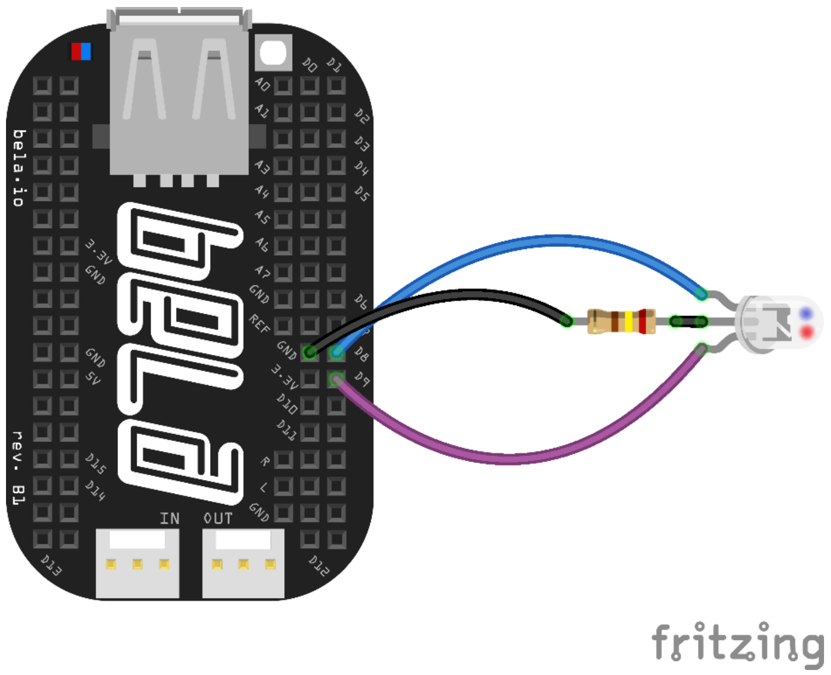


**Figure 22:** Bicolor LED Connection.

## 4.5 OLED

With the introduction of rotary encoders, there was a need to provide additional and more complex visual feedback besides the bicolor LED. Unlike potentiometers that have a limited rotational range that can be sufficiently estimated without additional indicators, rotary encoders have infinite rotation capability, making it impossible to track their values accurately without some form of visual feedback. To visualize the values of each parameter controlled by the encoders, an additional OLED screen was incorporated.

OLED (Organic Light Emitting Diode) comes in a wide range of options in terms of resolution, color, size, and driver requirement. The selected model for this project is a 96-inch monochrome SSD1315 OLED screen with a resolution of 128x64 pixels, which can be controlled using the I2C digital communication protocol. I2C provides a convenient method for sending and receiving data from a network of sensors or devices. One of the key advantages of I2C is its simplicity, as it only requires two wires for communication (data and clock), in addition to power and ground connections. To wire up the screen to Bela, the Data line (SDA) and Clock line (SCL) are connected to the SDA and SCL pins of the I2C2 bus on the board, while the GND and VCC pins to any of the available grounds and 3.3V supplies respectively (see Figure 23). [63]

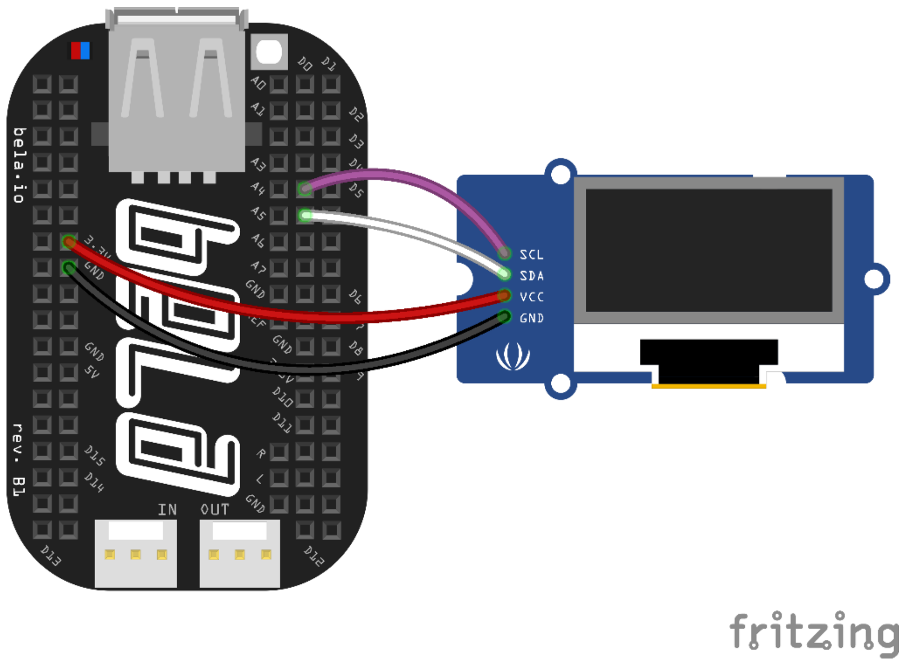


**Figure 23:** OLED Screen Connection.

## 4.6 External Power Supply

### 4.6.1 Setup

To ensure compatibility with external wall plugs or pedal board power supplies, a DC Barrel Jack was incorporated that accommodates either 2.1 mm or 2.5 mm plugs. Externally powering the board requires a power supply of 5V connected between one of the GND pins and P1.01. [64] However, this method has proven unstable, leading to unexpected reboots and incomplete boot-ups. [65]

On the other hand, the PocketBeagle provides built-in support for batteries, allowing for an external power supply ranging from 2.75V to 5.5V. To ensure the proper functioning of the audio codec, the board requires a minimum of 4.5V and 100mA. To connect the power supply to the battery input (see Figure 24), the sleeve of the DC Barrel is connected to the positive input labelled BAT+ (P2 – Pin 14), while its tip is connected to the ground reserved for the battery input (P2 – Pin 15). [66] This connection results in a negative polarity commonly used in guitar effect power supplies.

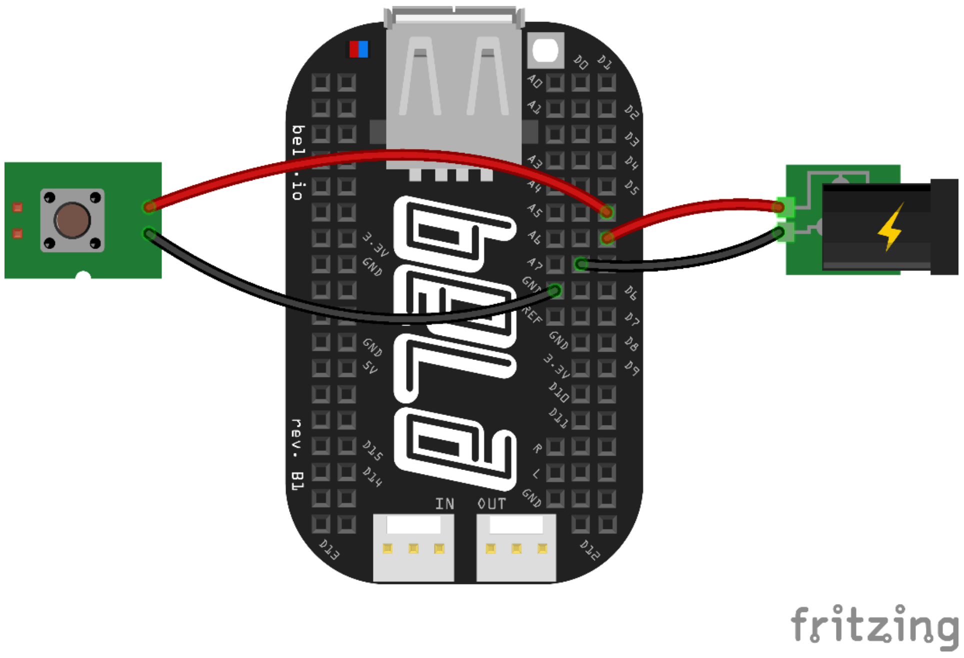


**Figure 24:** External Power Supply and Power Button Connection.

### 4.6.2 Power Switch

When powered by a USB supply, the board automatically powers up. However, a power switch must be added when an external supply is used, which is set up by incorporating a push button between P2 Pin 12 (labelled PB) and one of the available ground pins (see Figure 24). Shorting PB to ground acts as the main power button for the PocketBeagle and is also suitable for powering off the board correctly, eliminating the risk of memory loss that could occur when simply cutting the power from the board. [67]

### 4.6.3 Issue

While the external power supply can be helpful in specific scenarios, it has its drawbacks due to a particular design restraint of the PocketBeagle. This limitation states that the USB host port can only be powered by the 5V from the micro-USB port and not from any other sources. Consequently, when using the external power supply, any devices connected to the USB host port of the Bela Mini, such as Wi-Fi dongles or MIDI keyboards, become unusable. [68]

## 4.7 Expression Pedal

To enhance the device’s level of control and expressiveness, an expression pedal input was integrated. An expression pedal (see Figure 25) is a tool that enables real-time control over multiple parameters of multi-effect units, stomp boxes, synthesizers, and MIDI controllers. It offers a hands-free method to dynamically manipulate different aspects of an effect, whether in live performances or recording sessions. [69]

**Figure 25:** M-Audio Expression Pedal. [70]

### 4.7.1 Working Principle

An expression pedal contains a potentiometer, with its shaft linked to the pedal's treadle. The shaft of the potentiometer is rotated by rocking the treadle back and forth. The potentiometer has three pins: Ground, Signal, and Power. The resistance on each side of the central Signal pin is adjusted by rotating the shaft of the potentiometer. This adjustment alters the voltage on the middle pin by changing its proximity to either Power or Ground (see Figure 26). The analog input of a Bela system can measure the voltage change on the Signal pin. [71]

**Figure 26:** Expression Pedal Working Principle. [72]

### 4.7.2 Polarity

To ensure proper functionality, it is essential that the expression pedal's polarity matches that of the connected device. Polarity refers to how the wiring is configured for both the expression pedal and the effects unit. Most expression pedals use the TRS (Tip, Ring, Sleeve) configuration (see Figure 27), although some may utilize the RTS polarity. Certain pedals feature a polarity switch that allows switching between TRS and RTS. [73]

Since most expression pedals support the TRS wiring configuration, this polarity was implemented for the extension cape. With TRS, the tip of the 6.3 mm stereo jack socket is connected to the 5V supply of the Bela board, the ring to the analog input, and the sleeve to the ground. If the tip and sleeve were flipped, it would result in inverted readings from the potentiometer. However, swapping the tip and ring, as in the case of RTS configuration, would result in a short circuit.

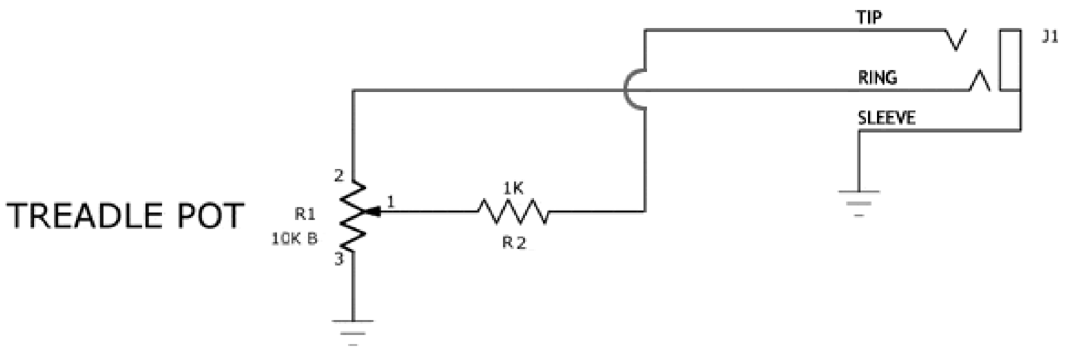
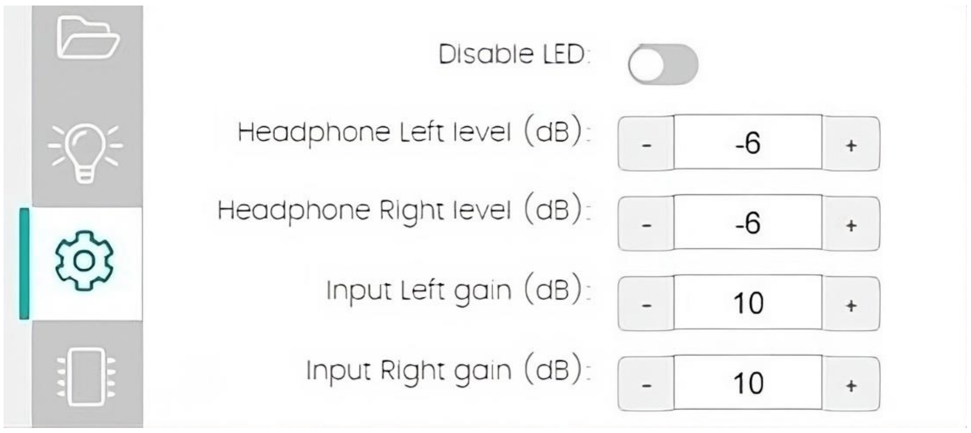


**Figure 27:** Expression Pedal Circuit. [74]

## 4.8 Audio I/O

Bela Mini offers two audio input and two audio output channels, accessible via 3-pin Molex-style connectors. Bela's audio input and output levels are determined by the adjustable settings of the audio codec, which can be configured through the Settings tab in the IDE (see Figure 28). This configuration option allows for connecting various musical instruments, microphones, and piezo pickups, providing a maximum gain of 59.5 db. When the gain is set to 0dB, the audio input range of Bela has a peak-to-peak amplitude of 1.8Vpp, while the headphone output 2.6Vpp. [75]



Disable LED:

Headphone Left level (dB): - -6 +

Headphone Right level (dB): - -6 +

Input Left gain (dB): - 10 +

Input Right gain (dB): - 10 +

**Figure 28:** IDE Audio Settings.

The Bela Mini Starter Kit includes two audio adapter cables connecting a 3.5mm stereo jack socket to the audio input and output. This connector type is highly suitable for accommodating input sources such as synthesizers or mobile phones and providing output to headphones or speakers. However, it is unsuitable for guitar pedal implementation, which usually incorporates 6.3mm mono jack sockets for audio input and output.

**Figure 29:** Bela Molex-style Audio Adapter Cables. [76]

To align with this standard, a 6.3mm mono jack socket was incorporated as an audio input to facilitate seamless connection with musical instruments, effect pedals, and other audio devices that can serve as audio sources. Two identical sockets were included for stereo audio output, enabling connection with guitar amps, audio interfaces, and comparable equipment. Additionally, a 3.5mm stereo jack socket was incorporated specifically for headphone usage, allowing the device to be operated without requiring external amplification.

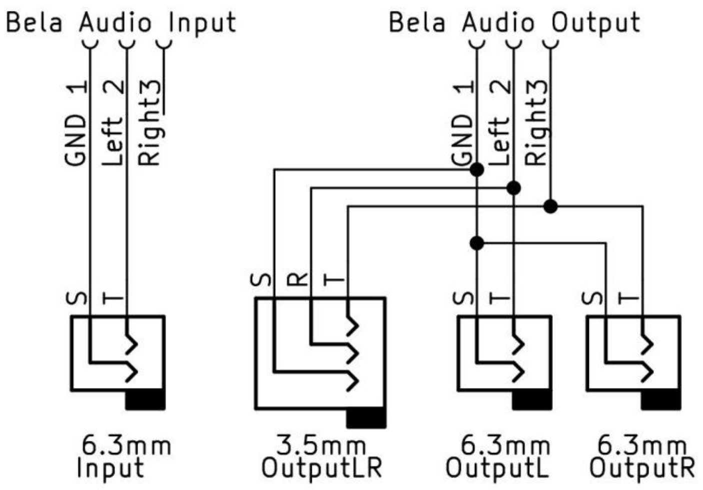


**Figure 30:** Audio Input and Output Circuit.

# 5. Extension Cape

## 5.1 Development Process

The increasing number of components and cable connections has resulted in the expansion of the hardware's size (see Figure 31), presenting a notable obstacle to the goal of designing a compact device. To overcome this issue, the creation of an extension cape has been proposed that combines all the necessary components into a single circuit.

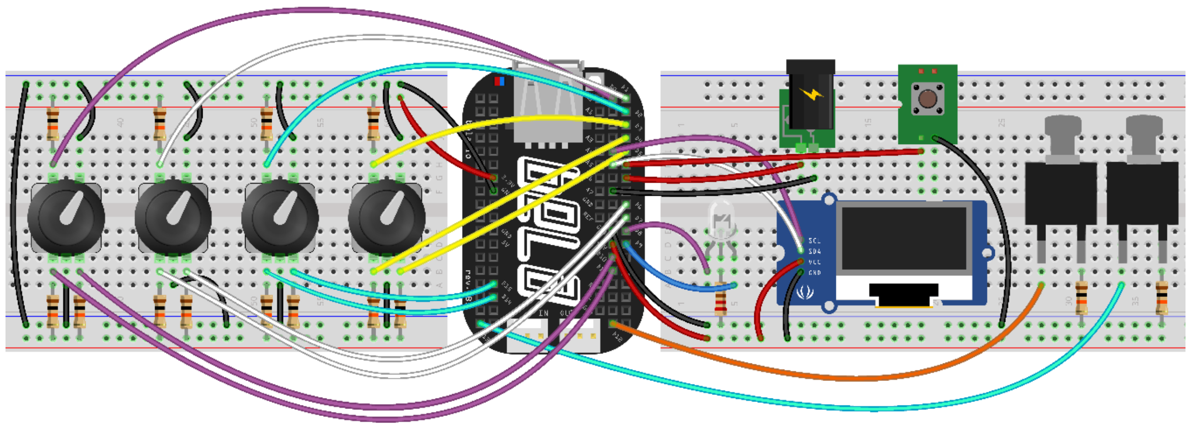


**Figure 31:** Breadboard Circuit including all Components.

As Bela is an open-source hardware and software platform, the schematics and printed circuit board (PCB) layouts for all Bela products are accessible to the public. [77] This accessibility proves beneficial when designing an extension cape for any Bela product, as it provides precise connector size and placement information. The development of Bela products utilized KiCad, a popular free software widely employed by the maker community for designing electronic hardware. Consequently, the proposed extension cape was also created using this software.

The development of the PCB began by setting up a schematic mirroring the breadboard layout, encompassing all the necessary components and their connections to the appropriate pins on the Bela board (see Figure 32). Subsequently, each element was assigned specific footprints, ensuring their accurate fit and placement on the circuit board. Once the schematic was completed, the PCB Editor was utilized to integrate all the components into a unified circuit and establish the connections according to the schematic design (see Figure 33).
(All files, including schematics and PCB layout are available in the GitHub repository.) [78]

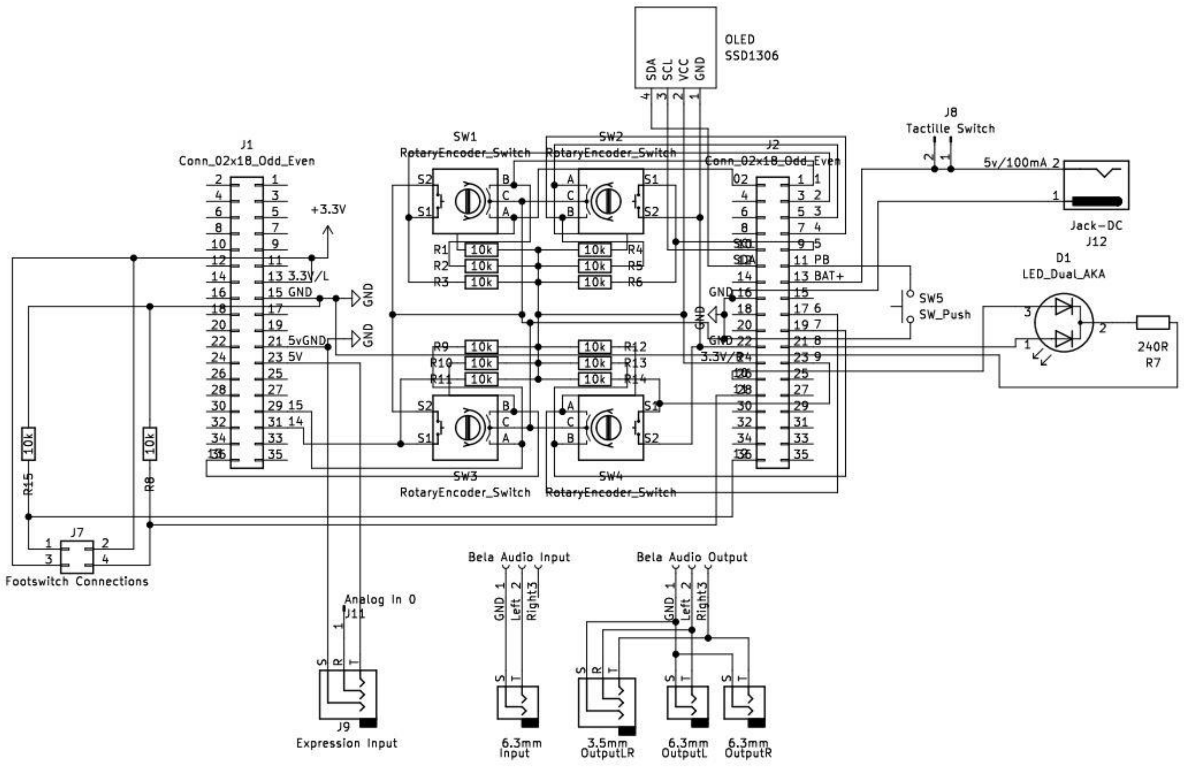


**Figure 32:** Extension Cape Schematic.

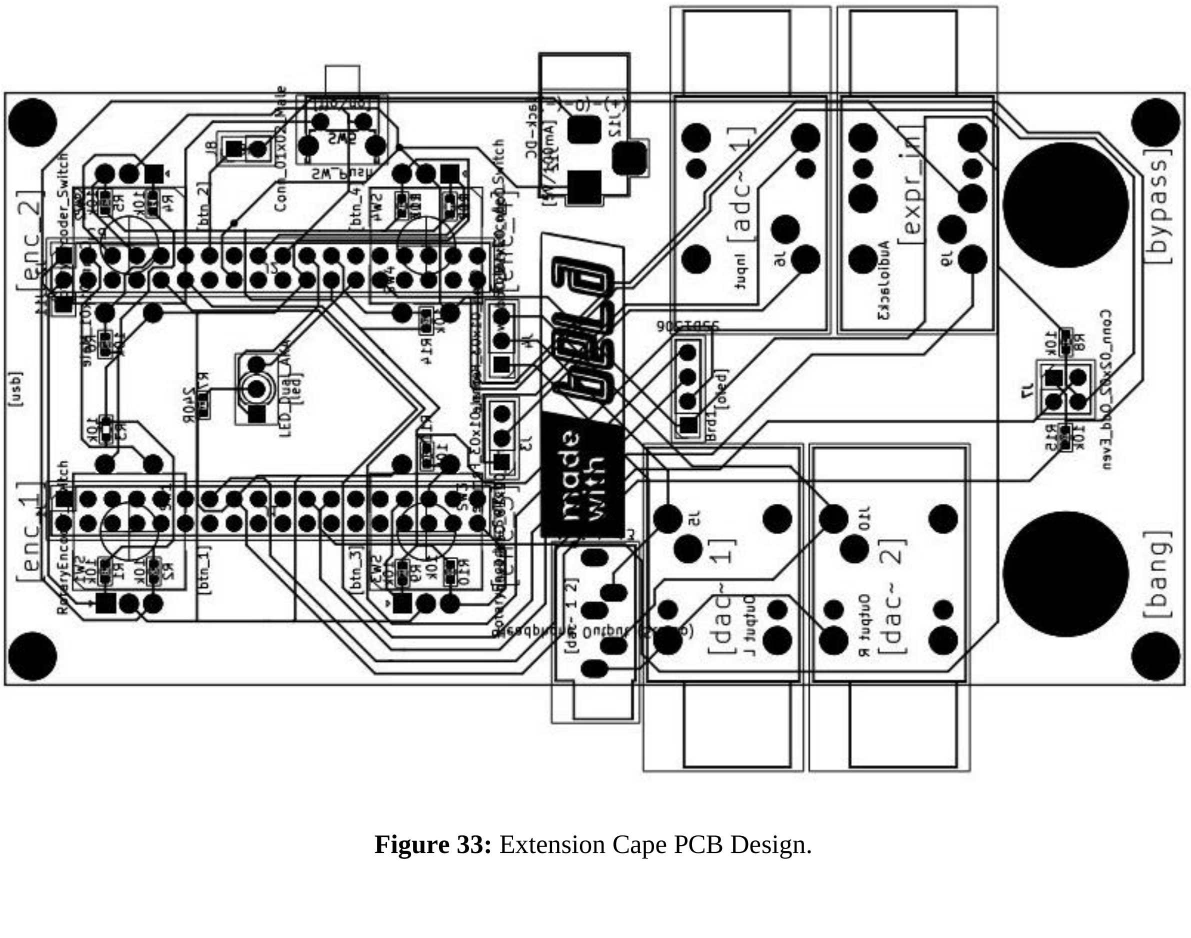


**Figure 33:** Extension Cape PCB Design.

### 5.1.1 Result

The final dimensions of the PCB measure 62x123mm and include footprints for four rotary encoders, each equipped with a switch, two momentary footswitches, a 6.3 mm mono jack socket for audio input, two identical sockets for stereo audio output, a 3.5 mm stereo jack socket for headphone output, a 6.3 mm stereo jack socket for expression pedal input, both a bicolor LED and OLED screen for visual feedback, a DC Barrel Jack enabling an external power supply and a tactile switch, acting as the main power button for the board.

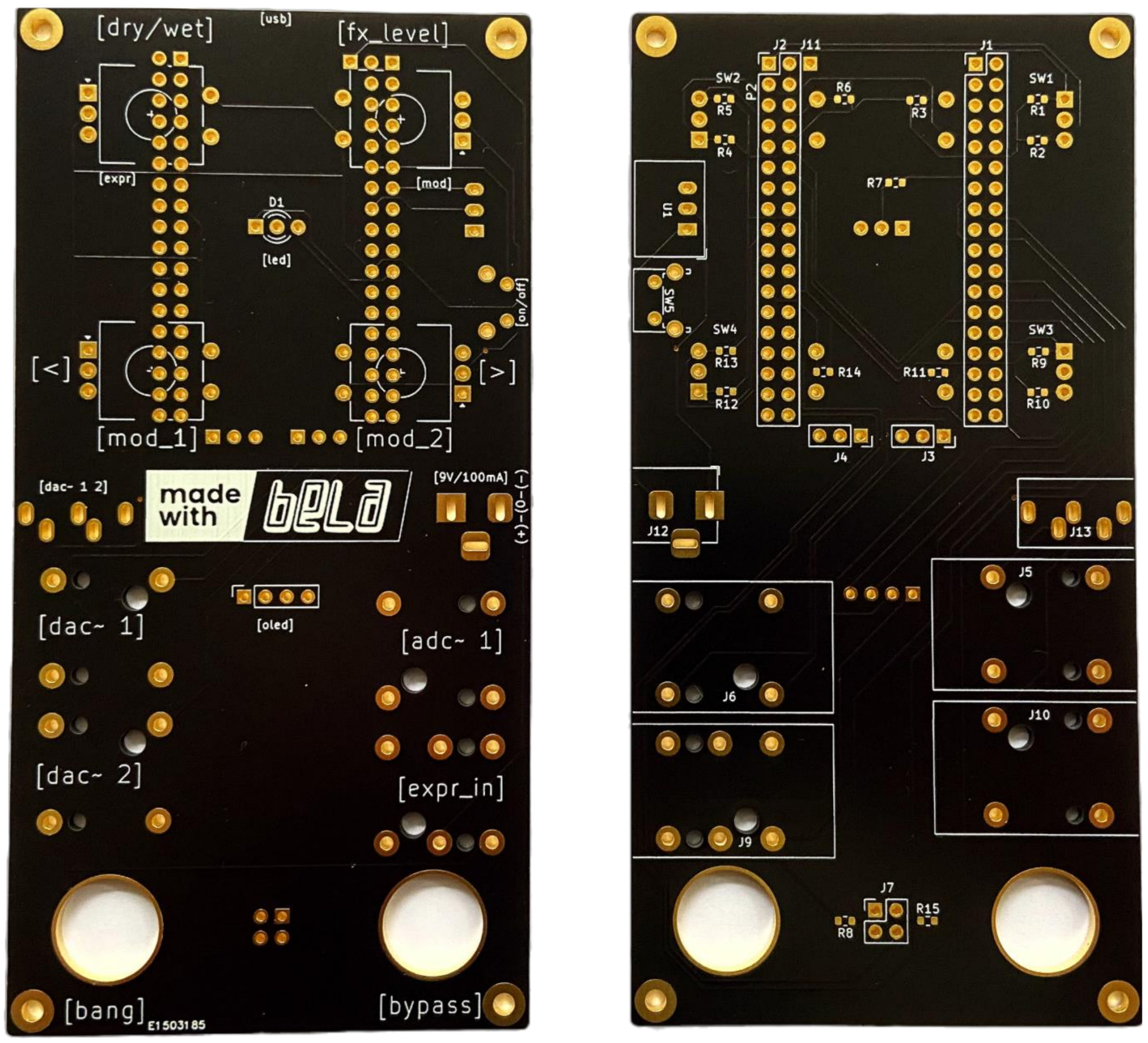


**Figure 34:** Printed Circuit Board.

## 5.2 Bill of Materials

The following list contains the necessary components required for assembling the extension cape. All these components fit seamlessly within their designated footprints on the PCB.

| | Component | Reference | Footprint | Quantity |
|---|---|---|---|---|
| 1 | SMD Chip Resistor 0603, 10 kOhm, 100 mW, 1% | R1, R2, R3, R4, R5, R6, R8, R9, R10, R11, R12, R13, R14, R15 | R_0603_1608Metric | 14 |
| 2 | SMD Chip Resistor 0603, 240 Ohm, 100 mW, 1% | R7 | R_0603_1608Metric | 1 |
| 3 | Tactile Switch, 6.6 x 7.4 mm, Horizontal PCB Mounting | SW | SW_Tactile_SPST_Angled_PTS645Vx58-2LFS | 1 |
| 4 | ALPS STEC12E08 – Incremental Rotary Encoder with Momentary Push Switch (6mm shaft) | [enc_1], [enc_2], [enc_3], [enc_4] | RotaryEncoder_Alps_EC11E-Switch_Vertical_H20mm | 4 |
| 6 | Neutrik NRJ4HF – 6.3mm Mono Jack Socket for Horizontal PCB Mounting | [adc~ 1], [dac~ 1], [dac~ 2] | Jack_6.35mm_Neutrik_NRJ4HF_Horizontal | 3 |
| 7 | Neutrik NRJ4HF 6.3mm Stereo Jack Socket for Horizontal PCB Mounting | [expr_in] | Jack_6.35mm_Neutrik_NRJ6HF_Horizontal | 1 |
| 8 | CUI Devices SJ1-3535NG – 3.5mm Stereo Jack Socket for Horizontal PCB Mounting | [dac~ 1 2] | Jack_3.5mm_CUI_SJ1-3535NG_Horizontal | 1 |
| 9 | Thomann PF4520 – Momentary Footswitch with 2 Solder Terminals | F_SW | Mounting Hole 12 mm | 2 |
| 10 | Bicolour 3mm (red/blue) LED with Common Cathode | [led] | LED_D3.0mm-3 | 1 |
| 11 | 0.96″ OLED Display 128×64 I2C SSD1315, Monochrome (White) | [oled] | PinSocket_1x04_P2.54mm_Vertical | 1 |
| 12 | Pin Header 2.54 mm, 2X18, straight | J1, J2 | PinHeader_2x18_P2.54mm_Vertical | 2 |
| 13 | Pin Socket 2.54 mm, 1X03, straight | J3, J4 | PinSocket_1x03_P2.54mm_Vertical | 2 |
| 14 | Pin Header 2.54 mm, 2X02, straight | J7 | PinHeader_2x02_P2.54mm_Vertical | 1 |
| 15 | Pin Header 2.54 mm, 1X02, straight | J8 | PinHeader_1x02_P2.54mm_Vertical | 1 |
| 16 | Pin Header 2.54 mm, 1X01, straight | J11 | PinHeader_1x01_P2.54mm_Vertical | 1 |
| 17 | CLIFF FCR681465 – DC Socket for 2.1mm & 2.5mm Plugs / Horizontal PCB Mounting | J12 | BarrelJack_Horizontal | 1 |

## 5.3 Assembly

Once all the necessary components were gathered, the following steps were followed to ensure proper installation and functionality of the board. (For component position, refer to Figure 35 or interactive BOM) [79]

**Step 1: Adding Resistors.**

The first step is to add the resistors to the circuit board. Resistors R1, R2, R3, R4, R5, R6, R8, R9, R10, R11, R12, R13, R14, R15 (10k) are soldered onto their designated areas, corresponding to the rotary encoders and switches. Additionally, resistor R7 (240R) is soldered to facilitate the LED operation.

**Step 2: Attaching Headers and Sockets**

Next, both 2x18 male headers (J1 and J2) are affixed to the circuit board, serving as the primary connection point with the Bela Mini, providing power supply as well as both digital and SPI communication. Furthermore, the 1x1 male header (J11) is connected to the A0 pin, establishing a connection with the expression pedal input. Additionally, 1x3 sockets (J3 and J4) are employed to establish the necessary connections for audio input and output.

**Step 3: Connecting Audio Inputs and Outputs**

To enable audio functionality, all three 6.3mm Mono Jack Sockets are attached to their designated place on the circuit board. Furthermore, the 6.3mm Stereo Jack Socket is connected to allow for an expression pedal input. Lastly, the 3.5mm Stereo Jack Socket is installed to accommodate the headphone output.

**Step 4: Adding Power Supply Components**

To set up the external power supply, both the DC Jack Barrel (J12) and Push Button (SW) are integrated into the circuit board.

**Step 5: Attaching Foot Switches**

Next, the 2x2 male header (J7) is affixed to the circuit board, allowing for the connection of both momentary foot switches. These foot switches are securely fastened in their designated 12mm mounting holes, and their legs are connected to the respective header pins.

**Step 6: Attaching Rotary Encoders**

The rotary encoders (Enc_1, Enc_2, Enc_3, Enc_4) are attached to the circuit board, ensuring that their bottom side does not come into contact with any underlying pins. This precaution is taken to prevent interference or malfunction.

**Step 7: Adding LED**

The bicolor LED is added, and its legs are cut to the appropriate length. The red anode is connected to digital pin 8, while the blue anode is attached to digital pin 10.

**Step 8: Attaching OLED**

Finally, the OLED display is connected to its designated footprint, finalizing the assembly process.

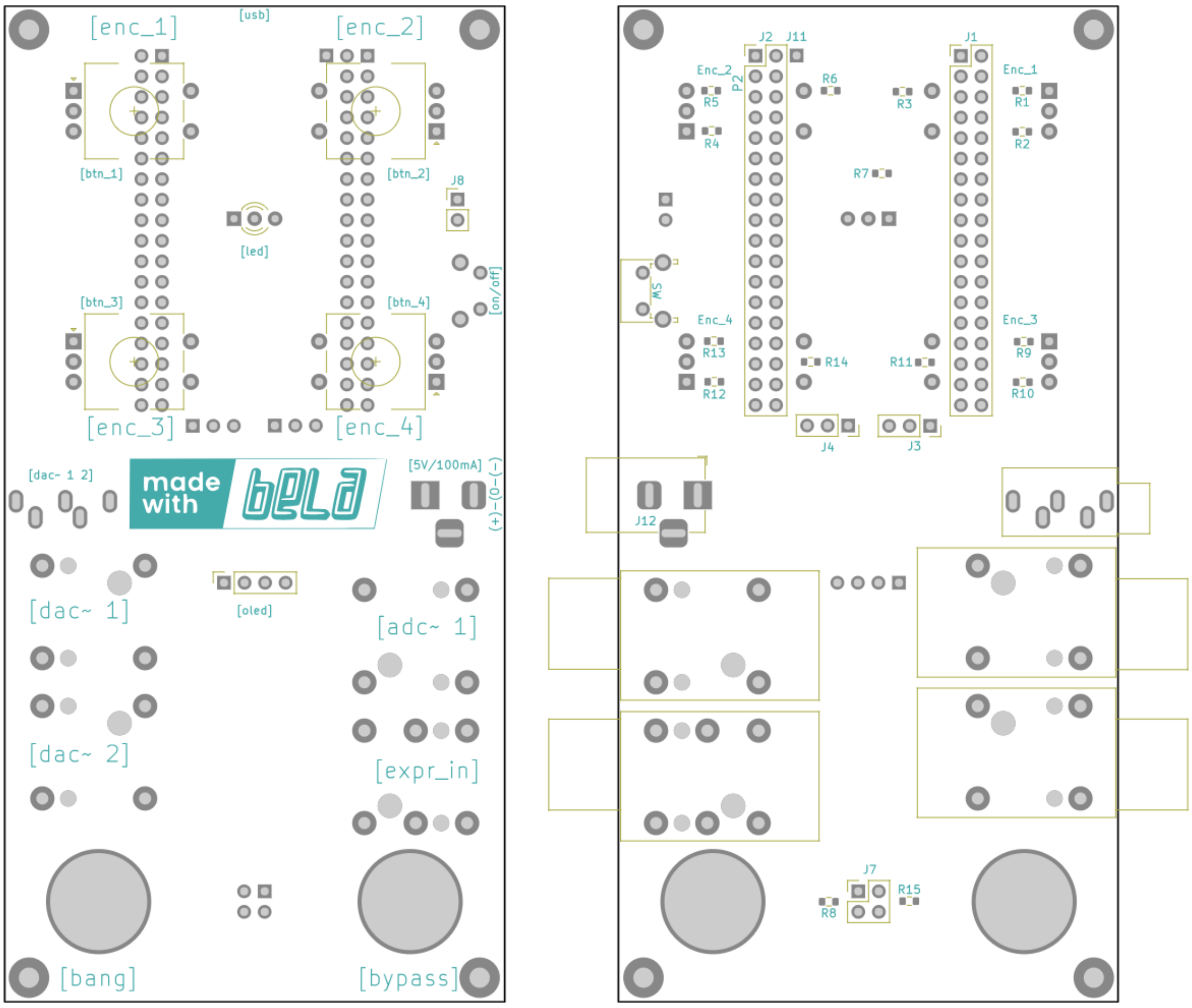


**Figure 35:** PCB Component Positions.

## 5.4 Enclosure

As shown in Figure 36, the assembled extension cape can be easily connected to the Bela Mini and used as is.

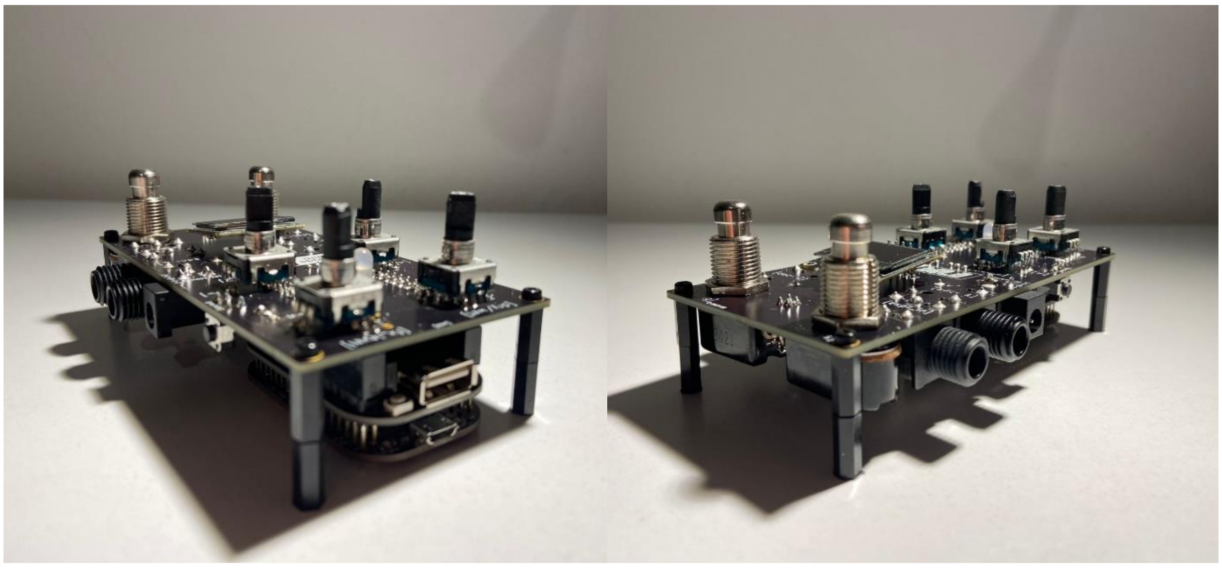


**Figure 36:** Extension Cape connected to Bela Mini.

However, for enhanced stability and compactness, an enclosure has been designed (see Figure 37), which can be manufactured using a laser cutter and any 3mm thick material suitable for this purpose.

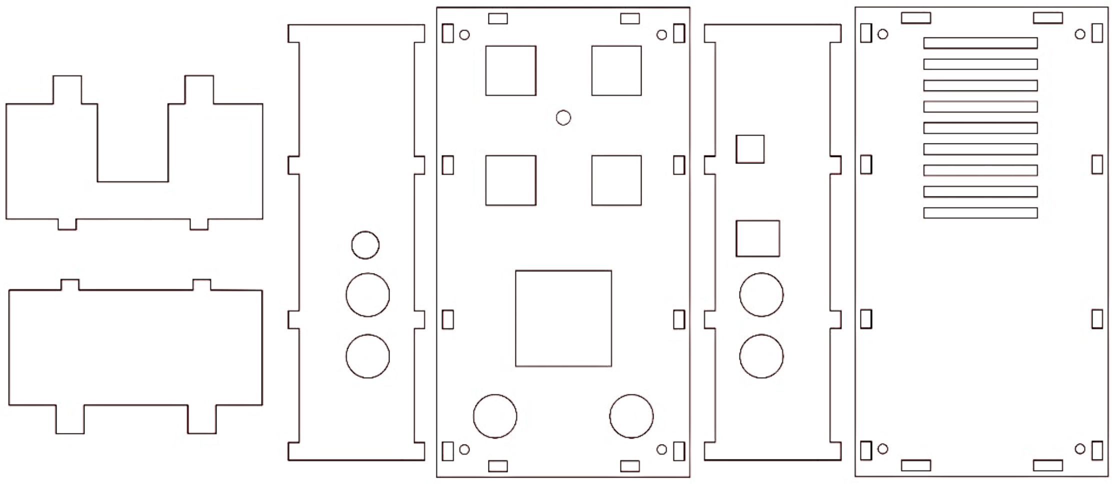


**Figure 37:** Enclosure Drawing. [80]

The cape features four mounting holes, one at each corner, intended for attaching the enclosure to the cape. To assemble the case, additional standoffs and screws are required to hold everything together. Four M2.5 5mm Female to Male Standoffs are mounted to the top of the PCB, along with four M2.5 25mm Female to Female Standoffs mounted to the bottom. Finally, eight M2.5 6mm Screws are added to attach the case’s top and bottom sides to the standoffs.

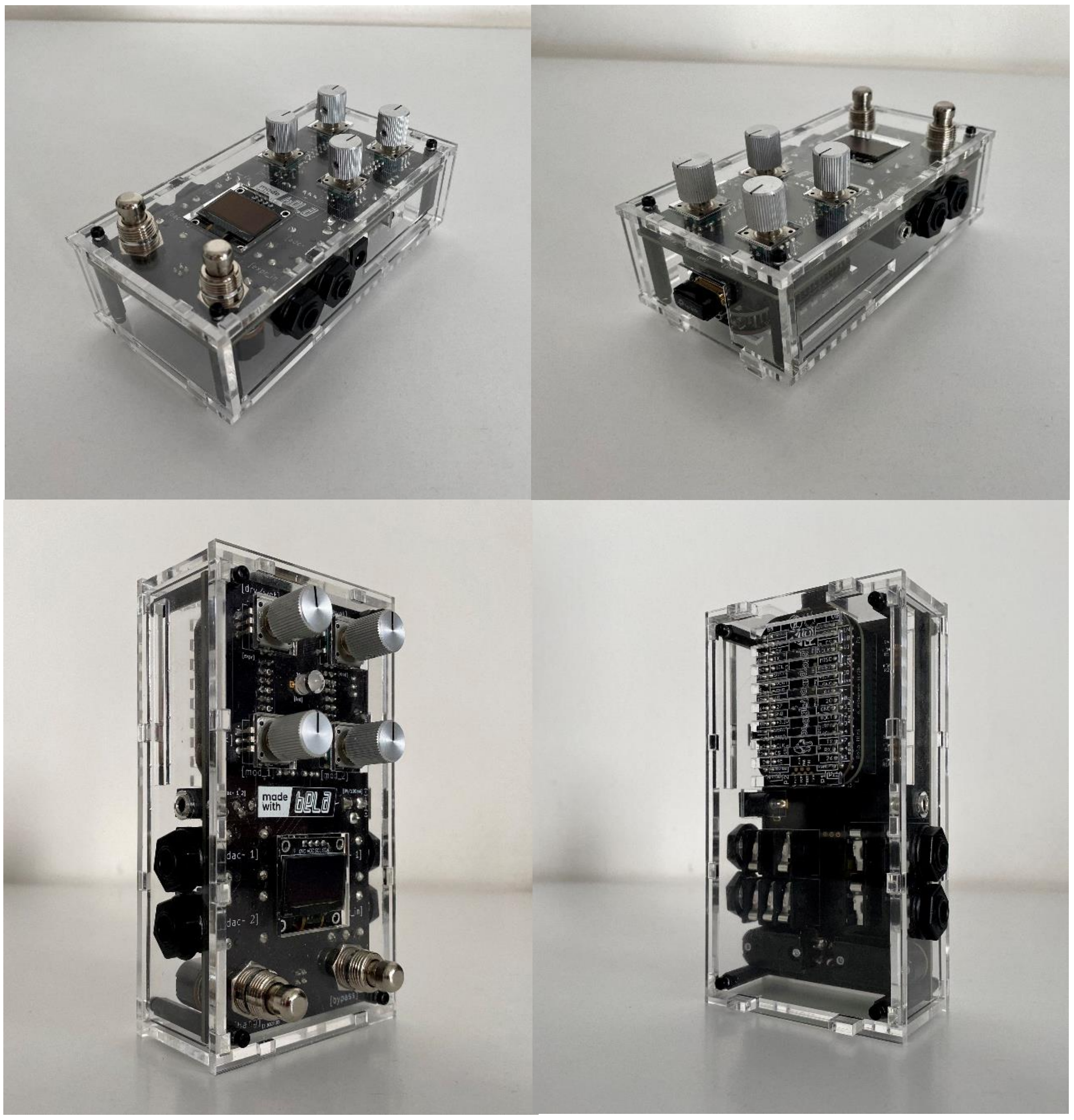


**Figure 38:** Extension Cape Enclosure.

# 6. Software Framework

A further project goal is that the software is easily customizable, allowing users with different coding backgrounds to develop and apply their unique audio effects on this specialized hardware. To make this possible, an example project called “Delay_Chain” [81] was created using the Pure Data platform. This comprehensive software framework provides the essential code for operating the hardware and integrates various obsolete audio effects like Hammond’s scanner vibrato, tape delay, and Schroeder reverberator. It aims to offer users a versatile and user-friendly platform for exploring and implementing their creativity in audio effect customization on this specialized audio hardware.

## 6.1 Pure Data

Pure Data (Pd), created by Miller S. Puckette, is a graphical computer music language that facilitates real-time applications. Utilizing a graph-oriented approach, Pd enables visual programming through data flow diagrams, offering rapid prototyping and the construction of dynamic patches and scripts. Moreover, it supports network communication, enabling real-time interaction and patch exchange among multiple Pd instances, even across the internet. [82]

Being open-source and freely available, Pd can be accessed on various platforms, empowering composers and musicians to explore, improvise, and create innovative forms of interactive music. The utilization of a block-based coding methodology has made it widely popular among beginners, as it provides an intuitive learning environment. As a result, Pd became an indispensable tool in the realm of computer music and the chosen software environment for the development of this project. [83]

## 6.2 Pd on Bela

Bela utilizes the libPd library, enabling the execution of Pd’s core DSP functionality on the board. It operates in a “headless” mode, implying that only the essential audio components of Pd run on Bela without the graphical interface. This means that the IDE does not function as a Pure Data patching environment, so object creation or editing is not possible within the IDE. Instead, patches are created using Pure Data on a laptop and then uploaded to Bela. The main patch of a project must always be named “_main.pd”; otherwise, it will not be recognized by the platform. Any additional sub-patches and abstractions can have any name and will be identified as long as they are connected to the main patch. [84]

### 6.2.2 Limitation

One significant drawback of the Bela system is its support limited exclusively to Pd Vanilla objects. [85] This can lead to restrictions on code complexity or necessitate the reconstruction of Pd extended objects using the available ones. While it is feasible to compile specific Pd extensions for Bela, achieving this requires advanced coding skills and may lead to buggy results or, in some cases, no support at all.

## 6.3 Main Patch

In the context of the Delay_Cain example project, the parent patch (see Figure 39) takes charge of managing all audio processes. It contains all three effects chained in interchangeable order and connected to the audio input and outputs, respectively. All other sub-patches and abstractions are associated with this main patch through the [interface] sub-patch, allowing Bela to identify them as integral components of the project.

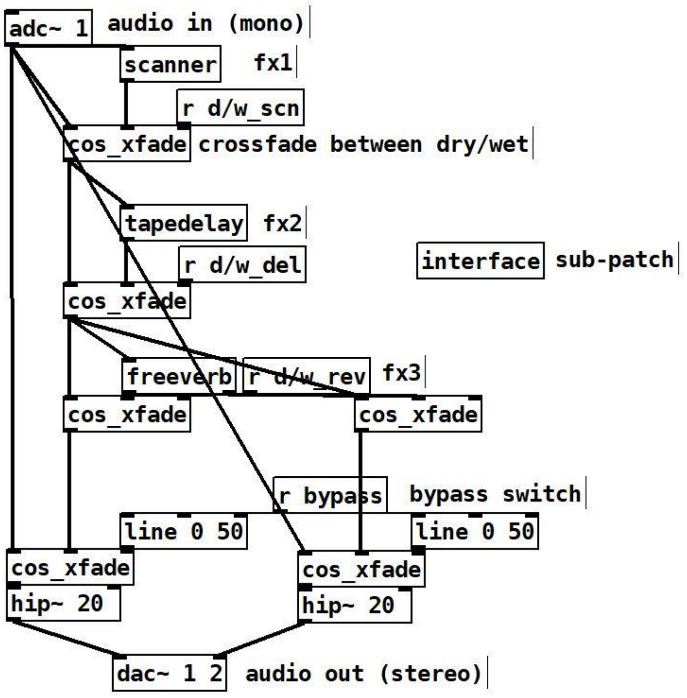


**Figure 39:** Main Patch.

### 6.3.1 Audio I/O

The [adc~] and [dac~] objects are used to handle audio inputs and outputs in Pd. To access the stereo audio input on Bela, the [adc~ 1 2] is utilized, while sending audio to the stereo output involves connecting to [dac~ 1 2]. [86] However, only [adc~ 1] can be employed when using the extension cape because the audio pin corresponding to [adc~ 2] is not connected to any physical input. On the other hand, both [dac~ 1] and [dac~ 2] objects are accessible since these are connected to the headphone output and the two mono 6.3mm sockets, respectively.

### 6.3.2 Crossfade

A crossfade mechanism has been integrated to allow blending between the dry input signal and the output of all three effects. Instead of using a linear reverse panner that results in a 3dB amplitude reduction at its center position, a sine-cosine law crossfader has been employed. This alternative crossfader minimizes amplitude reduction and ensures a smoother transition between the signals. [87]

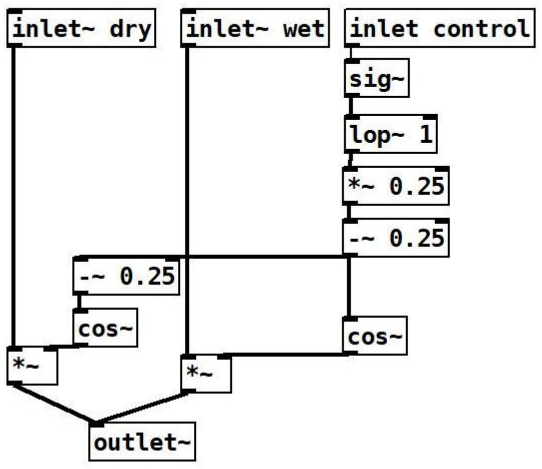


**Figure 40:** Crossfade Abstraction.

### 6.3.3 Bypass

The main patch also incorporates the bypass function, which is achieved solely through software within the digital processor. This form of digital bypassing is unique to DSP-based guitar effects. It shares similarities with true bypass, where the guitar signal is sent directly to the amplifier without interference, impedance changes, or buffering effects. However, in this case, the analog guitar signal is still converted to digital and then back to analog but without additional signal modifications. The sound quality of this bypass method relies heavily on the digital converter’s quality, band-limiting, and anti-aliasing filters. [88]

## 6.4 Addressing the Physical Interface

The extension cape features four rotary encoders, each equipped with a switch, as well as two footswitches, all of which can be assigned based on preferences. All the digital data received from these components is obtained and addressed within the [interface] sub-patch (see Figure 41).

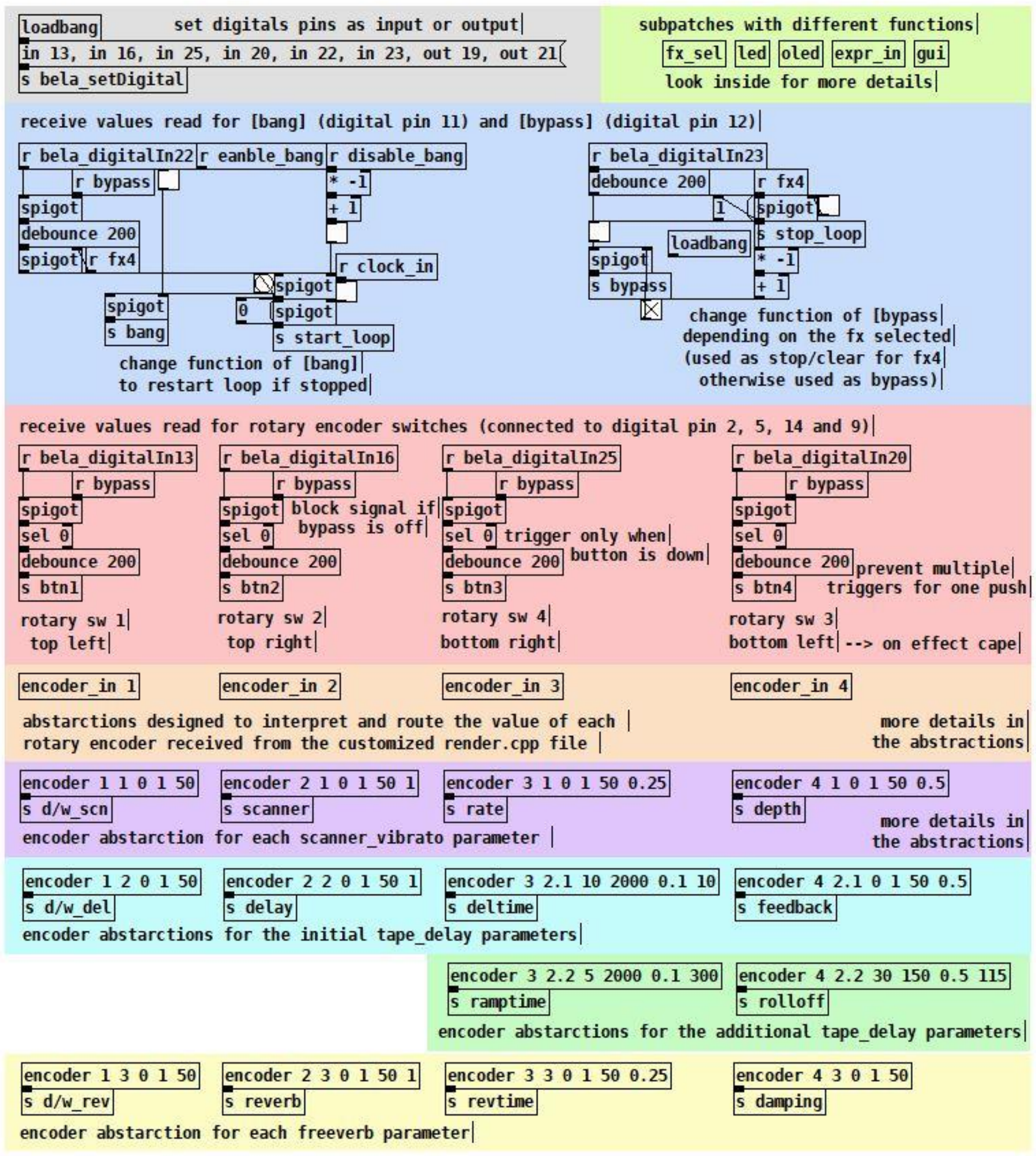


**Figure 41:** Interface Sub-patch.

### 6.4.1 Digital Input

Bela’s digital pins have two distinct states, commonly referred to as “high” or “low.” Each digital pin can be configured as either an input or an output, and their mode is initialized in the patch by sending messages to the [bela_setDigital] object. It’s important to note that Bela’s digital pins 0-16 are identified as pins 11-26 within the Pure Data environment. To read the value of a particular digital input, the [r bela_digitalIn$] object is used, where “$” represents the pin number. This digital input can only have two possible values: 1 or 0. The transition between these states depends on the input received at the pin. When the button connected to the digital input is pressed, it closes the circuit, and the input will read 3.3V, “which is interpreted as one on Bela. Conversely, when the button opens, the input pin reads 0V, representing 0 on Bela. [89]

### 6.4.2 Button Functions

In the case of the Delay_Chain project, the bottom left and right rotary encoder switches are utilized to alternate between effects in both forward and backward directions. The top left rotary encoder switch is designated to enable/disable the expression mode, while the top right is employed to access alternative parameters for a specific effect. Moreover, the primary function of the right footswitch is to serve as the bypass switch, but it can be assigned for alternative uses for particular effects. The right footswitch is labelled [bang], which embraces various tasks for different effects, such as tap tempo, record, play, etc.

### 6.4.3 Rotary Encoder

Interfacing with sensors such as rotary encoders require precise control of digital pins and accurate timing, which is often easier to achieve with C++ than with Pure Data. To accomplish this, a custom render.cpp file is added to the project, allowing signals and messages to be exchanged between the Pure Data project and the C++ code. By implementing the interfacing protocol in C++, the Pure Data patch can act as a receiver for the readings collected by the render.cpp file. [90]

To obtain rotary encoder values from the render.cpp file and transmit them to PD, two C++ codes were merged taken from the IDE's examples section. The primary code is a modified version of render.cpp that could interpret readings from an Ultrasonic Ranging Module and forward them to PD. The second code is an example that could retrieve and record rotary encoder values while also displaying them on the console. Upon correctly incorporating the encoder code into the render.cpp file, it became possible to transmit the data of four rotary encoders and receive their value with a receive object in PD.

### 6.4.4 Custom Render File Modification

The following code contains all the modifications added to render.cpp file.
(The full code is available in the GitHub repository.) [91]

```
// Custom libpd render.cpp to read and send four rotary encoder values to a
Pure Data patch.

//Encoder Modifications

Encoder gEncoder0;
//Scope gScope;
// Bela digital input channels connected to the encoder and button
unsigned int kEncChA = 0;
unsigned int kEncChB = 1;
// unsigned int kEncChBtn0 = 2;

Encoder gEncoder1;
unsigned int kEncChC = 3;
unsigned int kEncChD = 4;
// unsigned int kEncChBtn1 = 5;

Encoder gEncoder2;
unsigned int kEncChE = 15;
unsigned int kEncChF = 13;
// unsigned int kEncChBtn2 = 14;

Encoder gEncoder3;
unsigned int kEncChG = 6;
unsigned int kEncChH = 7;
// unsigned int kEncChBtn3 = 9;

// adjust the values below based on your encoder and wiring
unsigned int kDebouncingSamples = 15;
```

```
Encoder::Polarity polarity = Encoder::ANY; // could be ANY, ACTIVE_LOW,
ACTIVE_HIGH

     //--------------------------------------------------

    //gScope.setup(3, context->audioSampleRate);
        gEncoder0.setup(kDebouncingSamples, polarity);
        // Set the digital pins to inputs
        pinMode(context, 0, kEncChA, INPUT);
        pinMode(context, 0, kEncChB, INPUT);
        // pinMode(context, 0, kEncChBtn0, INPUT);

        gEncoder1.setup(kDebouncingSamples, polarity);
        pinMode(context, 0, kEncChC, INPUT);
        pinMode(context, 0, kEncChD, INPUT);
        // pinMode(context, 0, kEncChBtn1, INPUT);

        gEncoder2.setup(kDebouncingSamples, polarity);
        pinMode(context, 0, kEncChE, INPUT);
        pinMode(context, 0, kEncChF, INPUT);
        // pinMode(context, 0, kEncChBtn2, INPUT);

        gEncoder3.setup(kDebouncingSamples, polarity);
        pinMode(context, 0, kEncChG, INPUT);
        pinMode(context, 0, kEncChH, INPUT);
        // pinMode(context, 0, kEncChBtn3, INPUT);

        //--------------------------------------------------
{
        for(unsigned int n=0; n<context->digitalFrames; n++){
                bool a = digitalRead(context, n, kEncChA);
                bool b = digitalRead(context, n, kEncChB);

                // bool button0 = digitalRead(context, n, kEncChBtn0);
                Encoder::Rotation ret = gEncoder0.process(a, b);

                if(Encoder::NONE != ret)
              // {
                 //       rt_printf("%s : %3d\n", Encoder::CCW == ret ? "ccw"
: "cw ", gEncoder.get());
                    libpd_float("encoder1", gEncoder0.get()); // send to Pd

        }
}

    {
        for(unsigned int n=0; n<context->digitalFrames; n++){
                bool c = digitalRead(context, n, kEncChC);
                bool d = digitalRead(context, n, kEncChD);

                // bool button1 = digitalRead(context, n, kEncChBtn1);
                Encoder::Rotation ret = gEncoder1.process(c, d);

                if(Encoder::NONE != ret)
              // {
                 //       rt_printf("%s : %3d\n", Encoder::CCW == ret ? "ccw"
: "cw ", gEncoder.get());
                    libpd_float("encoder2", gEncoder1.get()); // send to Pd

        }
}

    {
        for(unsigned int n=0; n<context->digitalFrames; n++){
                bool e = digitalRead(context, n, kEncChE);
                bool f = digitalRead(context, n, kEncChF);

                // bool button2 = digitalRead(context, n, kEncChBtn2);
                Encoder::Rotation ret = gEncoder2.process(e, f);

                if(Encoder::NONE != ret)
              // {
                 //       rt_printf("%s : %3d\n", Encoder::CCW == ret ? "ccw"
: "cw ", gEncoder.get());
                    libpd_float("encoder3", gEncoder2.get()); // send to Pd

        }
}
        {
        for(unsigned int n=0; n<context->digitalFrames; n++){
                bool g = digitalRead(context, n, kEncChG);
                bool h = digitalRead(context, n, kEncChH);

                // bool button3 = digitalRead(context, n, kEncChBtn3);
                Encoder::Rotation ret = gEncoder3.process(g, h);

                if(Encoder::NONE != ret)
              // {
                 //       rt_printf("%s : %3d\n", Encoder::CCW == ret ? "ccw"
: "cw ", gEncoder.get());
                    libpd_float("encoder4", gEncoder3.get()); // send to Pd

        }
}

}
```

### 6.4.5 Encoder Receiver

To account for the possibility of infinite increments or decrements of the encoder values received from render.cpp, it was necessary to impose limits on their range within Pure Data. To achieve this, a system was developed that detects if the value is increased or decreased and outputs -1 or 1 for each change, respectively. Additionally, five send objects were integrated for both negative and positive values, enabling one encoder input to be routed and control multiple parameters if needed. To further refine the system, an if statement was introduced to compare the amount of the change in the value received from the render.cpp file to a threshold of 3. This evaluation is used to provide a longer debounce time for slow rotation of the encoder and no debouncing when it is rotated fast.

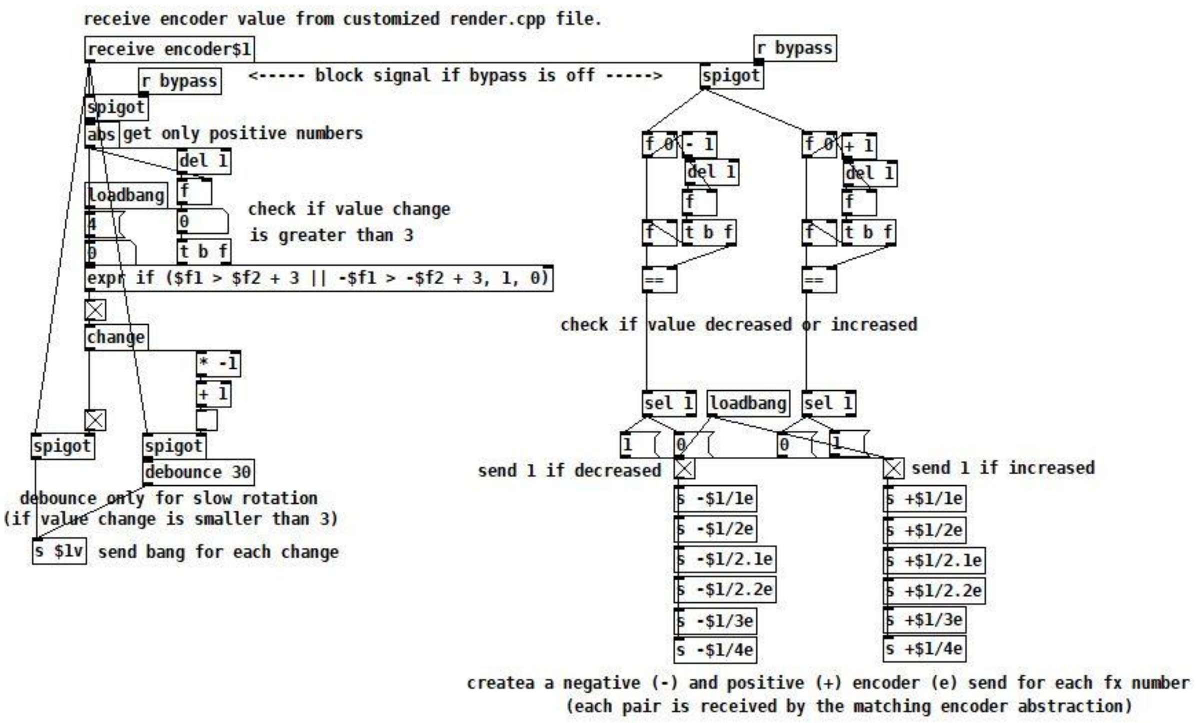
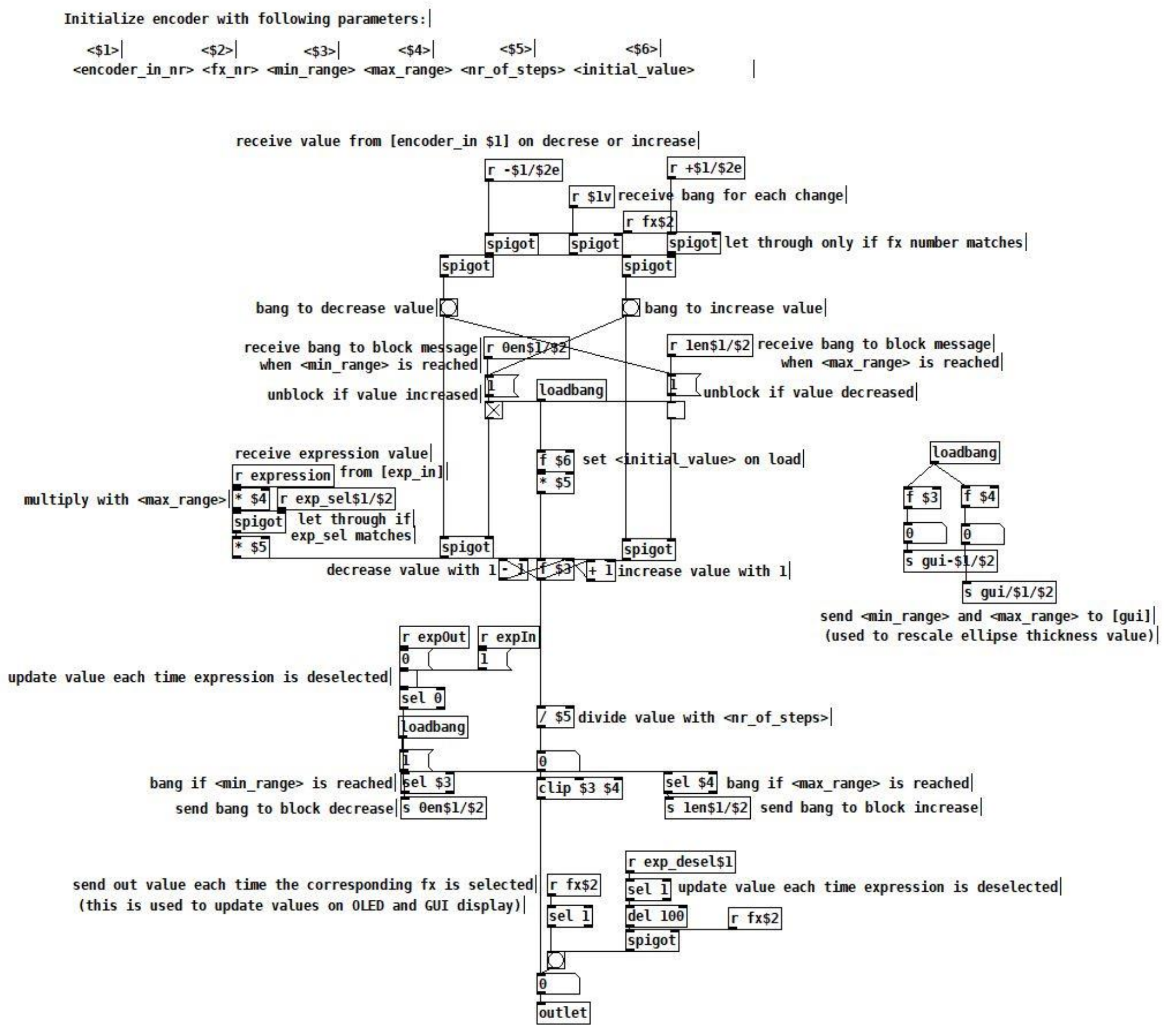

receive encoder value from customized render.cpp file.
receive encoder$1
r bypass
r bypass
<----- block signal if bypass is off ----->
spigot
spigot
abs
get only positive numbers
del 1
f
loadbang
0
t b f
4
0
check if value change
is greater than 3
expr if ($f1 > $f2 + 3 || -$f1 > -$f2 + 3, 1, 0)
change
* -1
+ 1
spigot
spigot
debounce 30
debounce only for slow rotation
(if value change is smaller than 3)
s $1v
send bang for each change
f 0
- 1
del 1
f
f
t b f
==
f 0
+ 1
del 1
f
f
t b f
==
check if value decreased or increased
sel 1
loadbang
sel 1
1
0
0
1
send 1 if decreased
send 1 if increased
s -$1/1e
s -$1/2e
s -$1/2.1e
s -$1/2.2e
s -$1/3e
s -$1/4e
s +$1/1e
s +$1/2e
s +$1/2.1e
s +$1/2.2e
s +$1/3e
s +$1/4e
createa a negative (-) and positive (+) encoder (e) send for each fx number
(each pair is received by the matching encoder abstraction)

**Figure 42:** Encoder Receiver Sub-Patch.

## 6.4.6 Encoder Abstraction

In order to control multiple parameters using one encoder input, an abstraction was set up to address the values received from the [encoder_in] sub-patch. Abstractions are reusable sub-patches that allow for the creation of multiple instances, each with distinct behaviors, by setting different creation arguments in the object box. [92] This particular abstraction is initiated with the following parameters: <encoder_in_nr> <fx_nr> <min_range> <max_range> <nr_of_steps> <initial_value>. This means that [encoder 2 3 0 100 50 25] translates into an abstraction that takes input from [encoder_in 2], assigns it to effect number 3, spans a range from 0 to 100 in 50 steps, and loads at an initial value of 25.



Initialize encoder with following parameters:
<$1> <$2> <$3> <$4> <$5> <$6>
<encoder_in_nr> <fx_nr> <min_range> <max_range> <nr_of_steps> <initial_value>
receive value from [encoder_in $1] on decrese or increase
r -$1/$2e
r +$1/$2e
r $1v
receive bang for each change
r fx$2
spigot
let through only if fx number matches
bang to decrease value
bang to increase value
receive bang to block message when <min_range> is reached
r 0en$1/$2
r len$1/$2
receive bang to block message when <max_range> is reached
unblock if value increased
loadbang
unblock if value decreased
receive expression value
r expression
from [exp_in]
multiply with <max_range>
* $4
r exp_sel$1/$2
let through if exp_sel matches
* $5
f $6
set <initial_value> on load
loadbang
f $3
f $4
s gui-$1/$2
s gui/$1/$2
decrease value with 1
- 1
f $3
+ 1
increase value with 1
send <min_range> and <max_range> to [gui] (used to rescale ellipse thickness value)
r expOut
r expIn
update value each time expression is deselected
sel 0
/ $5
divide value with <nr_of_steps>
bang if <min_range> is reached
sel $3
clip $3 $4
sel $4
bang if <max_range> is reached
send bang to block decrease
s 0en$1/$2
s len$1/$2
send bang to block increase
r exp_desel$1
send out value each time the corresponding fx is selected (this is used to update values on OLED and GUI display)
r fx$2
sel 1
update value each time expression is deselected
del 100
spigot
outlet

**Figure 43:** Encoder Abstraction.

## 6.5 Expression Pedal Input

The expression pedal is equipped with a potentiometer, and its signal pin connects to Analog In 0 on the board. Bela's analog inputs are treated like audio signals, and the [adc~] object is used to receive this data in Pd. The audio inputs on Bela start at [adc~ 3], as the first two channels are reserved for audio inputs. These inputs are sampled at an audio sampling rate and treated as audio signals. [93]

Since the potentiometer's analog reading comes into Pd at an audio rate, it must be converted to a control rate signal to remap its value. To achieve this, an [env~] object is employed, which acts as an envelope follower and outputs the Root Mean Square (RMS) amplitude in decibels, with one normalized to 100 dB. Next, the output is sent to a [dbtorms~] object, which converts the decibel value to a linear amplitude. This results in a control value of 0 when the potentiometer is turned down and a reading close to one when turned up completely.

To control the desired parameter using the expression pedal, a selection system is implemented that assigns the incoming value based on the user's selection using the rotary switch knobs. The top left rotary encoder switch is designated to turn the expression mode on/off. Pressing this switch selects and links the assigned parameter to the expression pedal for control. By pressing the buttons of the other encoders, users can choose the parameter they want to control. To exit the expression mode, the top left button must be pressed again.

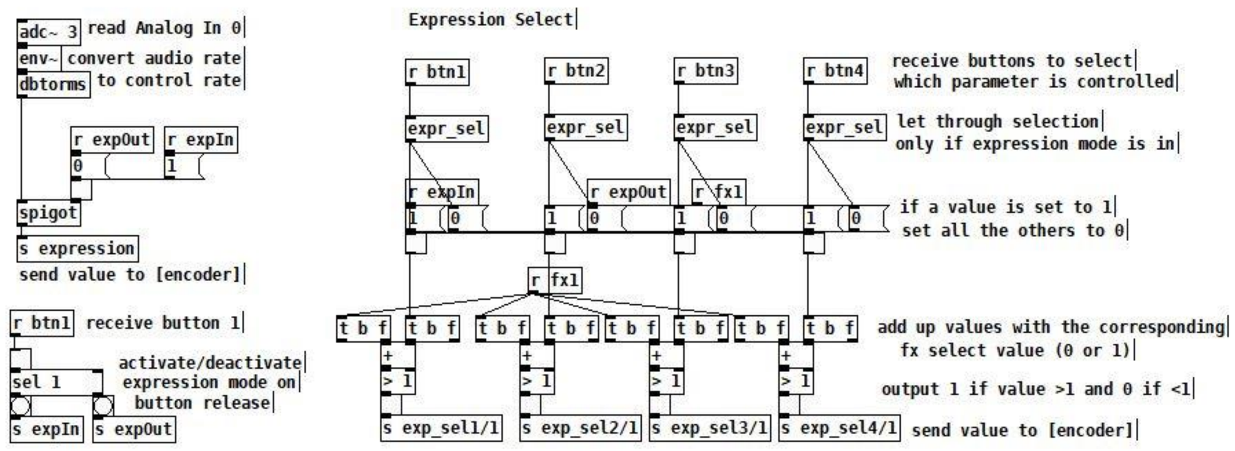


**Figure 44:** Expression Sub-patch Section.

## 6.6 Operating the LED

To operate the LED, both digital pins connected to its anodes have to be set up as outputs by sending the appropriate messages to the [bela_setDigital] object. In the case of the extension cape, the red anode of the LED is connected to digital pin 8, while the blue anode is attached to pin 10. These correspond to [bela_digitalOut19] and [bela_digitalOut21] in Pd. Sending a value of 1 to these objects turns the state of the pins "high," which means that they output 3.3v. On the other hand, when they receive a value of 0, they enter a "low" state, outputting no voltage at all. [94]

The Delay_Chain project sets the LED's initial color to blue, and its on/off state is managed by the bypass switch. Upon entering expression mode, the LED turns red, while accessing alternative parameters changes its color to purple, indicating the activation of this feature. Additionally, the LED flashes purple for 200 ms to signify the transition between effects. Purple is achieved by alternating between red and blue every ten milliseconds.

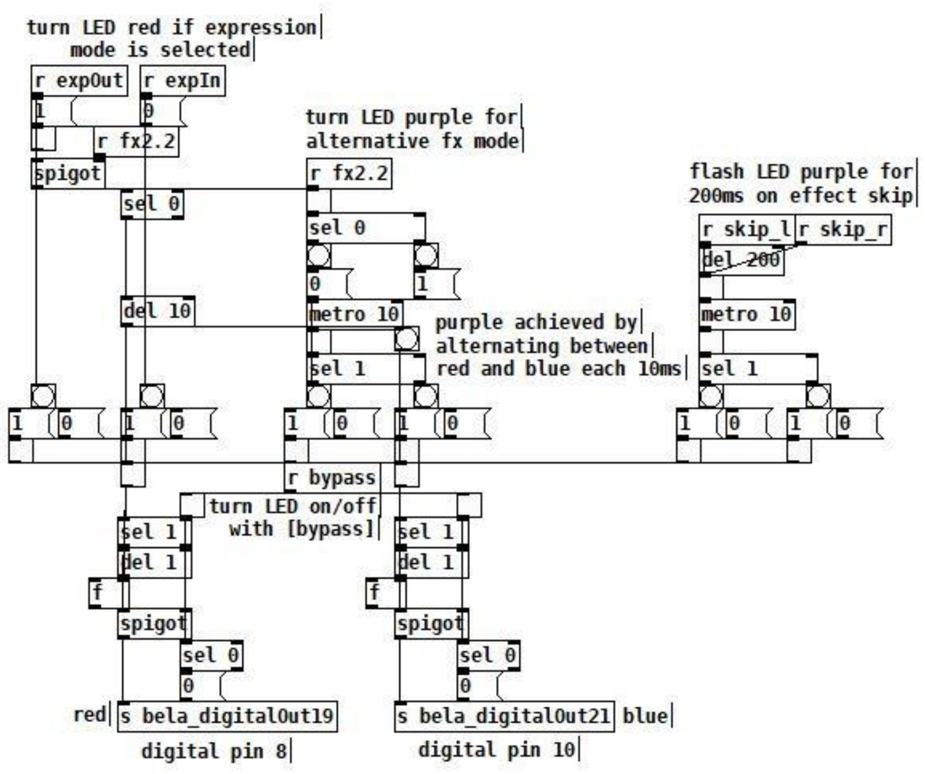

turn LED red if expression mode is selected
r expOut
r expIn
r fx2.2
spigot
sel 0
del 10
turn LED purple for alternative fx mode
r fx2.2
sel 0
metro 10
sel 1
purple achieved by alternating between red and blue each 10ms
flash LED purple for 200ms on effect skip
r skip_l
r skip_r
del 200
metro 10
sel 1
r bypass
turn LED on/off with [bypass]
sel 1
del 1
f
spigot
sel 0
red
s bela_digitalOut19
digital pin 8
s bela_digitalOut21
blue
digital pin 10

**Figure 45:** LED Sub-Patch.

## 6.7 Driving the OLED

### 6.7.1 OSC Bridge

The OLED screen is connected to the I2C2 pins of the Bela board. Directly accessing these pins from Pure Data is not possible, but a convenient solution is available using an OSC to OLED (O2O) bridge for Linux. This framework utilizes the u8g2 Arduino library and Bela's OscReceiver and UdpServer classes, facilitating the transmission of OSC messages to drive the OLED screen connected to the I2C2 pins. [95]

### 6.7.2 OSC in Pd

OSC, short for Open Sound Control, is a communication protocol for transmitting information between computers, synthesizers, and various multimedia devices. Each OSC message consists of a title followed by a corresponding value or string. Pure Data is capable of formatting and transmitting OSC messages across a network. This process involves using the [oscformat] object to construct packets byte by byte, enabling their transmission over the network in UDP binary mode. To send these messages to a remote host, the [netsend] object is used, which is configured with the destination's IP address or name and the associated port number utilized by the receiving host. [96]

### 6.7.3 Drawing to the Screen

The main.cpp file in the O2O bridge serves as an OSC receiver, specifically designed to listen and receive OSC messages with this pre-established format. These OSC messages are crucial in controlling the content displayed on the OLED screen. Whenever the receiver receives a message that conforms to the expected format, it utilizes the u8g2 library to render the corresponding content onto the screen. The actual content to be displayed is configured within the main.cpp code of the O2O bridge. However, the specific values required for rendering the content are supplied by Pd via OSC. [97]

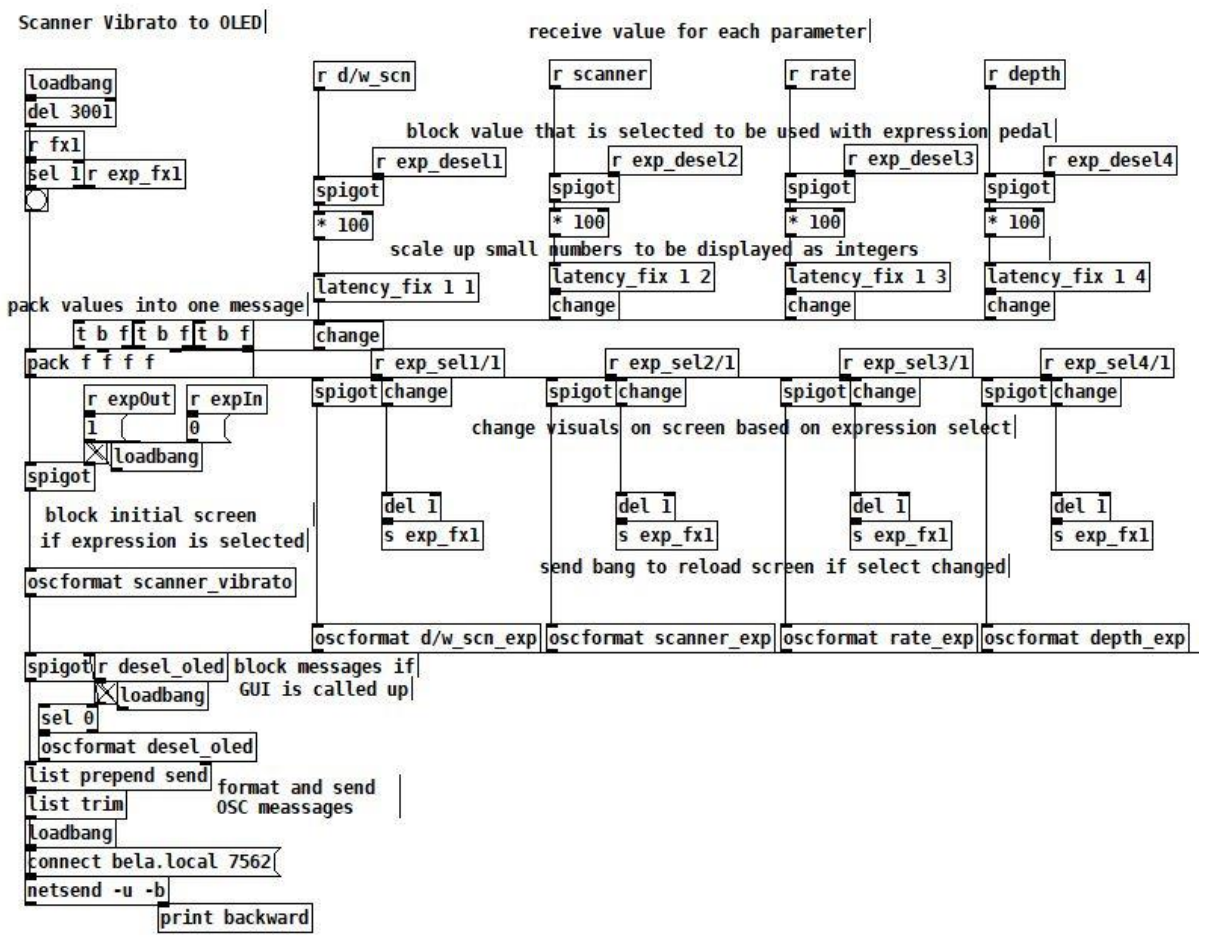


**Figure 46:** OLED Sub-Patch Section.

### 6.7.4 Running O2O as a Service

To ensure parallel functioning of the O2O bridge with the Delay_Chain project, it must be configured to function as a background service, actively monitoring OSC messages transmitted from the main project. This process involves creating a “system” service file within the /lib/systemd/system/ directory and specifying certain variables that “system” uses to establish the execution parameters for a program within the service. Running a program as a service on Bela proves advantageous when concurrent processes need to run alongside the main Bela program. However, it's important to note that any program executed this way cannot access Bela's audio, analog, and digital I/Os while another program is running. [98]

## 6.8 Graphical User Interface

### 6.8.1 The Bela GUI

To facilitate remote parameter adjustment, a graphical user interface (GUI) has been developed to display and control parameters via a smartphone or tablet. Bela offers an integrated GUI creation feature within its IDE, allowing users to design graphical interfaces to interact with their Bela system. The GUI integration uses p5.js, a JavaScript library that facilitates the creation of dynamic and powerful audiovisuals in web browsers. The GUI functionality facilitates bidirectional data transfer, allowing data to flow between the Bela project and the graphical interface. To implement the graphical user interface using p5.js, a file called "sketch.js" is added to the core project files. This code is responsible for creating the actual content that will be displayed on the GUI. [99]

### 6.8.2 Data Transfer

To establish communication between Pure Data and the GUI, the data transmission involves grouping values using the [pack] object in Pd, which is then sent to the GUI using [s bela_guiOut]. The sketch.js file receives these grouped values through Bela.data.buffers[0]. Conversely, for sending data from the GUI to Pd, the buffer containing all values in sketch.js is transmitted to Pd using Bela.data.sendBuffer(). These values are received in Pd using [r bela_guiControl] and distributed with the [unpack] object. [100]

### 6.8.3 GUI Examples

The graphical user interface of the Delay_Chain project is built upon two GUI examples available in the example section of Bela's IDE.

The first example code demonstrates how to send values from PD to the interface and visualize them as rings (see Figure 47). This project includes four oscillators grouped using the [pack] object in the PD patch and sent to the GUI with [s bela_guiOut]. In the sketch.js file, the values are received using Bela.data.buffers[0] and used to render the thickness of each of the four rings in the interface.

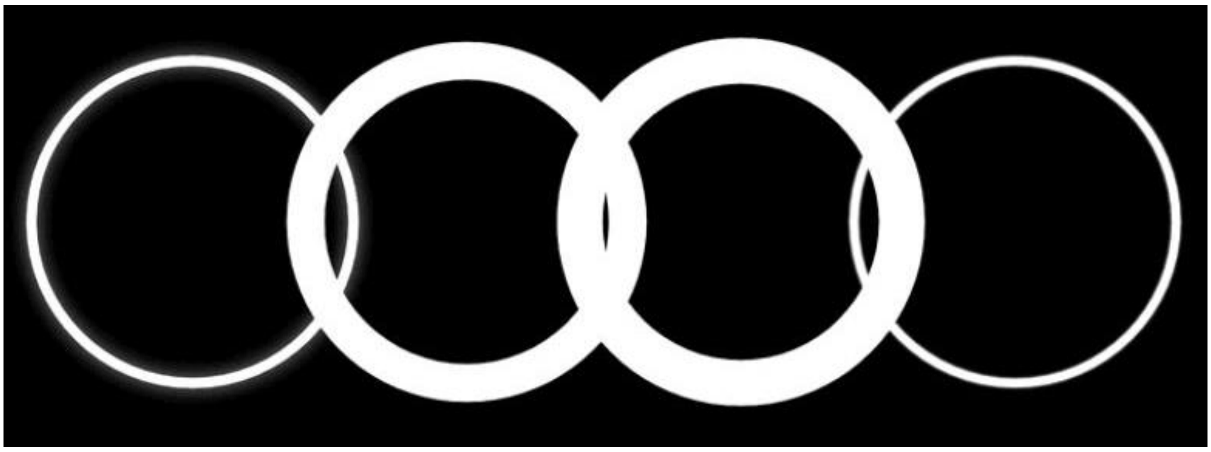


**Figure 47:** Pd to GUI Example. [101]

The second example code illustrates the process of mapping and transmitting the mouse's position on the screen to Pd. To achieve this, the sketch.js code captures the X and Y coordinates of the mouse and sends them to Pd using the command Bela.data.sendBuffer(0, 'float', [mouseX, mouseY]). These values are received in Pd using the object [r bela_guiControl] and can be utilized to control various parameters within the system.

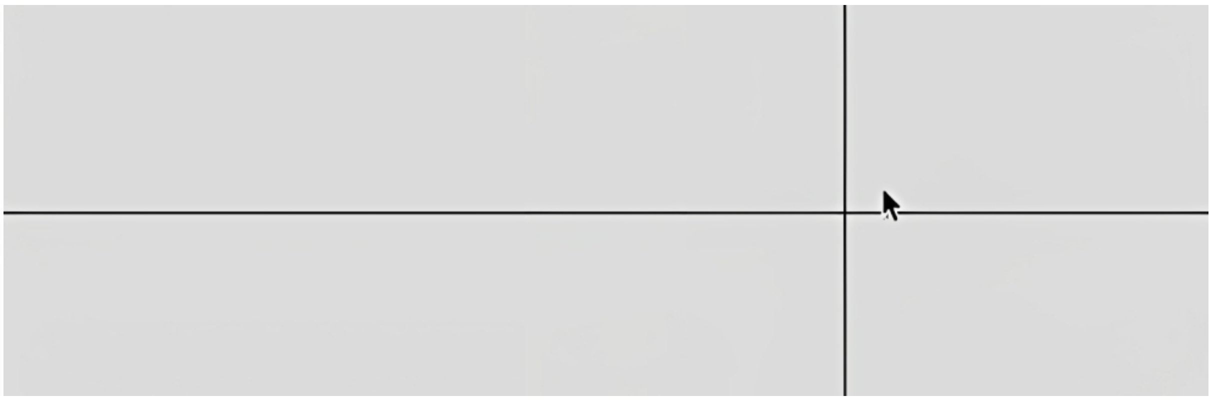


**Figure 48:** GUI to Pd Example. [102]

### 6.8.4 Creating the GUI

The visuals of the first example code were customized to match the layout of the physical interface by repositioning the rings accordingly. Additionally, the title of each effect and the name for each parameter were included, along with a textbox at the center of the rings to display their values. To distinguish between the three sets of effects in the Delay_Chain, an if statement was implemented to modify the color of the rings, text, and numbers based on the data received from Pd. Finally, to ensure the correct display of float numbers on the p5.js visualization, all values were converted to integers and multiplied appropriately.

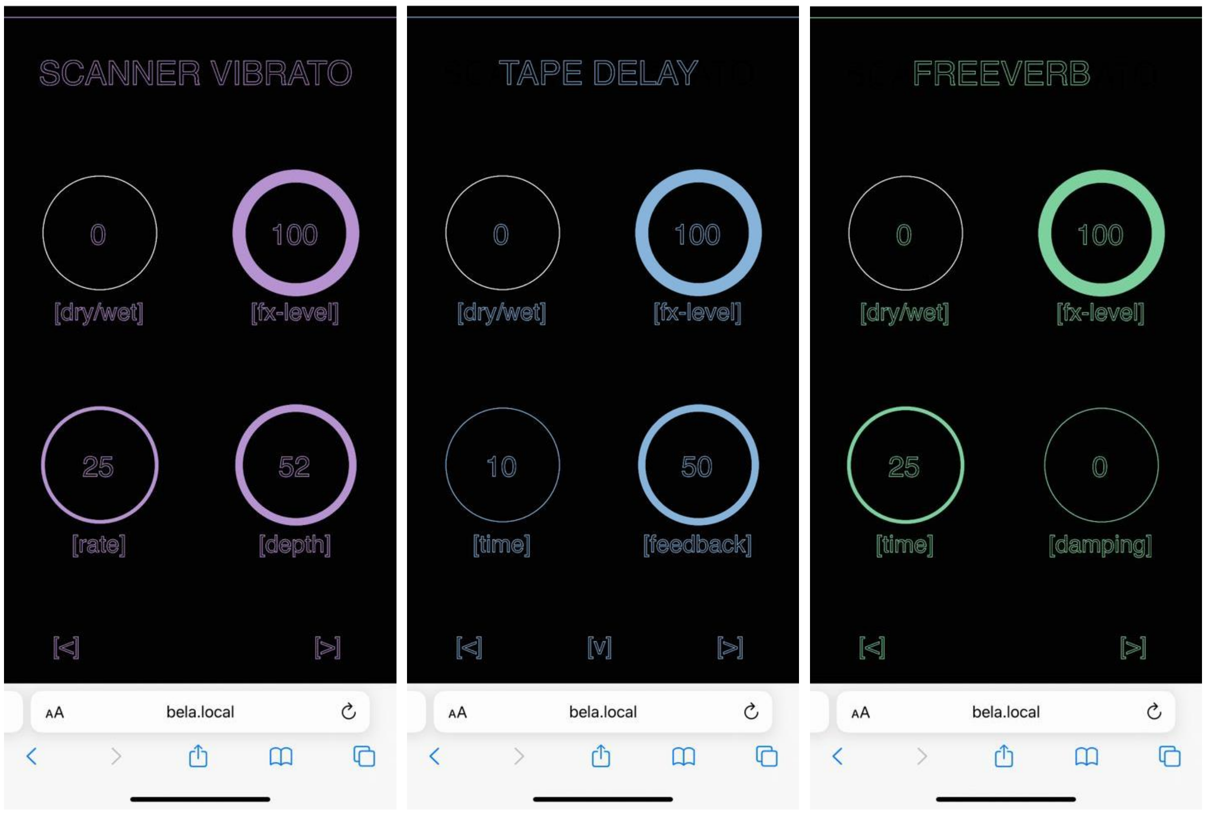


**Figure 49:** GUI Visual Layout.

### 6.8.5 Adding Touch Functionality

The utilization of the second example facilitated the incorporation of the touchscreen functionality into the GUI. To ensure compatibility with touchscreen devices such as smartphones or tablets, the mouse function was replaced with the touch function in the sketch.js code. This modification guarantees proper operation without the necessity of a cursor. Apart from reading and transferring the touch position and size of the screen, all other processes are managed in Pd. First, the display is partitioned into sections corresponding to the locations of visual knobs and buttons on the screen. Then, a selection mechanism is implemented, which determines the part of the display that is touched, directing the Y value from the touch function to adjust the corresponding encoder or activate the relevant button.

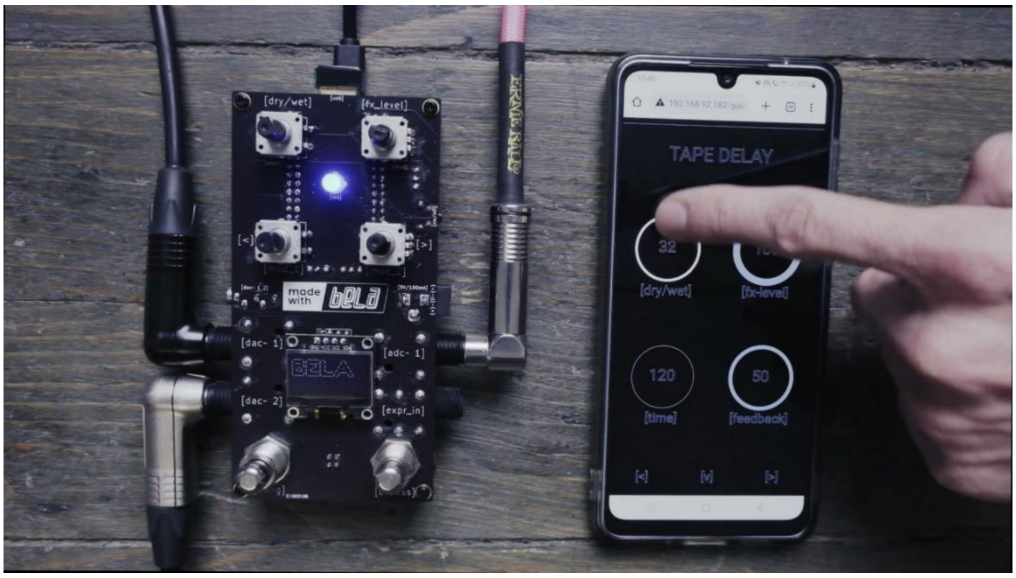


**Figure 50:** GUI Touch Functionality.

## 6.9 Network Setup

To access the GUI of Bela from any remote device, a network connection must be established with the board. Although Bela does not offer built-in network connectivity, it supports most Linux-compatible Wi-Fi Dongles that can be connected to its USB host. This method enables two configuration options: Bela can either be set up to connect to an existing network automatically or function as a wireless access point itself. [103]

### 6.9.1 Method 1

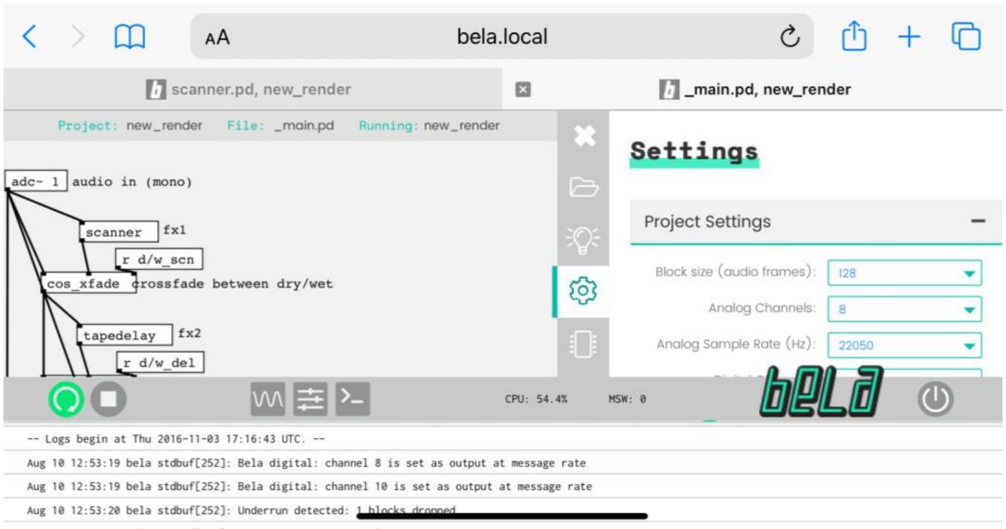
Configuring Bela to connect to a specific network is done by adding the network's name and password into the wpa_supplicant.conf file. The connection is accomplished by running the following command line in the IDE's terminal: root@bela:~# wpa_passphrase [Name] [Password] >>/etc/wpa_supplicant/wpa_supplicant.conf, where [Name] is replaced with the actual name of the Wi-Fi network and [Password] with its respective password. Once complete, the board will automatically search for the designated network and establish a connection whenever it is within range. For setting up a network connection with a smartphone or tablet, a hotspot connection can be created. This enables accessing the IDE and its integrated functions, including the GUI, without requiring an active internet connection.

### 6.9.2 Method 2

Setting up Bela as a wireless access point involves altering the network configuration to advertise its own network rather than attempting to connect to an existing one. This requires creating a hostapd.conf file containing the name of the network interface in use as well as the SSID and passphrase of the network. Furthermore, it is necessary to ensure that the interface acts as a DHCP server, assigning suitable IP addresses to connected devices. Converting the board into a network access point is more convenient than connecting to existing networks, as it avoids the need to provide network credentials for individual devices. However, this method requires more advanced dongle models capable of functioning as access points and may lead to incompatibility with certain operating systems, which might prevent a successful network connection.

### 6.9.3 Application

Once a network connection is successfully established with the Bela board, access the IDE using any web browser. This grants access to a range of functionalities and actions, including settings, GUI, and the oscilloscope, all accessible through a remote connection. Furthermore, the IDE can be easily added to the home screen, effectively transforming it into a mobile application. This convenient feature enhances accessibility and enables a greater level of control.



Project: new_render   File: _main.pd   Running: new_render

```
adc~ 1  audio in (mono)
scanner  fx1
r d/w_scn
cos_xfade  crossfade between dry/wet
tapedelay  fx2
r d/w_del
```

Block size (audio frames): 128

Analog Channels: 8

Analog Sample Rate (Hz): 22050

```
-- Logs begin at Thu 2016-11-03 17:16:43 UTC. --
Aug 10 12:53:19 bela stdbuf[252]: Bela digital: channel 8 is set as output at message rate
Aug 10 12:53:19 bela stdbuf[252]: Bela digital: channel 10 is set as output at message rate
Aug 10 12:53:20 bela stdbuf[252]: Underrun detected: 1 blocks dropped
```

**Figure 51:** Bela IDE on Mobile Device.

# 7. Effect Chain

## 7.1 The Hammond Vibrato Scanner Unit

The first effect in the chain is an emulation of Hammond's Vibrato Scanner unit, capable of producing different modulation effects. The original unit encompasses two main components: a delay line and a scanner. The delay line consists of a series of second-order audio filter stages where each stage is shifted in phase relative to the previous one. This results in an increasing delay between every successive step of the line (about 50µs per filter stage). The taps of the line are fed into the scanner, which is a single-pole 16-throw air-dielectric capacitor switch connecting nine selected taps from the line to the output. [104]

As the scanner gradually transitions between these taps, it alternates the phase shift applied to the sound, causing slight variations to the pitch that result in a vibrato effect. The depth of the vibrato depends on the width of the frequency shift fed into the scanner, which means that scanning about one-third of the delay line would produce a lighter vibrato effect while scanning the whole line would significantly increase its depth. [105] In addition to the vibrato, a chorus effect can be achieved by mixing the dry input signal with multiple outputs on the delay line. A notable limitation of this original design is the fixed gear ratio tied to the generator-run motor, which spins the rotor continuously at about 7Hz. [106]

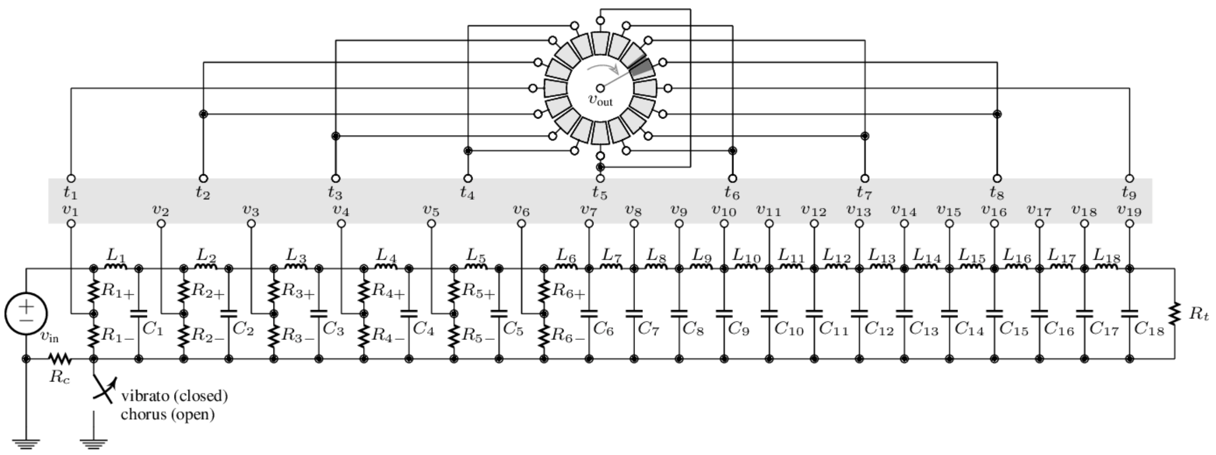

$v_{out}$
$t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$ $t_8$ $t_9$
$v_1$ $v_2$ $v_3$ $v_4$ $v_5$ $v_6$ $v_7$ $v_8$ $v_9$ $v_{10}$ $v_{11}$ $v_{12}$ $v_{13}$ $v_{14}$ $v_{15}$ $v_{16}$ $v_{17}$ $v_{18}$ $v_{19}$
$L_1$ … $L_{18}$
$R_{1+}$ … $R_{6+}$
$R_{1-}$ … $R_{6-}$
$C_1$ … $C_{18}$
$v_{in}$
$R_c$
$R_t$
vibrato (closed)
chorus (open)

**Figure 52:** Hammond Vibrato Scanner Schematic. [107]

## 7.2 Modelling in Pd

### 7.2.1 Considerations

Hammond's Vibrato Scanner is a unique effect that cannot be reproduced with simple LFOs. [108] To model the unit in Pure Data, a physical modeling approach was adopted, commencing with the challenge of emulating the delay line. Various externals for all-pass and comb filters could have been suitable for this purpose. However, the constraint of Bela supporting only Vanilla objects necessitated working within a limited object library.

### 7.2.2 Delay Line

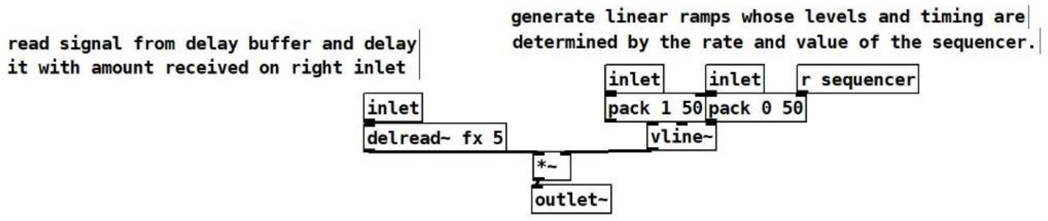
The primary approach to replicate the delay line was to utilize the [delwrite~] object, capable of allocating memory to a buffer and recording an audio signal into it. This is subsequently read by the [delread~] object with the same identifier, allowing for a variable delay time that can be defined in milliseconds. By employing multiple [delread~] objects for the same buffer, several delay taps were set up, each delayed with a couple of microseconds relative to the preceding one. Combining these delayed signals with the original one results in periodic amplifications and cancellations, effectively producing a comb filter.

### 7.2.3 Scanner

The subsequent step involved developing a mechanism akin to the scanner, enabling the sampling of these delay taps. This was achieved by creating a simple counter that incremented by a specified velocity and reset upon reaching the designated maximum value. The counter's value was then utilized in a select object to trigger an output when it matched a particular argument. This resulted in the creation of a sequencer, which essentially emulated the scanner's behavior.

### 7.2.4 Pairing

To connect the delay line to the sequencer, [vline~] objects were placed at the output of each tap. The [vline~] object generates linear ramps with controllable levels and timing through sent messages. By employing ramp-up and ramp-down messages for each [vline~] object, the volume of the delay taps could be toggled on and off in a specific order. This allowed each successive sequencer trigger to increase the volume of its assigned tap while reducing the volume of the previous one, ensuring smooth transitions between taps through rate-relative ramp durations.

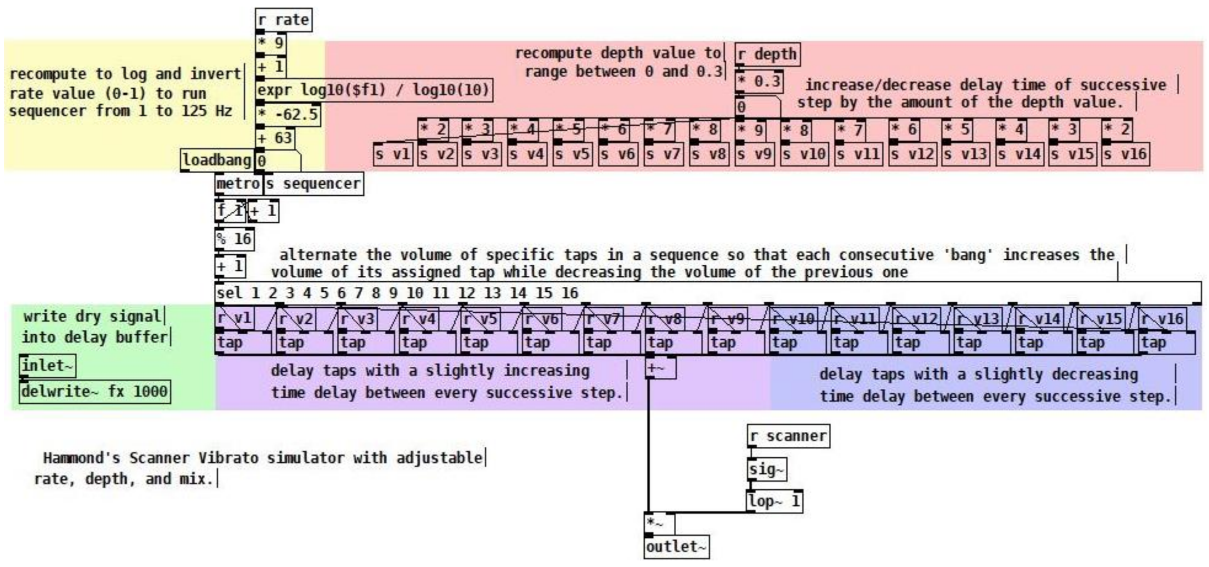


read signal from delay buffer and delay it with amount received on right inlet
generate linear ramps whose levels and timing are determined by the rate and value of the sequencer.
inlet
inlet
r sequencer
inlet
pack 1 50
pack 0 50
delread~ fx 5
vline~
*~
outlet~

**Figure 53:** Delay Line Tap Abstraction.

## 7.2.5 Result

Unlike the original design, which had fixed rates, depth, and mix settings, this emulator offers greater versatility by allowing users to adjust all parameters. It enables crossfading between the wet and dry signals, offering various mix settings and producing diverse chorus, vibrato, and chorus-vibrato effects. The time delay between the taps can be finely tuned, ranging from 0 to 300 microseconds, which alters the depth of the applied phase shift on the sound. Additionally, users can modify the sequencer's rate, achieving a speed anywhere between 1 to 125 Hz. This flexibility results in phaser-like effects at lower rates and ring shift modulation at higher rates.



r rate
* 9
+ 1
expr log10($f1) / log10(10)
* -62.5
+ 63
recompute to log and invert rate value (0-1) to run sequencer from 1 to 125 Hz
loadbang
0
recompute depth value to range between 0 and 0.3
r depth
* 0.3
0
increase/decrease delay time of successive step by the amount of the depth value.
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 8
* 7
* 6
* 5
* 4
* 3
* 2
s v1
s v2
s v3
s v4
s v5
s v6
s v7
s v8
s v9
s v10
s v11
s v12
s v13
s v14
s v15
s v16
metro
s sequencer
f 1
+ 1
% 16
+ 1
alternate the volume of specific taps in a sequence so that each consecutive 'bang' increases the volume of its assigned tap while decreasing the volume of the previous one
sel 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
write dry signal into delay buffer
inlet~
delwrite~ fx 1000
r v1
r v2
r v3
r v4
r v5
r v6
r v7
r v8
r v9
r v10
r v11
r v12
r v13
r v14
r v15
r v16
tap
+~
delay taps with a slightly increasing time delay between every successive step.
delay taps with a slightly decreasing time delay between every successive step.
r scanner
sig~
lop~ 1
*~
outlet~
Hammond's Scanner Vibrato simulator with adjustable rate, depth, and mix.

**Figure 54:** Hammond's Scanner Vibrato Simulator in Pure Data.

## 7.3 Tape Delay

The second effect in the signal chain is a tape delay simulator featuring variable-speed tape and saturation. This patch was originally created by Robert Thomas to run on Salt, a Eurorack module based on Bela. [109] The code is available as an example for the Bela platform and has been adapted to run on Bela Mini in combination with the extension cape.

This tape delay implementation aims to recreate the essential physical characteristics of a tape machine that contribute to its distinctive sound quality. Besides offering flexible delay times up to 5000 ms, it allows users to adjust the ramp time of the motor that is driving the tape. Additionally, along with a standard feedback parameter setting, a roll-off option is available to control the cut-off frequency of the delayed signal, giving further control over the delayed sound. Furthermore, to achieve the authentic sound quality of the physical tape, the patch incorporates a saturation algorithm. This algorithm utilizes a Pade approximation of the tanh function, which is applied to the delayed signal, replicating the warm and saturated tones characteristic of the original tape-based delays.

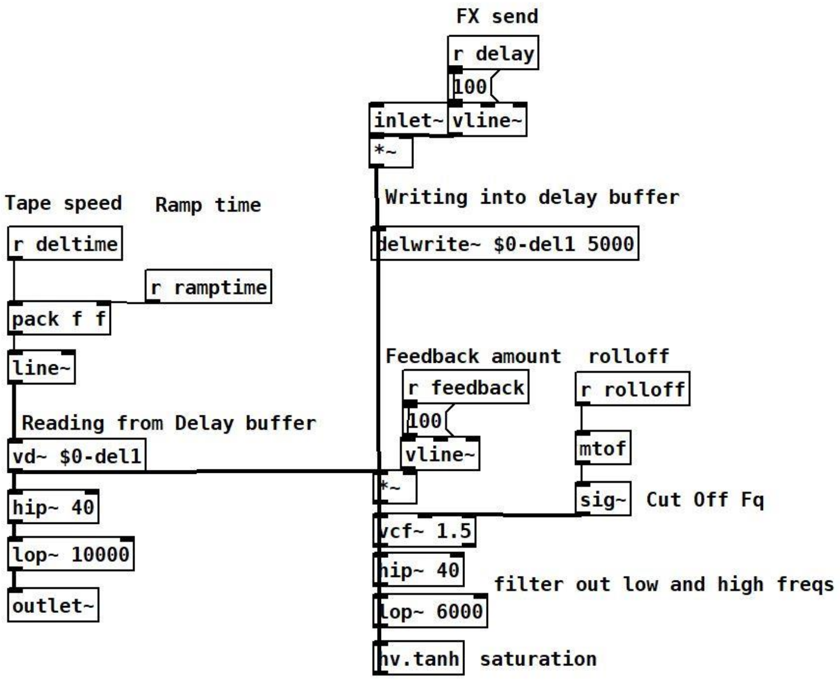


**Figure 55:** Tape Delay Simulator in Pure Data.

## 7.4 Freeverb

The third element of the chain is a Schroeder reverberator implementation using the Freeverb algorithm. This design draws inspiration from Manfred Schroeder's original artificial reverberation concept from the early 1960s, which involved interconnected all-pass filters, a bank of feedback comb filters in parallel, and a mixing matrix. A recent iteration of this reverberator is known as the Freeverb algorithm, which is open-source software developed in the C++ programming language. This implementation utilizes four consecutive all-pass filters and eight parallel filtered-feedback comb filters. It is highly regarded for its exceptional ability to generate high-quality reverberation effects while requiring little computational power. [110]

While Freverb is available as an external object for Pd, it wasn't possible to implement it on the Bela platform due to its limitations. Nevertheless, there is an alternative called [vfreeverb~] designed by Katja Vetter, which is a direct conversion of the Freeverb external using only Vanilla objects. [111] This application features adjustable room size by adjusting the amount of the dry signal sent to the filters. Additionally, a damping parameter is available that is achieved by applying a low pass filter between 22000 and 800 Hz to the output of the comb filters. Subsequently, just as in the case of the tape delay, a crossfade can be added to adjust the mix between the dry and wet signal.

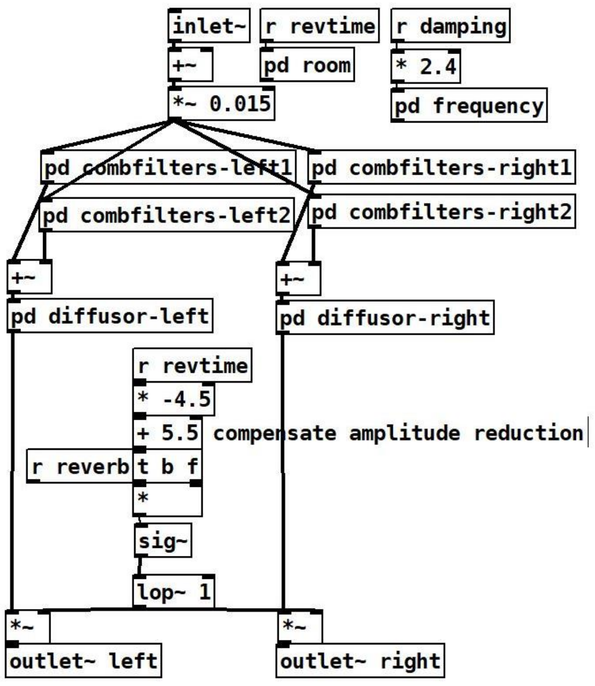


**Figure 56:** Freeverb Implementation in Pure Data.

# 8. Performance

## 8.1 Real-time Computing

Bela is a hard real-time system designed to guarantee a response within a specified time frame, meaning it must meet stringent deadlines to function properly. However, while Bela is powerful enough to meet most time constraints, more computationally intensive tasks like applying complex audio effects may require code optimization to achieve the desired performance. All I/O in Bela, including audio, analog, and digital, is synchronized to the same master clock, resulting in strong alignment with no latency or jitter. Due to this synchronization, handling I/O in Bela follows unique programming conventions compared to other platforms. [112]

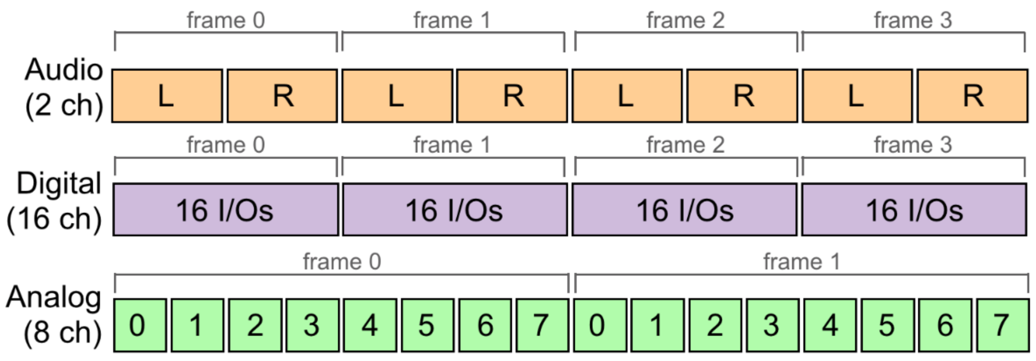


**Figure 57:** Beal I/O Processing Diagram. [113]

### 8.1.1 Block-based Audio Processing

The framework utilizes block-based audio processing to handle audio tasks efficiently. This approach is necessary as it must handle multiple processes simultaneously, making it challenging to ensure that the audio code runs precisely and produces each audio sample in time. To address this, the hardware groups audio samples into blocks, which are then passed to the audio callback function called "render." As the audio hardware plays back one block of samples, the processor simultaneously calculates the next one to ensure it is ready in time. This arrangement reduces the frequency of calls to the audio code, allowing the system to tolerate more timing uncertainty. The larger the block size, the more variation in timing the processor can handle. [114]

### 8.1.2 Latency

Processing audio in blocks introduces a trade-off in the form of latency. The larger the block size, the greater the latency, defined as the delay between an action and its reaction, which is a fundamental property of most DSP systems. Several sources contribute to latency in a digital system, including the audio converters themselves, which can add up to 500 microseconds each. Yet, the primary source of latency originates from buffering within the operating system. This buffering latency directly correlates with the audio block size, which in the case of Bela can range from 2 to 4096 samples, resulting in a possible latency between 1 and 190 milliseconds. [115]

Latency profoundly impacts the perceived quality of audio and interactive systems. Excessive latency can lead to audible delays between input and output, reducing the system's responsiveness. However, if the buffer size is too small, the processor may need more time to process the data, resulting in buffer underruns that manifest as gaps or clicks in the audio output. Striking the right balance between block size and latency is crucial for optimizing audio system performance and user experience. [116]

## 8.2 Latency Measurement

According to David Wessel and Matthew Wright, digital instruments should ideally have a latency of no more than 10ms. [117] To determine whether the latency specification of the device meets this criterion, a measurement has been carried out.

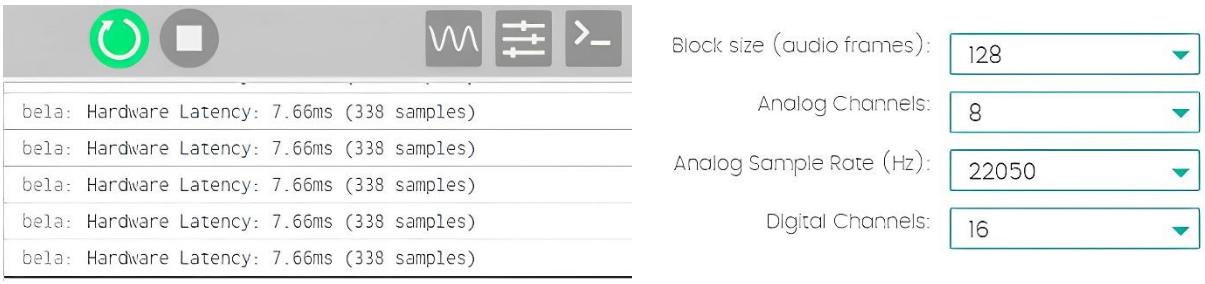
### 8.2.1 Setup

A second Bela Mini board was utilized to conduct the evaluation, running a latency measurement example code written in the C++ programming language. This code generates a regular pulse with a sudden onset and measures the time it takes for the pulse to return, providing an estimation of the latency introduced by any external audio device. [118]

The measurement device's output was connected to the input of the device under test, while its input received the output of the tested hardware. The code considers and subtracts the latency introduced by the measurement hardware, focusing solely on the latency caused by the device under test. The measurement results are displayed in both milliseconds and samples on the IDE console.

### 8.2.2 Results

**Test 1:** When running the complete Delay_Chain example project, a block size of 128 samples is necessary to prevent buffer underruns, ensuring a smooth audio output without gaps or clicks. At this block size, the measured latency is 338 samples, equivalent to 7.66 milliseconds at a sample rate of 44.1kHz. This result comfortably falls below the desired latency threshold of 10 milliseconds, ensuring adequate audio quality and responsiveness.

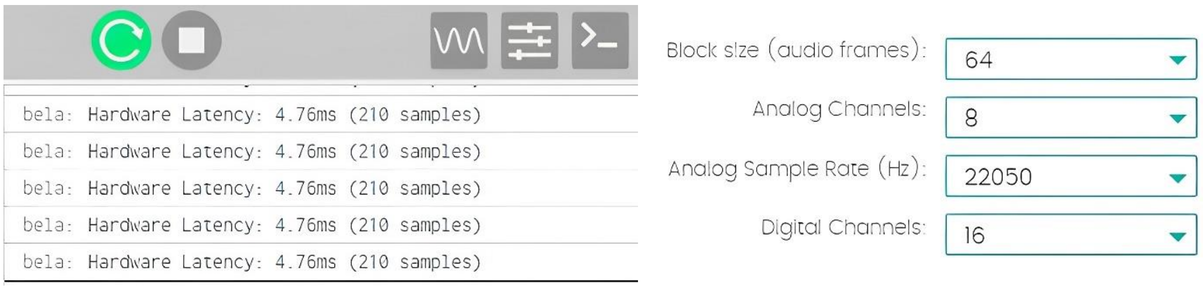


```
bela: Hardware Latency: 7.66ms (338 samples)
bela: Hardware Latency: 7.66ms (338 samples)
bela: Hardware Latency: 7.66ms (338 samples)
bela: Hardware Latency: 7.66ms (338 samples)
bela: Hardware Latency: 7.66ms (338 samples)
```

| | |
|---|---|
| Block size (audio frames): | 128 |
| Analog Channels: | 8 |
| Analog Sample Rate (Hz): | 22050 |
| Digital Channels: | 16 |

**Figure 58:** Latency Measurement Test 1.

**Test 2:** Removing the GUI functionality from the project could further reduce the latency. By discarding both the sketch.js code and [gui] sub-patch, the project can run with a block size of 64 samples without encountering underruns. With this configuration, the measured latency drops to 4.76 milliseconds (210 samples), less than half of the maximum acceptable latency established by Wessel and Wright.

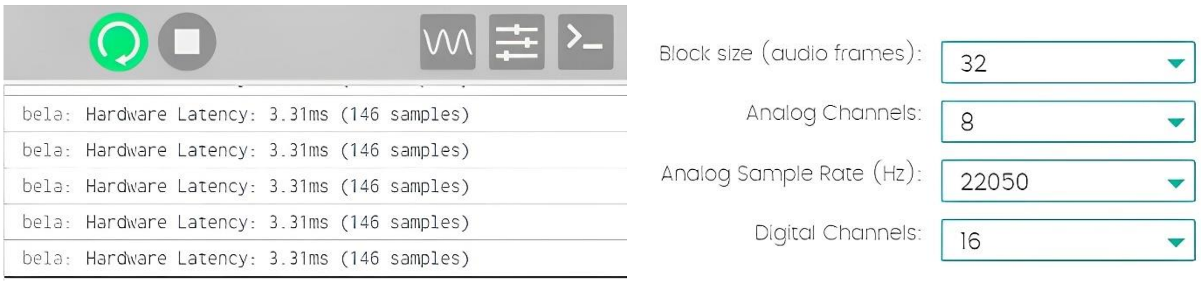


```
bela: Hardware Latency: 4.76ms (210 samples)
bela: Hardware Latency: 4.76ms (210 samples)
bela: Hardware Latency: 4.76ms (210 samples)
bela: Hardware Latency: 4.76ms (210 samples)
bela: Hardware Latency: 4.76ms (210 samples)
```

| | |
|---|---|
| Block size (audio frames): | 64 |
| Analog Channels: | 8 |
| Analog Sample Rate (Hz): | 22050 |
| Digital Channels: | 16 |

**Figure 59:** Latency Measurement Test 2.

**Test 3:** Additionally, it is possible to run the project with an even smaller block size of 32 samples, achieving a latency of 3.31 milliseconds (146 samples). However, this setup results in occasional block drops, particularly during parameter adjustments. Attempting to remove further components, like the OLED functionality, did not entirely resolve the underrun issue. Running the project with a block size of 32 samples would either necessitate significant software restructuring or 64 samples might be the practical minimum for operating a project of this complexity on Bela.



```
bela: Hardware Latency: 3.31ms (146 samples)
bela: Hardware Latency: 3.31ms (146 samples)
bela: Hardware Latency: 3.31ms (146 samples)
bela: Hardware Latency: 3.31ms (146 samples)
bela: Hardware Latency: 3.31ms (146 samples)
```

Block size (audio frames): 32
Analog Channels: 8
Analog Sample Rate (Hz): 22050
Digital Channels: 16

**Figure 60:** Latency Measurement Test 3.

## 8.3 Cross-platform Testing

Given that the device is based on an open platform, assessing its ability to run effects designed for other comparable platforms was essential. To achieve this, patches designed for the OWL, Organelle, Daisy, and a Raspberry Pi custom unit were incorporated into the device, creating a signal chain of cross-platform audio effects. This multi-effect setup consists exclusively of Pure Data patches, as the existing software framework is developed in this programming language. This way, only the audio code had to be replaced in the main patch of the Delay_Chain project, as this already encompassed all framework functionalities.

### 8.3.1 Daisy

The first element in the cross-platform chain is a bi-quad (second order) lowpass filter with adjustable cut-off frequency and resonance settings. This effect is a sample patch for the Daisy Pod, a breakout board designed for the Daisy Seed. [119] The Daisy Pod offers various audio input/output options, such as 3.5mm stereo jacks for line-level audio, a headphone output, and

TRS MIDI input. Additionally, it provides two tactile switches, two RGB LEDs, two analog potentiometers, and a rotary encoder. [120]

Since the effect is built entirely using Pd Vanilla objects, incorporating it on Bela didn't necessitate any modifications to the audio code. Only adjustments to the software framework for managing the physical interface were needed. The integration was achieved by creating a sub-patch called [daisy], containing only the necessary audio code, and placing it in the position of the Scanner Vibrato effect within the main patch of the Delay_Chain project. Subsequently, new rotary encoder ranges were set up to cater to the specific requirements of this effect. Additionally, the names of the effects and their parameters had to be updated in the OLED and GUI code.

As a result, the Daisy effect became seamlessly integrated and interconnected with all the functionalities provided by the extension cape. For instance, the expression pedal input could be assigned to control the cut-off frequency of the lowpass filter, effectively transforming it into a wah-wah pedal.

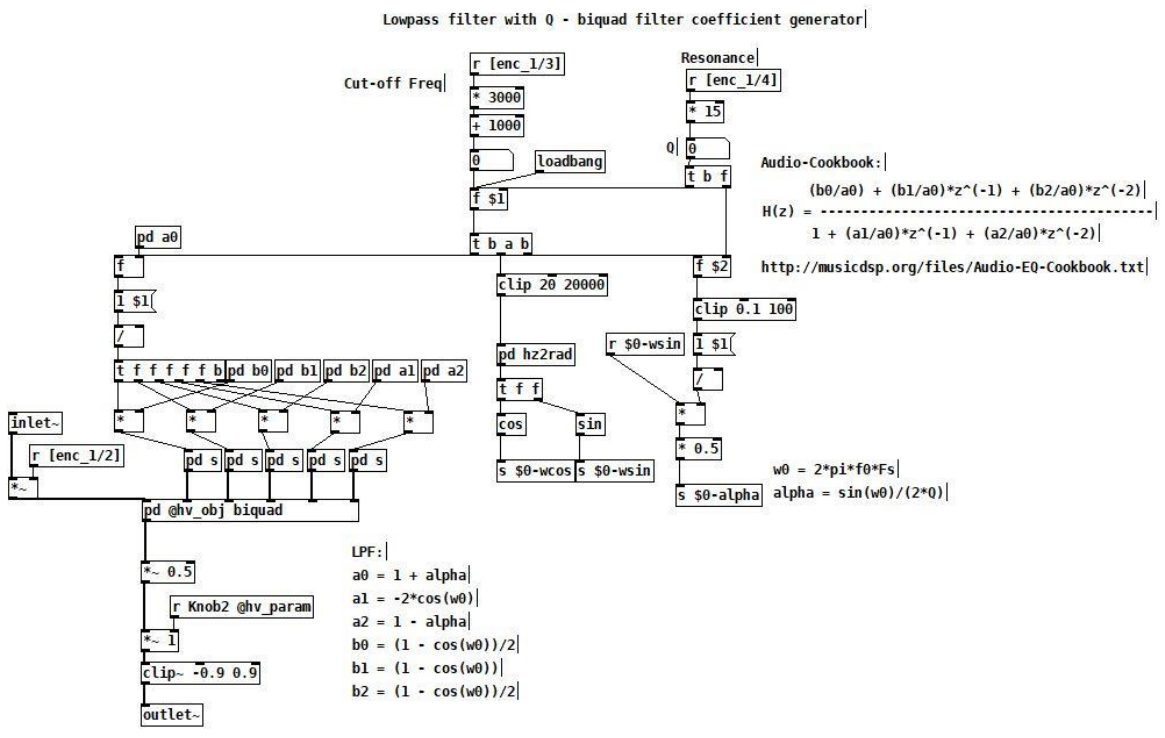


**Figure 61:** Daisy - Lowpass Filter Patch Integration.

## 8.3.2 OWL

The second effect in the signal chain is a tremolo designed in the tube amp style specifically for the OWL. It is available in the official Patch Library on the producer's website. [121] The OWL pedal has four analog potentiometers, one footswitch, one LED, stereo audio I/O, and an expression pedal input, [122] which means all effects designed for this platform usually only have four parameters. For the selected tremolo effect, the rate, resonance, filter depth, and tremolo depth can be adjusted, while the expression pedal is exclusively assigned to control the rate. Additionally, the code incorporates a tap tempo functionality to set the effect's rate, which is assigned to the footswitch of the pedal.

Since this patch also utilizes Pd Vanilla objects, the integration process followed the same steps as with the Daisy patch. Initially, the [owl] sub-patch was set up, containing all the necessary code for the effect, which was then replaced with the Tape Delay patch in the Delay_Chain project to take the second position in the cross-platform signal chain. Rotary encoder abstractions were established for each tremolo parameter, and the OLED and GUI codes were updated to display the corresponding information. Additionally, the tap tempo functionality was assigned to the [bang] footswitch on the extension cape to set the rate of the tremolo effect.

By following these steps, the OWL effect could also be seamlessly integrated and interconnected with all the functionalities provided by the extension cape. In contrast to the OWL, with the effect cape, the expression pedal can be assigned to all parameters and not only the rate.

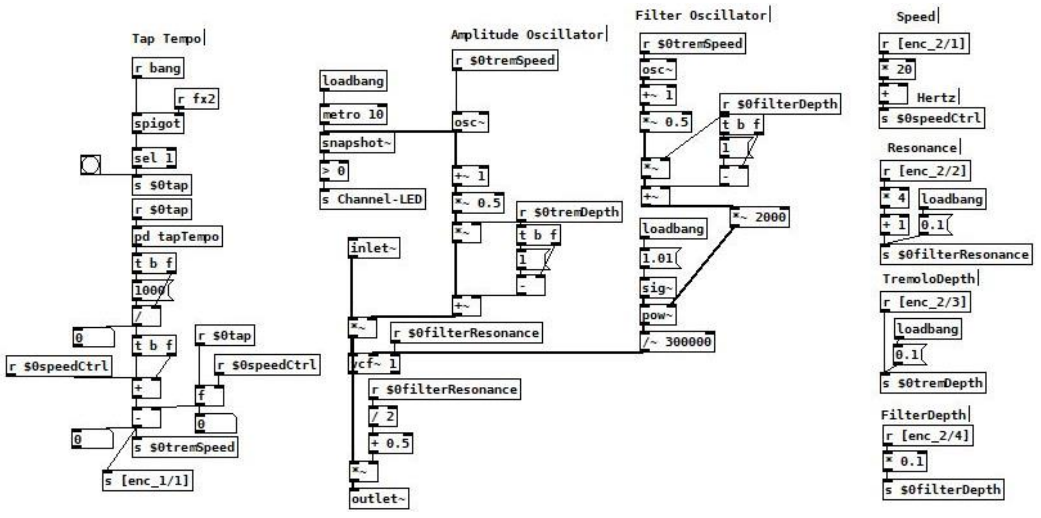


**Figure 62:** OWL - Tremolo Patch Integration.

### 8.3.3 Organelle

The third effect in the chain is the LFO Delay, which is available in the official patch directory of the Organelle on the Critter & Guitari website. [123] The Organelle has arguably the most complex physical interface among all the chosen platforms. This includes an OLED display screen, an RGB LED, four parameter knobs, a rotary selection knob with a push-button select, a volume knob, and twenty-five maple keys. [124] In the case of the LFO Delay, the first two parameter knobs are responsible for controlling the LFO Rates of both Delay A and B, the third knob sets a common delay time, and the fourth knob adjusts the Dry/Wet Mix. Additionally, it is worth noting that a sequencer is also available in the patch, but it requires using the maple keys, so it wasn't implemented for this particular purpose.

Integrating a sound effect designed for the Organelle demands more time and consideration than the previous two platforms due to the complexity and distinctness of its software framework. The Organelle platform employs a "Mother Patch" that serves as a bridge between the hardware and Pd, handling all functionalities and I/O routing. Consequently, all patches designed for this platform include numerous individualized objects that cater to various hardware functionalities. Additionally, many Organelle patches incorporate extended Pd objects, which can cause compatibility issues with the Bela platform.

Fortunately, the LFO Delay patch consists solely of Pd Vanilla objects, making it compatible without any problems related to extended objects. The main challenge during implementation was finding and extracting the necessary code for the audio effect, as the project comprises several sub-patches containing both the audio code and code for addressing hardware functionalities. Additionally, certain elements, such as the sequencer, had to be removed since they couldn't be used due to the lack of buttons on the extension cape. After extracting the essential code snippets and setting up the essential encoder abstraction, it became possible to create the [organelle] sub-patch and integrate it as the third effect in the chain.

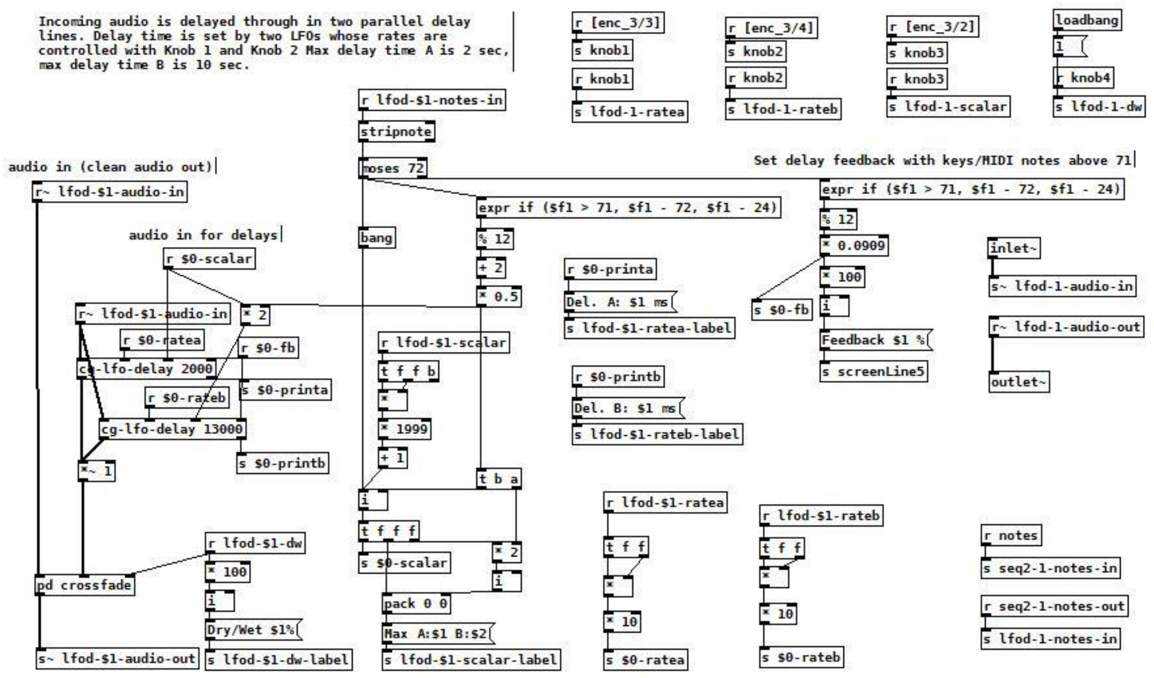


**Figure 63:** Organelle - LFO Delay Patch Integration.

### 8.3.4 Raspberry Pi

The fourth effect added to the cross-platform chain is a looper created by Pierre Massat, available through his blog "Guitar Extended." [125] This patch is designed to run on a Raspberry Pi, interfacing with a standard Arduino Uno, which connects 12 momentary push buttons and four analog potentiometers. [126] It presents functionalities such as starting/stopping recording, setting the length of the array buffer, clearing the buffer, and an on/off bypass switch. Additionally, it offers features like loop reversing and exporting the loop as a WAV file.

Although this patch is not fully compatible with Pd Vanilla, as it incorporates the [z~] and [limiter~] extended objects from the Zexy extended library, workarounds have been introduced for the Bela platform. Instead of compiling these extended objects for Bela, suitable Vanilla replacements are used, enabling the proper functioning of the looper patch on the Bela platform.

Setting up this patch requires additional software frameworks to achieve full functionality with the extension cape. Firstly, more complex footswitch functionalities are needed to serve as record, playback, overdub, stop, play, and clear functions. For this purpose, the [bang] footswitch is assigned to start recording/overdubbing on the first impulse and start playback on the second impulse. The [bypass] footswitch is configured to stop and play the loop on each signal, and a double tap clears the buffer. Additional LED actions are set up to signal record mode, playback mode, and the loop restart. Furthermore, the top left encoder is set up to control the overall output volume of the looper's signal, and the push button of the same encoder is set to export the content of the buffer in WAV format. All exported loops are accessible and can be downloaded from the respective project folder in the Bela IDE. Apart from the audio reverse functionality, which overloads the CPU of the Bela board, the entire patch is able to run on the platform.

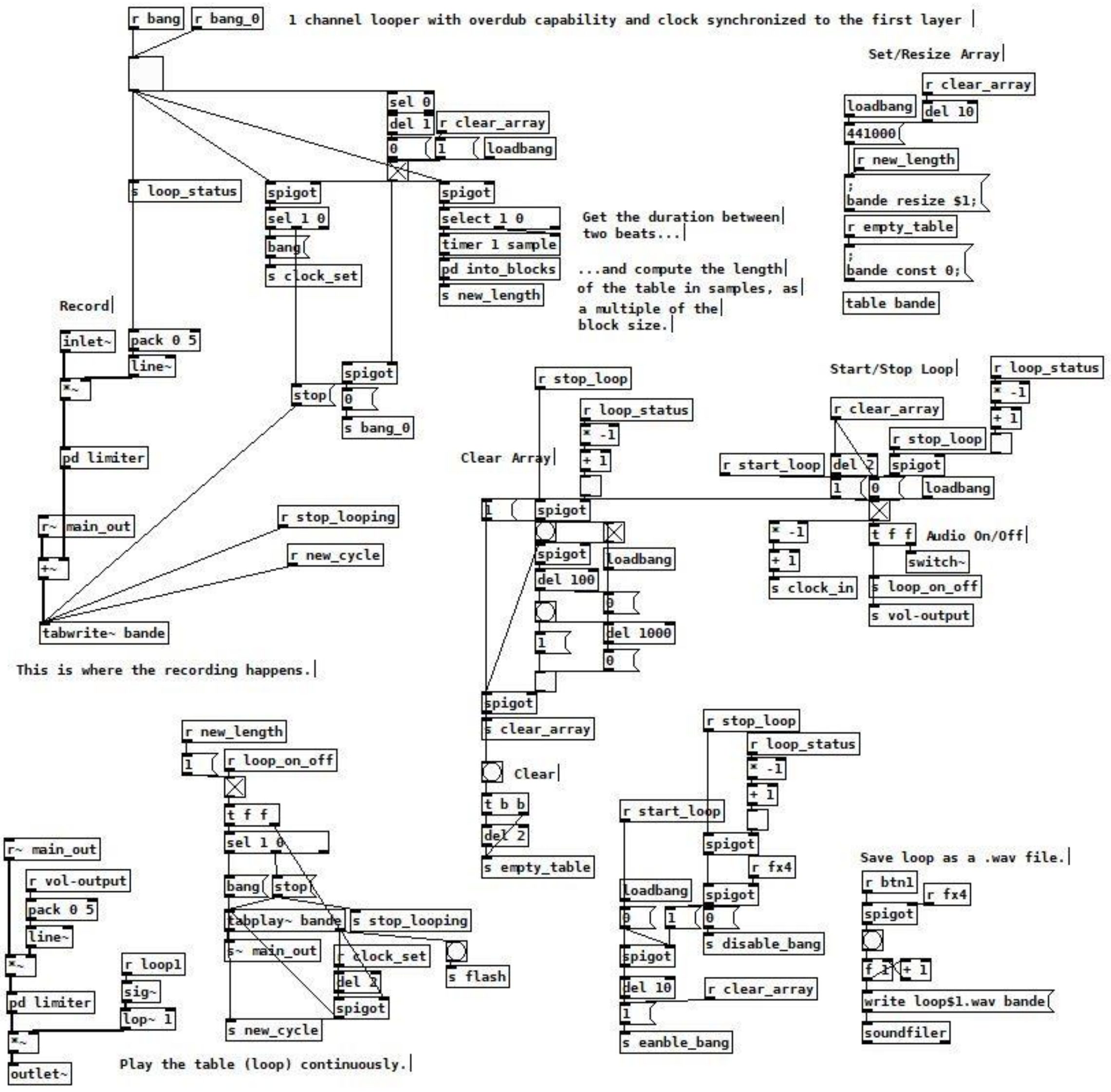


**Figure 64:** Raspberry Pi Custom Unit - Looper Patch Integration.

### 8.3.5 Results

Pure Data patches from all four platforms were successfully integrated and executed. Patches intended for the OWL and Daisy platforms were relatively straightforward to implement, benefiting from their low CPU load as microcontroller platforms, as well as their exclusive use of Vanilla objects. On the other hand, patches from both the Organelle and the Raspberry Pi custom unit are also functional, but they necessitate more setup time and adjustments. Some components had to be excluded due to either the limited processing capabilities of the Bela board or design differences in the physical interface.

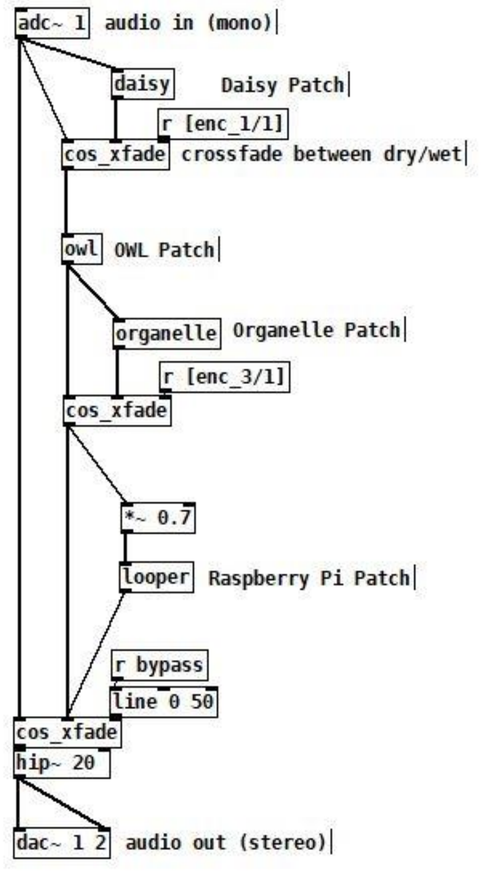


**Figure 65:** Cross-Platform Effect-Chain.

# 9. Online Study Survey

To further assess the outcome of the project, an online study survey was carried out. Participants were presented with a demo video [127] and asked to evaluate the device's practical and artistic applications while also assessing the significance of customization and open design in audio hardware production.

## 9.1 Result

The results of the survey are categorized into three sections. The first section examines the importance of customization and reconfigurability of effect pedals, the second assesses the device's practical and artistic applications, while the third delves into the importance of open design in audio hardware production,

### 9.1.1 Section 1

**Question 1** (see Figure 66)
Among the 19 participants, 12 individuals indicated that they are either familiar or very familiar with guitar effect pedals and their usage. Meanwhile, 3 participants classify themselves as intermediate, and 4 participants state that they are either not familiar or not at all familiar with the subject.

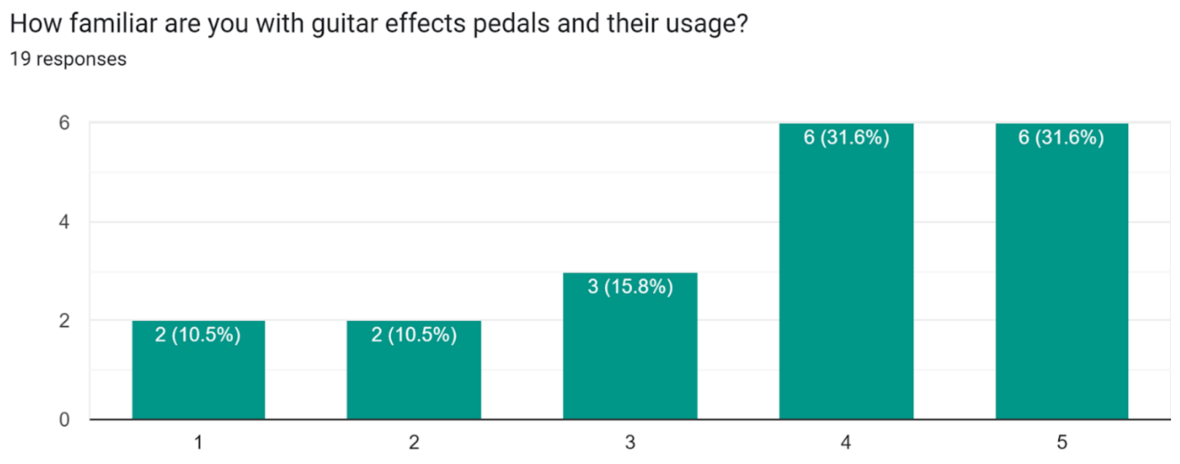


**Figure 66:** Survey Question 1.

**Question 2** (see Figure 67)

Only three out of the nineteen participants have prior experience with reconfigurable effect pedals.

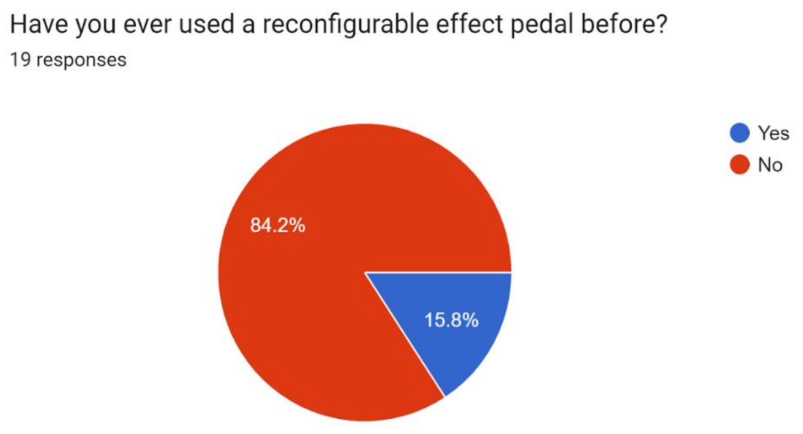


**Figure 67:** Survey Question 2.

**Question 3** (see Figure 68)

Sixteen out of the nineteen individuals are acquainted with the Pure Data programming environment.

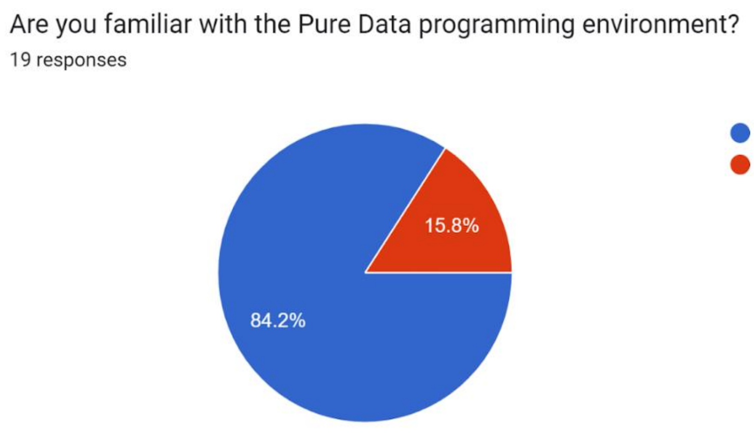


**Figure 68:** Survey Question 3.

**Question 4** (see Figure 69)

Out of the sixteen participants indicating familiarity with Pure Data, twelve assess themselves as intermediate or advanced users, two consider themselves to be relative beginners, and an additional two regard themselves as experts.

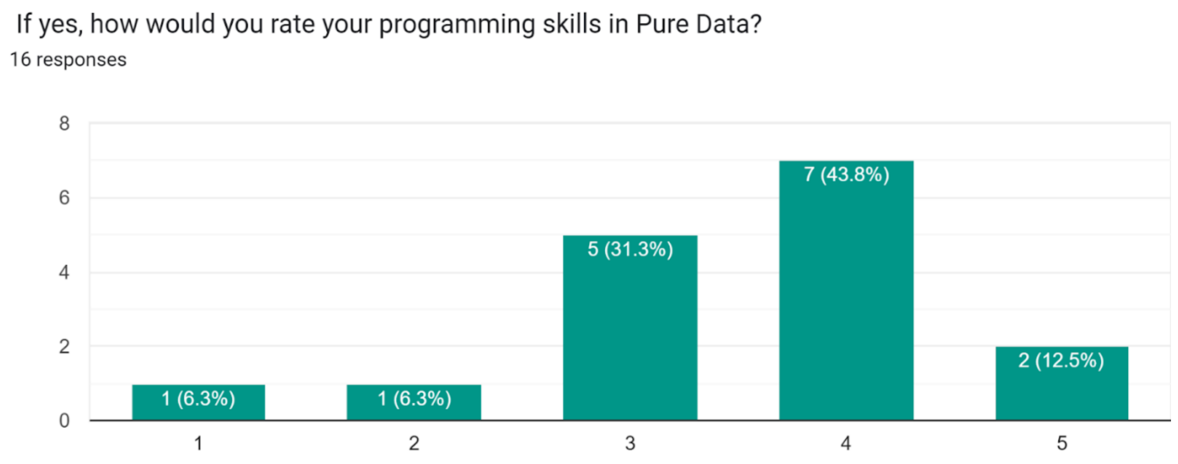


**Figure 69:** Survey Question 4.

**Question 5** (see Figure 70)

Ten out of the nineteen respondents deem the ease of programming and customization, in effect, pedals to be significant or highly significant. Eight participants attribute moderate importance to it, and only one person regards it as of minor significance.

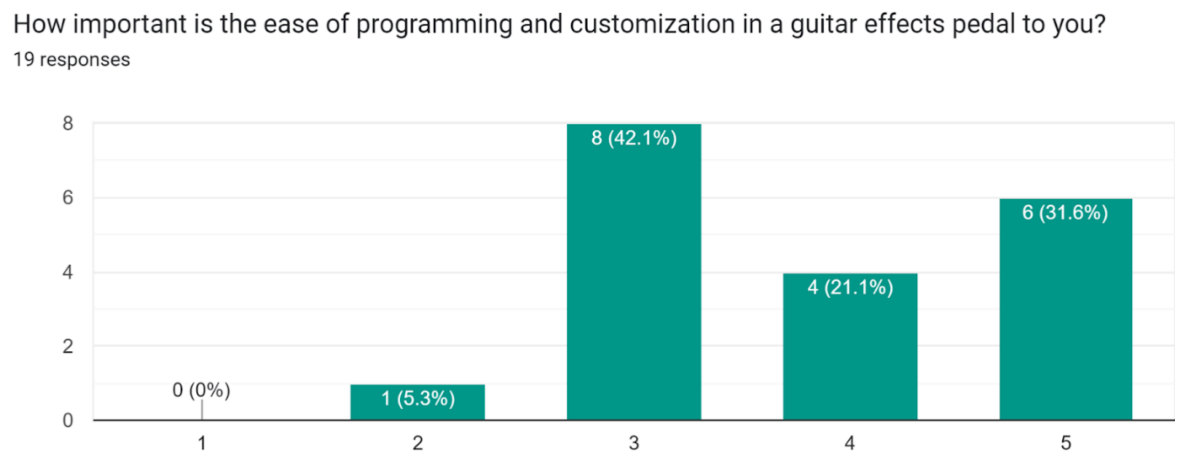


**Figure 70:** Survey Question 5.

**Question 6** (see Figure 71)

Seventeen out of the nineteen respondents view the capability to craft effects that align with their individual musical style as either important or highly important. Conversely, only two assign it a moderate or minor level of significance.

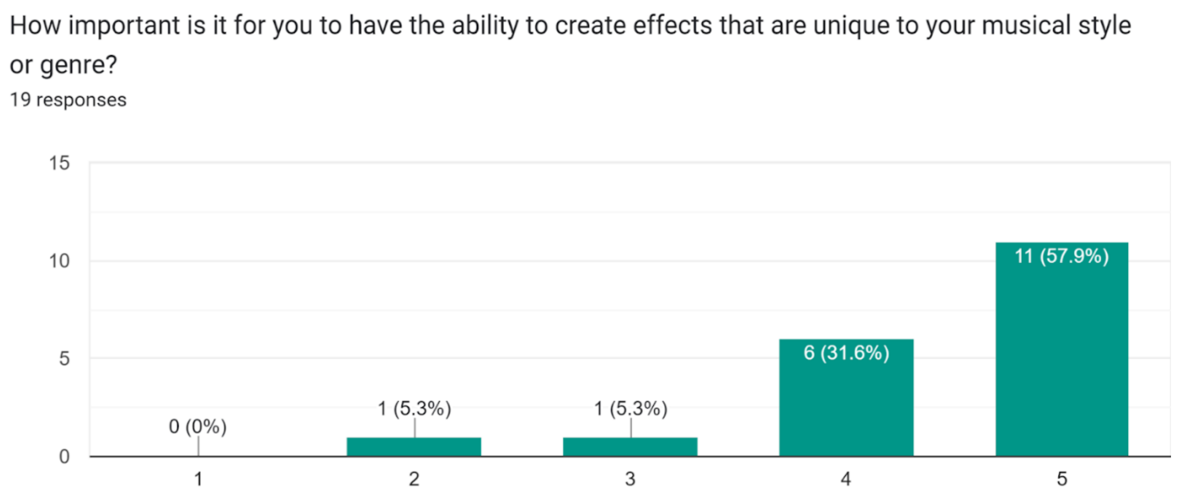


**Figure 71:** Survey Question 6.

**Question 7** (see Figure 72)

Thirteen out of the nineteen respondents are inclined or very inclined to contemplate using a reconfigurable effect pedal as a substitute for audio effects that are rare or outdated. Meanwhile, five have uncertainties, and one respondent is less likely to do so.

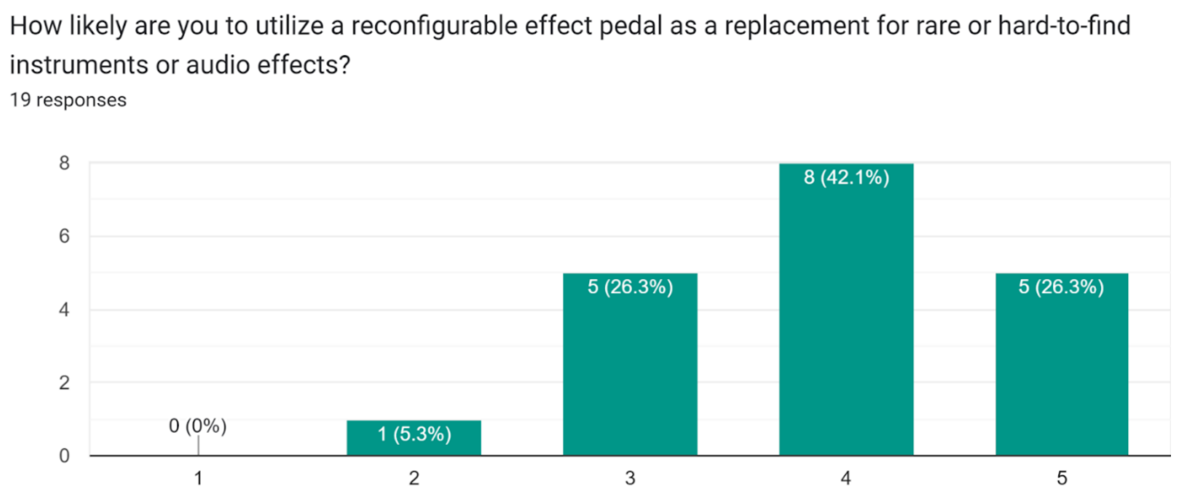


**Figure 72:** Survey Question 7.

**Question 8** (see Figure 73)

Eight participants consider having encountered situations where they felt constrained by the pre-defined effects offered in commercial guitar pedals while striving to realize a particular artistic concept. On the other hand, eight participants had never undergone such restrictions, while three couldn't recollect whether they had or not.

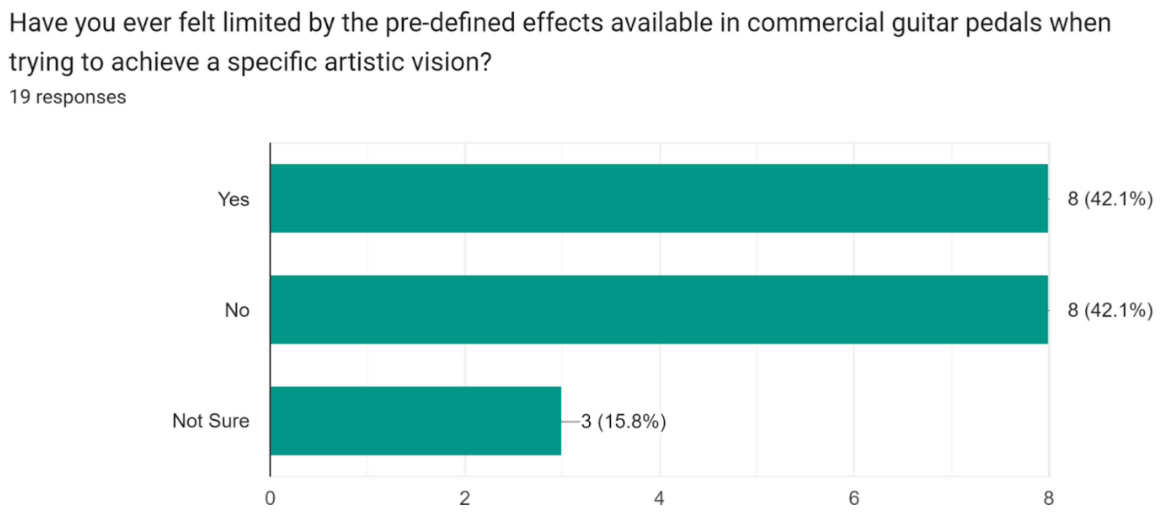
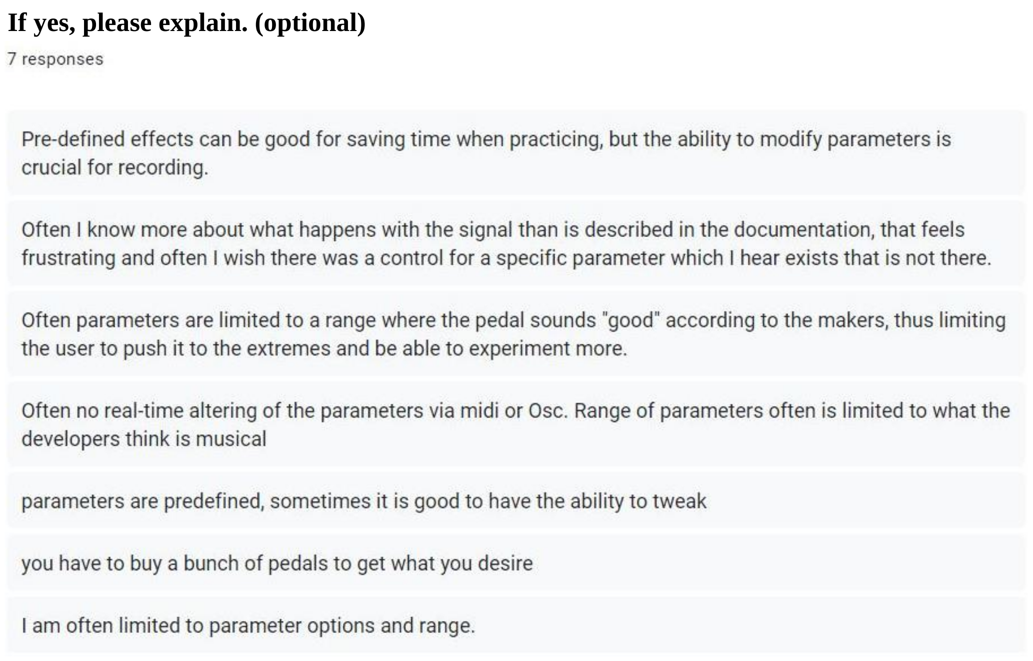


**Figure 73:** Survey Question 8.

**Optional Explanation** (see Figure 74)

Participants who have encountered limitations with predefined effects attribute it to the restricted parameter choices and ranges established by producers to align with their notion of what is considered pleasing or musical. This limitation, in turn, restricts users from pushing the effects to their limits and conducting more extensive experimentation. One respondent acknowledges that pre-defined effects can be advantageous for efficient practice but emphasizes that the capability to adjust parameters is indispensable when it comes to recording.


**If yes, please explain. (optional)**

7 responses

Pre-defined effects can be good for saving time when practicing, but the ability to modify parameters is crucial for recording.

Often I know more about what happens with the signal than is described in the documentation, that feels frustrating and often I wish there was a control for a specific parameter which I hear exists that is not there.

Often parameters are limited to a range where the pedal sounds "good" according to the makers, thus limiting the user to push it to the extremes and be able to experiment more.

Often no real-time altering of the parameters via midi or Osc. Range of parameters often is limited to what the developers think is musical

parameters are predefined, sometimes it is good to have the ability to tweak

you have to buy a bunch of pedals to get what you desire

I am often limited to parameter options and range.

**Figure 74:** Optional Explanation.

**Question 9** (see Figure 75)

Ten out of the nineteen participants would likely or very likely embrace an effects pedal that provides a significant degree of customization, even if it demands a certain amount of learning and setup. Meanwhile, seven participants expressed reservations, and two were not open to the idea of adopting such a pedal.

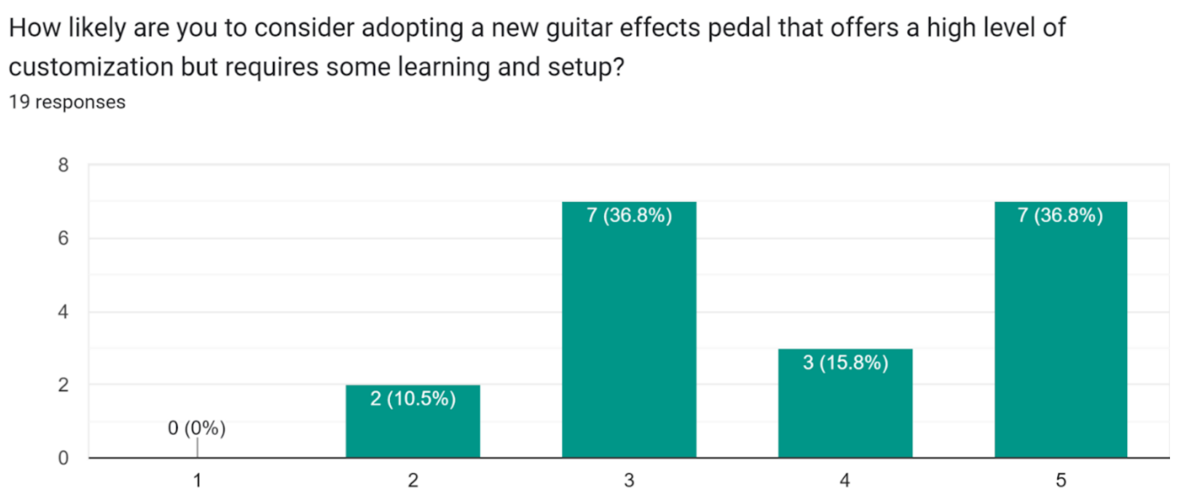


**Figure 75:** Survey Question 9.

**Question 10** (see Figure 76)

Every participant shares the belief that reconfigurable effect pedals have the potential to ignite innovative, creative approaches to guitar playing and composition.

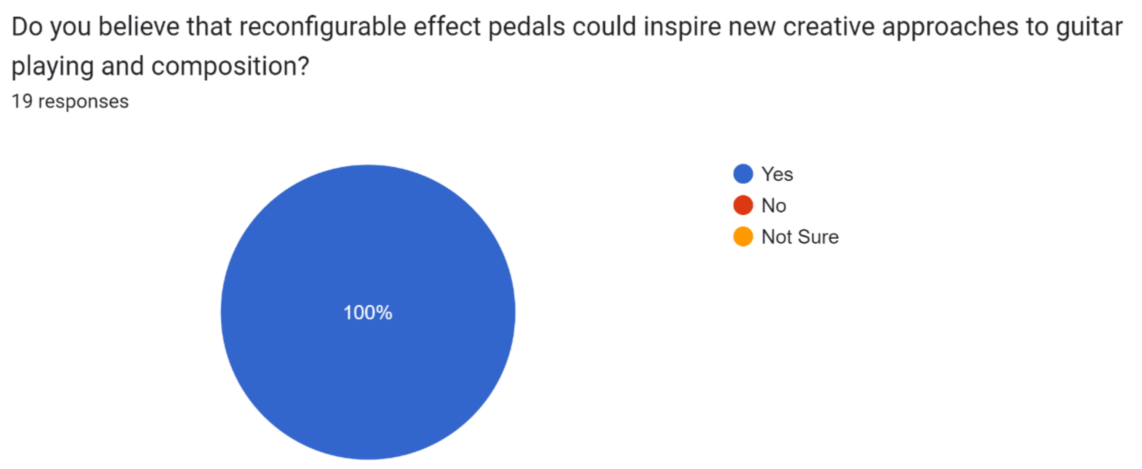


**Figure 76:** Survey Question 10.

### 9.1.2 Section 2

**Question 11** (see Figure 77)

Sixteen participants consider having a user-friendly interface and intuitive controls on effect pedals to be important or highly important. In contrast, two respondents assign it a moderate level of importance, while one participant views it as less significant.

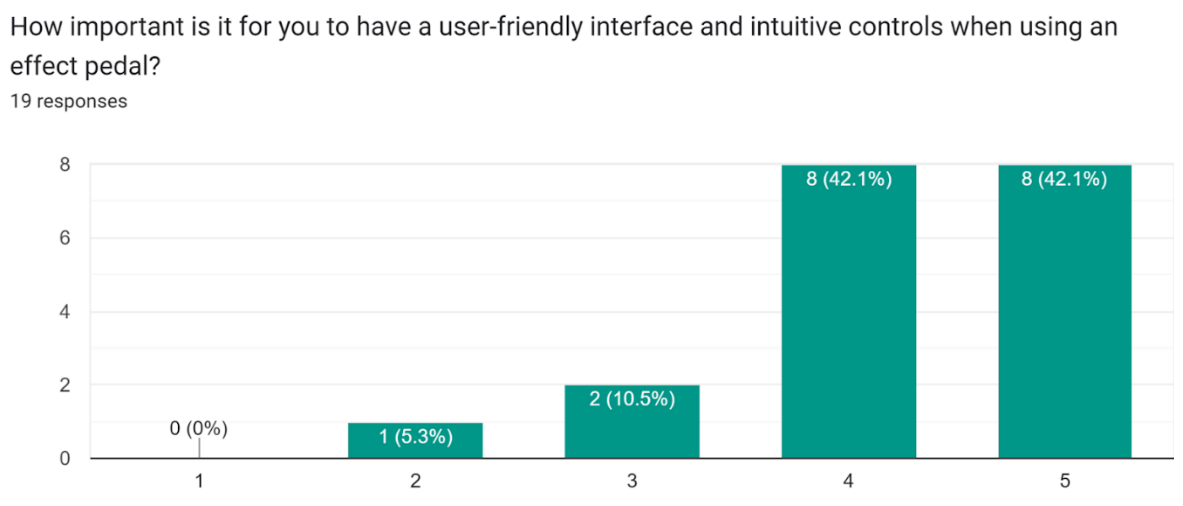


**Figure 77:** Survey Question 11.

**Question 12** (see Figure 78)

Seven participants assess the device's interface for its user-friendliness as average, nine consider it easy, and three perceive it as very easy to use.

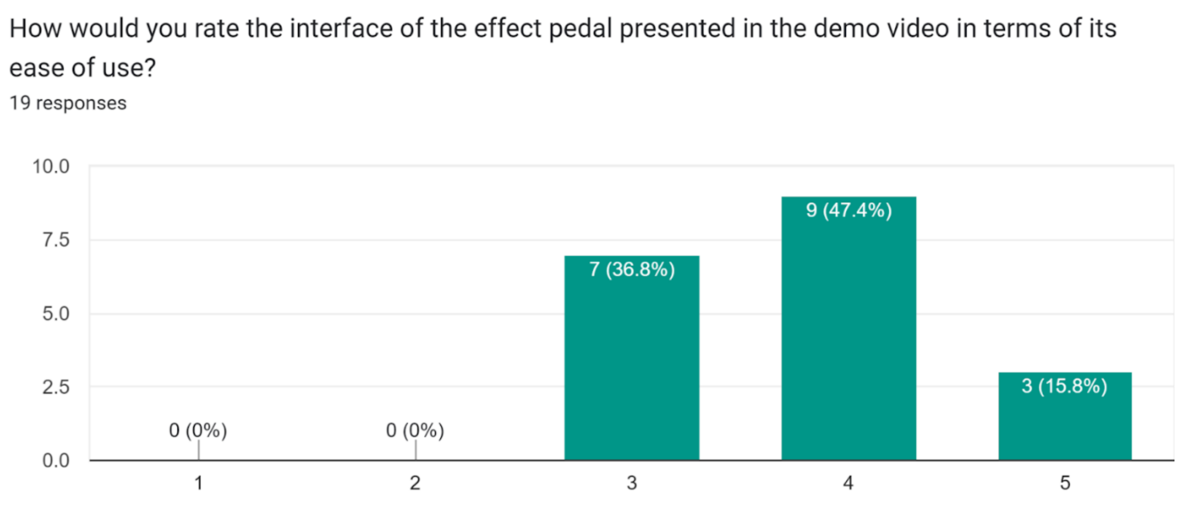


**Figure 78:** Survey Question 12.

**Question 13** (see Figure 79)

Seven participants evaluated the degree of uniqueness of the effects featured on the device as unique, with an additional two deeming them very unique. In contrast, seven participants rate the effects as average, while three participants don't consider them unique.

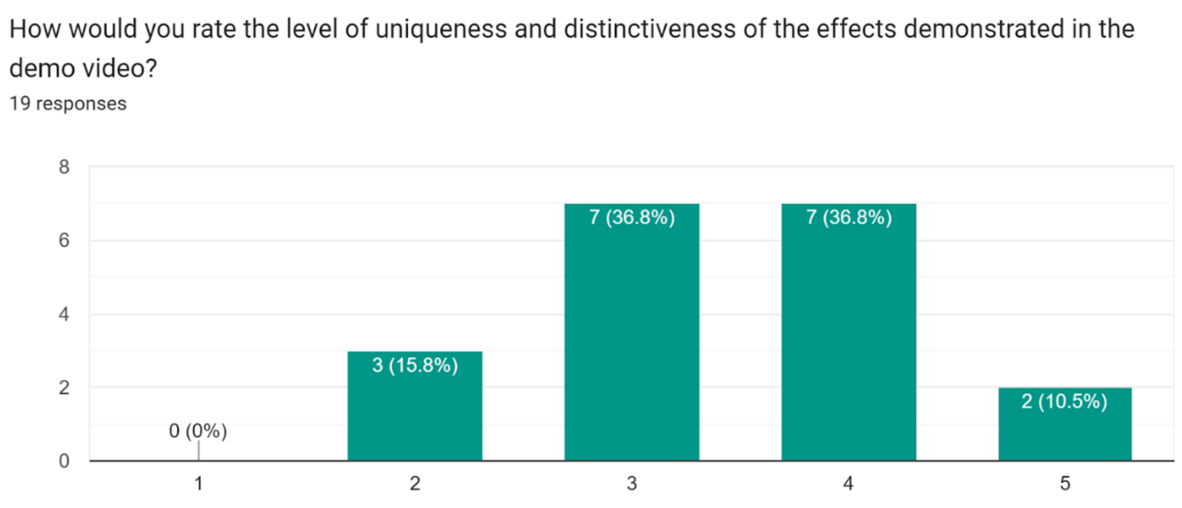


**Figure 79:** Survey Question 13.

**Question 14** (see Figure 80)

Fifteen out of the nineteen participants identified the Scanner Vibrato simulator as the most standout among the three effects, while four participants regarded the Tape Delay simulator as the most distinctive. None of the respondents perceive the Freeverb implementation as a uniquely distinct audio effect.

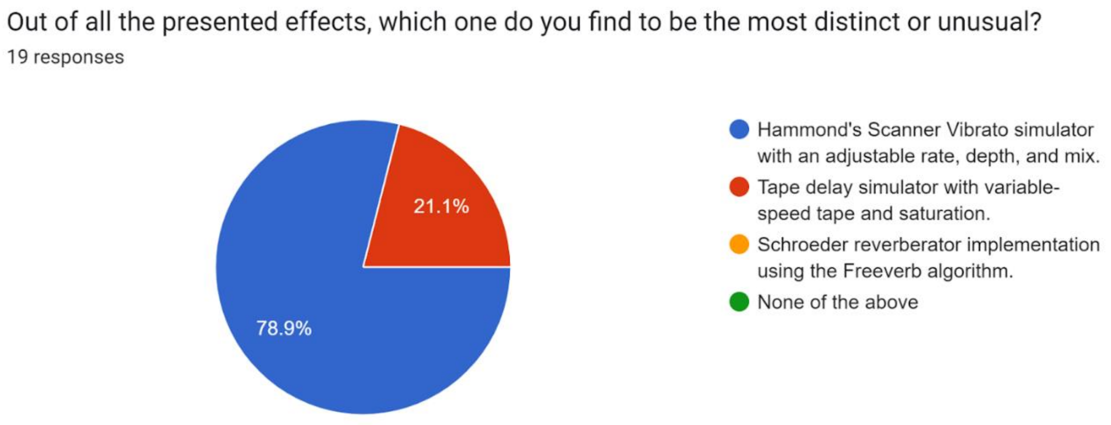


**Figure 80:** Survey Question 14.

**Question 15** (see Figure 81)

Eighteen participants rate the quality and fidelity of the effects generated by the pedal as either high or very high, whereas only one respondent regards it as acceptable.

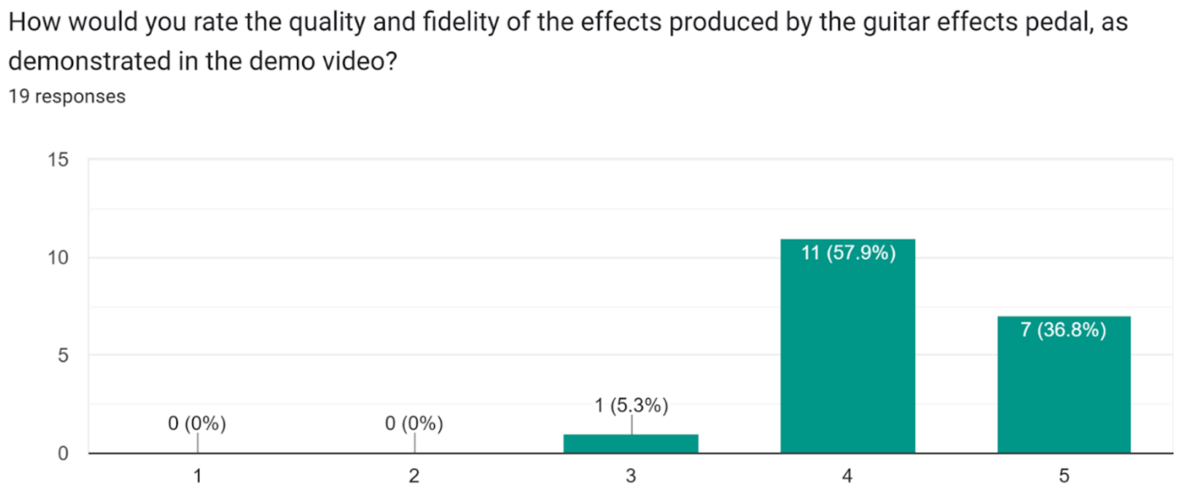


**Figure 81:** Survey Question 15.

**Question 16** (see Figure 82)

Regarding artistic expression, fourteen participants believe that the effects produced by the pedal effectively enhance the guitar's sound and contribute positively to musical performance. Meanwhile, four participants find the enhancement to be very effective, and one participant rates it as average.

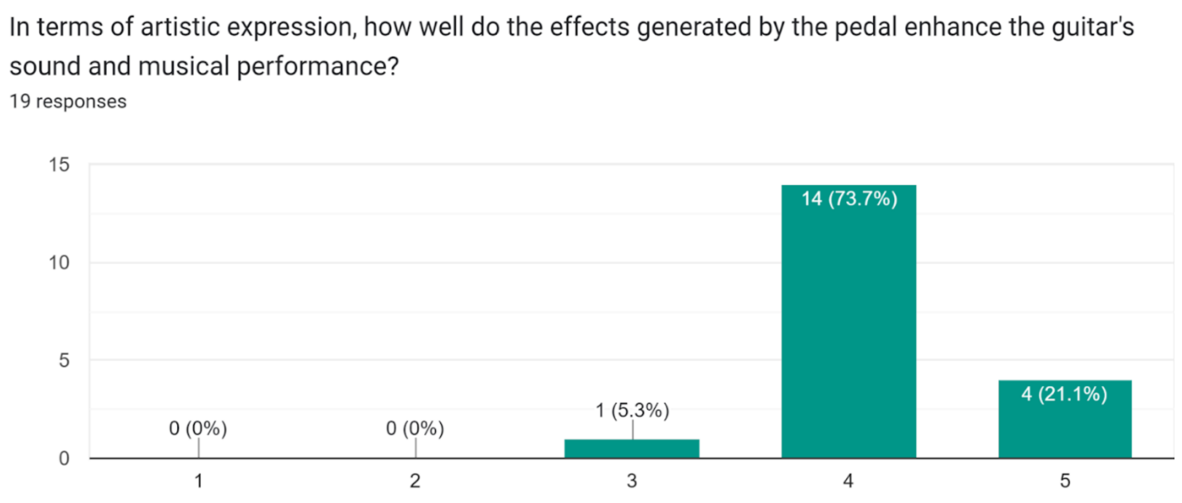


**Figure 82:** Survey Question 16.

**Question 17** (see Figure 83)

Fifteen out of the nineteen participants expressed an intention to integrate this effect pedal into their setup. Four participants are uncertain about this decision, but none of the participants would entirely dismiss the possibility.

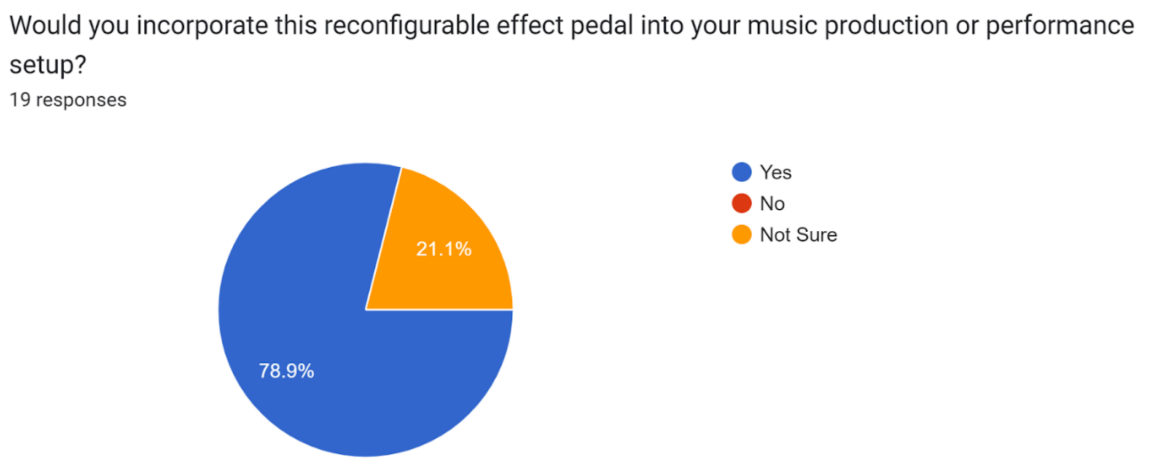


**Figure 83:** Survey Question 17.

**Question 18** (see Figure 84)

Sixteen out of the nineteen participants exhibited an interest in creating and utilizing their own custom effects on the unit, with three participants being undecided and no participant ruling out the possibility.

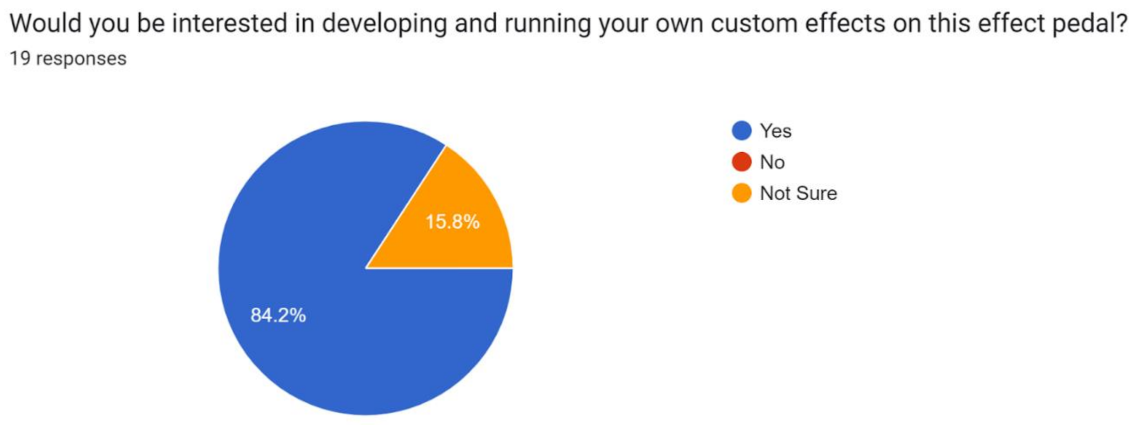


**Figure 84:** Survey Question 18.

### 9.1.3 Section 3

**Question 19** (see Figure 85)

Thirteen out of the nineteen participants are acquainted with or highly knowledgeable about the concept of open-source hardware and software, while six have either moderate or limited understanding of the subject.

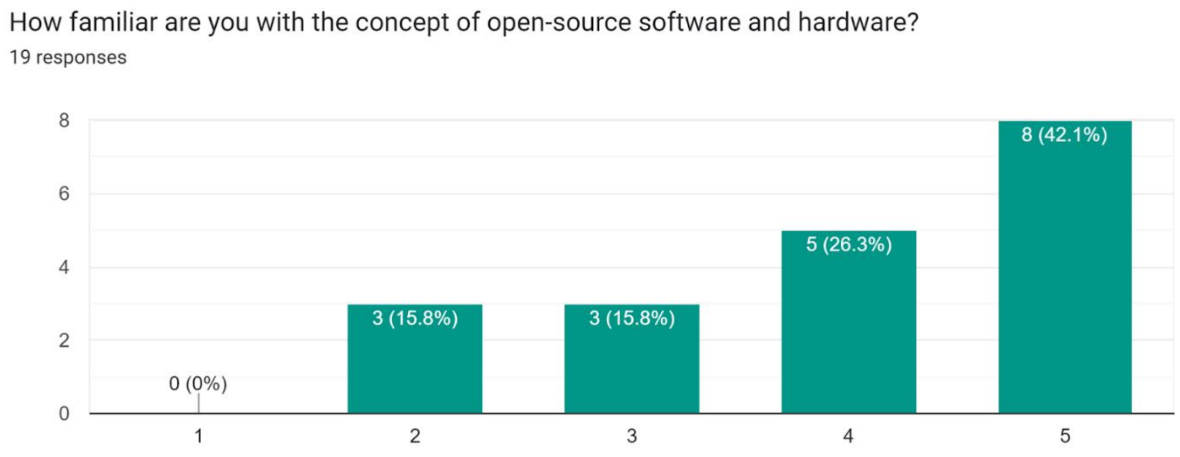


**Figure 85:** Survey Question 19.

**Question 20** (see Figure 86)

Twelve out of the nineteen participants have already utilized open-source software or hardware in their music production or performance setup, whereas seven respondents have not yet engaged with this type of technology.

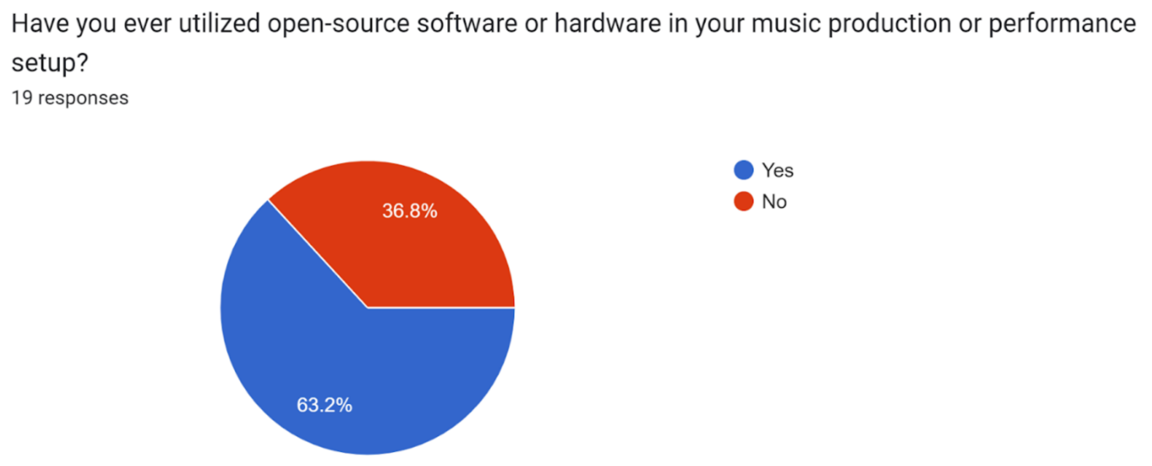


**Figure 86:** Survey Question 20.

**Question 21** (see Figure 87)

Eighteen out of the nineteen respondents regard access to comprehensive documentation and support as important or very important when utilizing an open-source guitar effect pedal, with only one respondent assigning it a moderate level of importance.

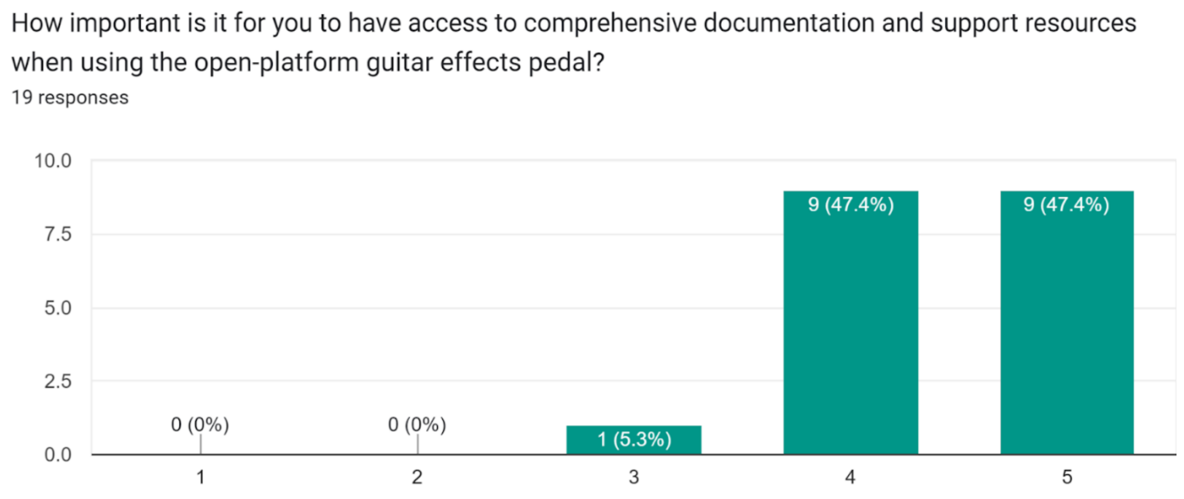


**Figure 87:** Survey Question 21.

**Question 22** (see Figure 88)

Ten out of the nineteen participants view the capability to collaborate and share patches with others as important or highly important. Meanwhile, eight participants consider it to have a moderate level of importance, and one participant regards it as less significant.

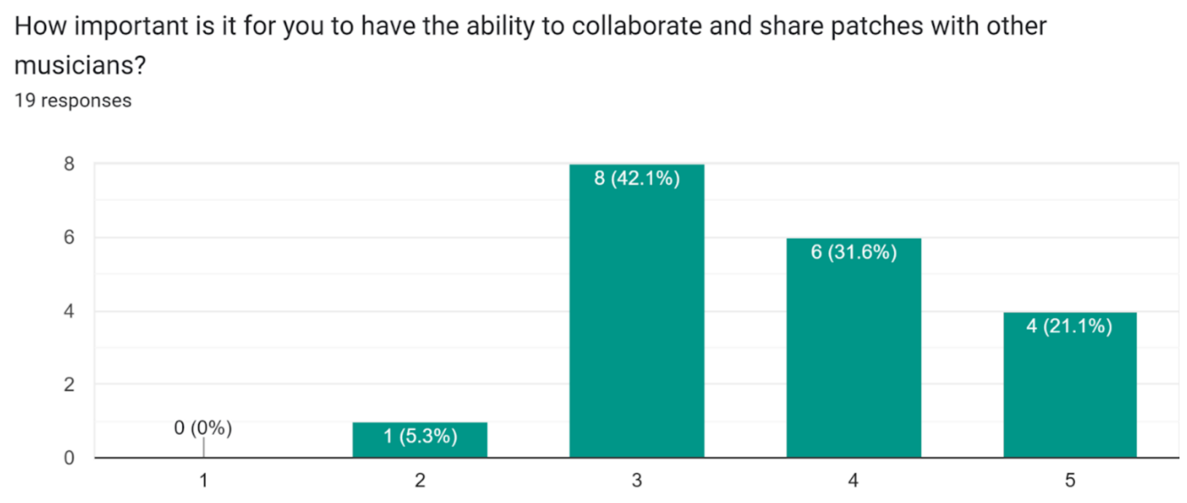


**Figure 88:** Survey Question 22.

**Question 23** (see Figure 89)

Eighteen out of the nineteen participants regard access to a supportive community or resources as important or very important when utilizing an open-source guitar effect pedal, with only one respondent assigning it a moderate level of importance.

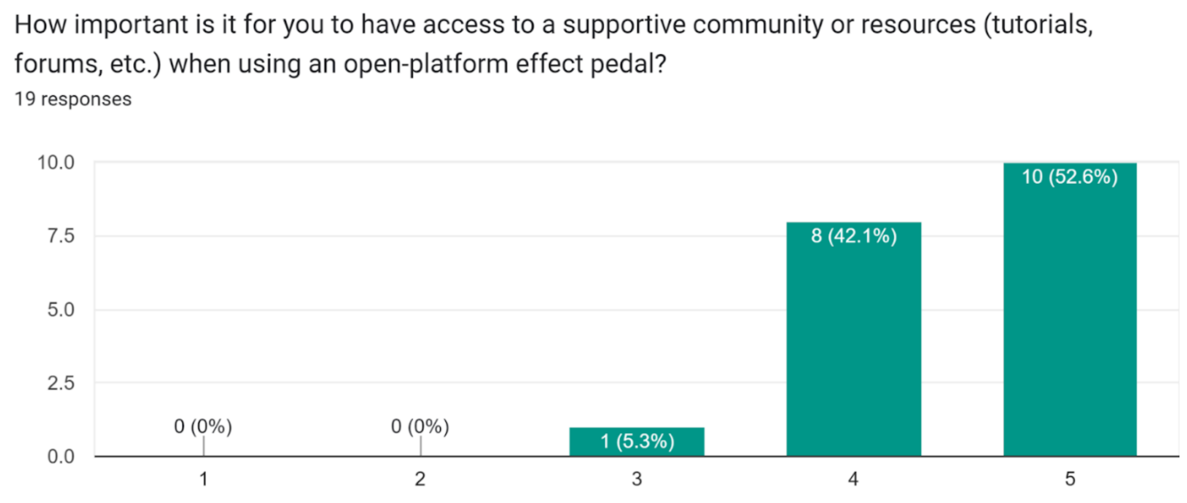


**Figure 89:** Survey Question 23.

**Question 24** (see Figure 90)

Eight out of the nineteen respondents consider contributing to the continued advancement of open-source guitar effects. Another eight participants hold a moderate level of motivation for such contributions, while three participants are either unlikely or not at all likely to contribute.

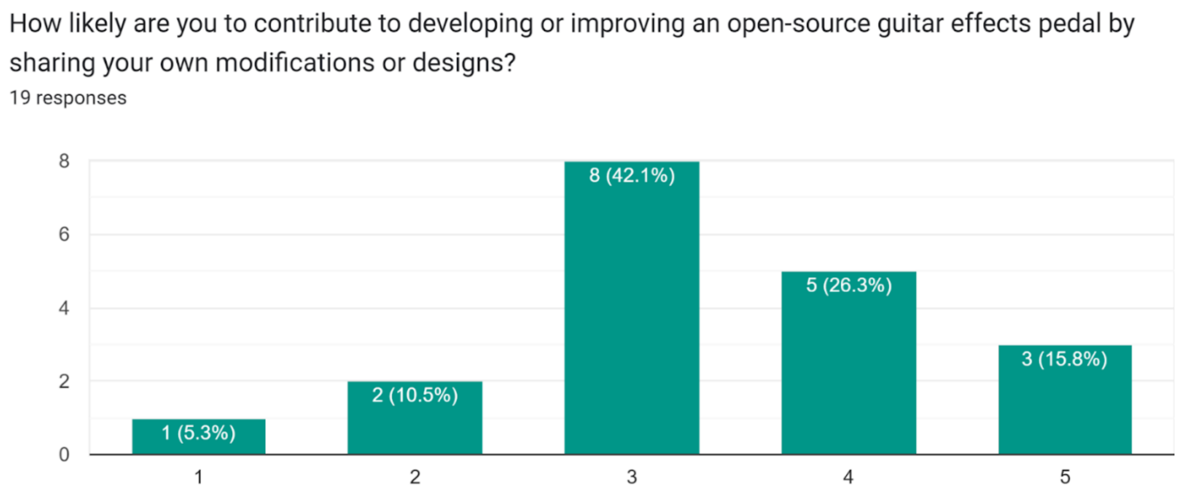


**Figure 90:** Survey Question 24.

**Question 25** (see Figure 91)

Fifteen out of the nineteen participants hold the viewpoint that the open nature of guitar effect pedals promotes innovation and nurtures a sense of collaboration within the music community. Four of the respondents expressed uncertainty on this matter, while none disagreed with the notion.

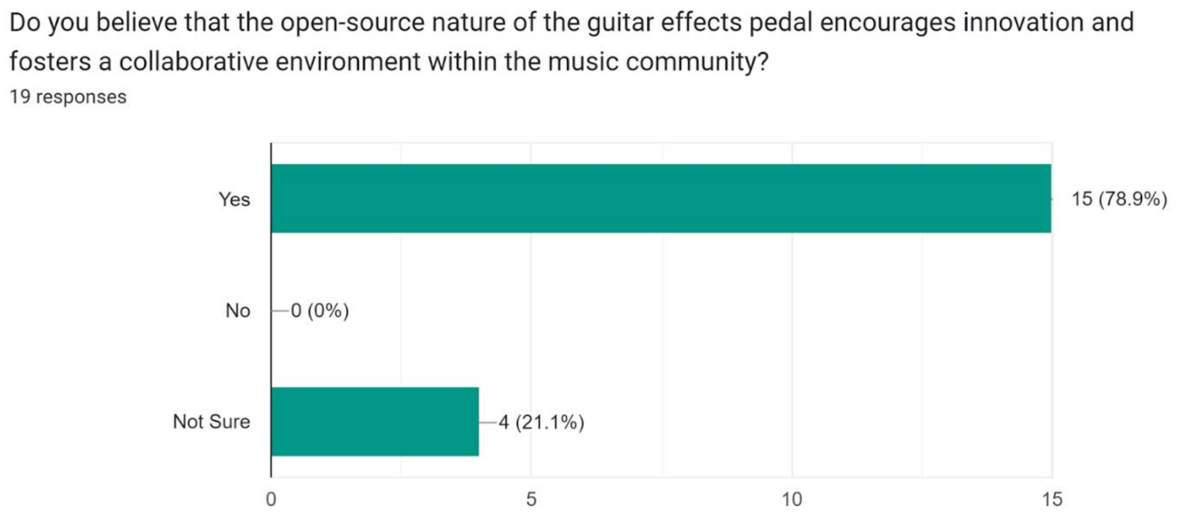


**Figure 91:** Survey Question 25.

**Question 26** (see Figure 92)

Fifteen out of the nineteen participants would place their trust in investing in a guitar effect pedal that relies on open-source technologies. Four participants are uncertain about this decision, but none rules out the possibility.

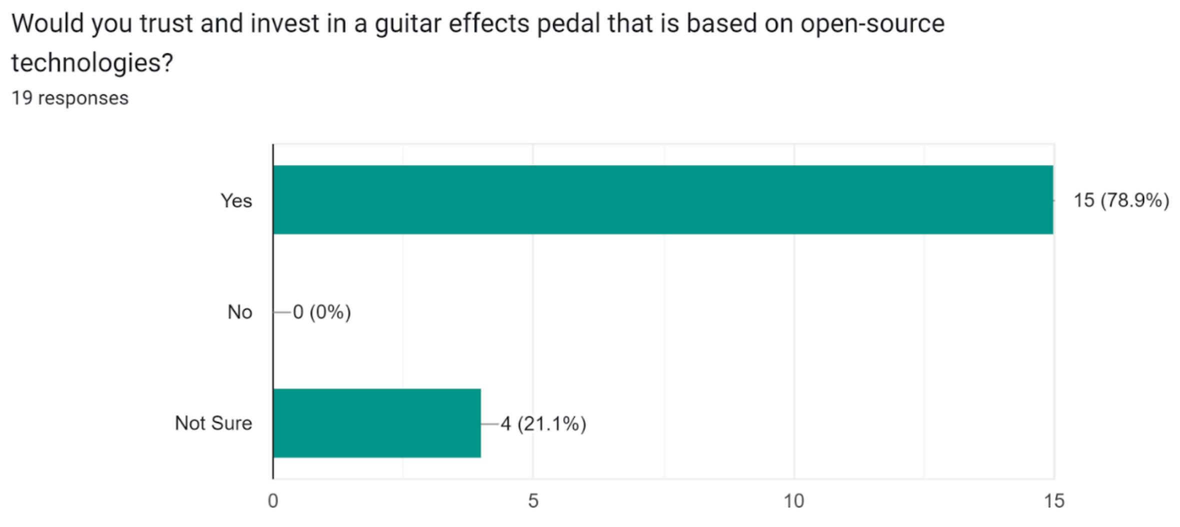


**Figure 92:** Survey Question 26.

**Question 27** (see Figure 93)

Five out of the nineteen participants have concerns or reservations regarding the reliability or stability of open-platform effect pedals when compared to traditional ones. In contrast, fourteen participants were either uncertain or had no reservations about this issue.

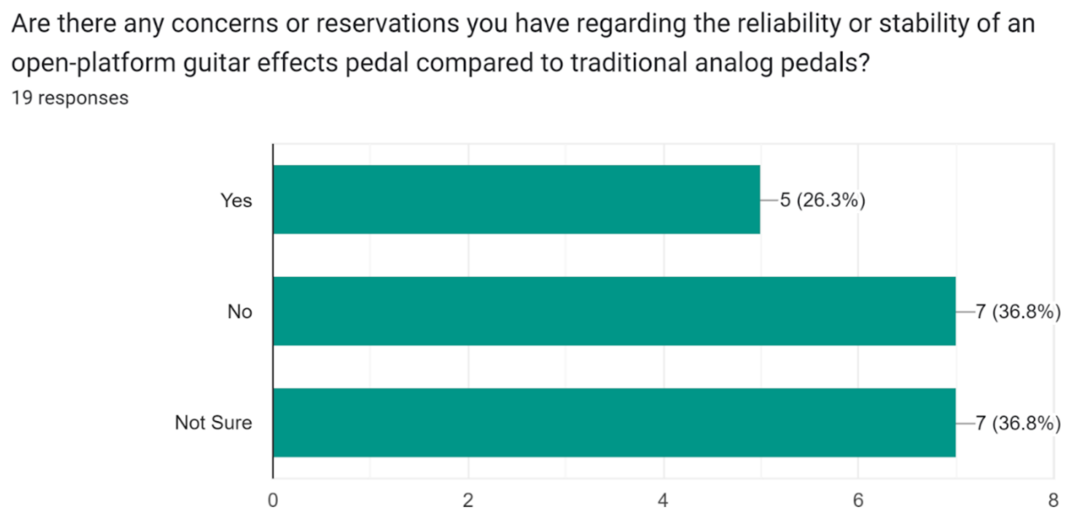


**Figure 93:** Survey Question 27.

## 9.2 Discussion

The survey findings reveal that many participants have advanced levels of familiarity with guitar effect pedals, but only a small number have experience with reconfigurable effect pedals. Despite this lack of experience, many emphasized the importance of customization, easy programming, and personalization in effect pedals, reflecting a desire for hands-on sound crafting and aligning effects with their musical style. A notable outcome showed a unanimous belief among participants, stating that reconfigurable effect pedals could inspire novel and inventive approaches to guitar playing and composition.

Furthermore, the survey outcome highlights a generally positive reception of the device's practical and artistic applications, with a significant number of respondents expressing an intention to integrate the effect pedal into their setup. While the effects present on the device are not perceived as outstanding or unique, the Scanner Vibrato simulator emerges as a standout among participants, with a significant majority identifying it as the most distinctive effect. Most participants found the device's interface easy to use, but a few considered it average in terms of user-friendliness. This mixed response could indicate potential areas for improvement in terms of usability. However, the quality and fidelity of the effects generated by the pedal were rated as high, which is a promising indicator of the device's technical performance.

The survey also revealed a favorable perception of the open nature of guitar effect pedals, fostering innovation and collaboration within the music community. Most participants recognized the potential of open-platform technology to inspire new approaches to music creation and desired customization and personalization. Respondents showed a strong inclination towards customization, collaboration, and community engagement, suggesting a growing recognition and impact of open-source solutions in the realm of music production and performance. This shared optimism points towards a collective recognition of the transformative potential of open-platform technology in shaping the future of music creation.

# 10. Conclusion

This thesis demonstrates the potential of utilizing embedded computer technologies to replicate rare or outdated audio effects through the development of an open-platform guitar effects pedal using Bela's embedded computing framework. The report thoroughly documents the device's construction, design, and development process of the hardware, as well as the establishment of a comprehensive software framework using Pure Data that enables guitarists to create and apply distinct audio effects on dedicated audio hardware. This framework not only allows for creating multi-effect setups but also pays tribute to historical audio effects like Hammond's Scanner Vibrato unit.

Crucial factors such as latency, hardware design, and software adaptability have been addressed, ensuring that the proposed pedal meets performance and compatibility expectations. The latency test results indicate that the pedal not only meets but exceeds the latency standards required for exceptional audio quality and responsiveness. Additionally, cross-platform test results indicate that the platform is capable of integrating and running audio effect patches designed for similar platforms, allowing for the blending of patches from various sources, further validating its success.

An online survey conducted as part of the study illustrates the positive reception of the device's practical and artistic applications. Participants express a keen interest in customization, easy programming, and personalization, emphasizing the potential for innovative sound crafting and musical expression. Additionally, the survey underscores the significance of open design in fostering collaboration and innovation within the music community.

## References

[1] T-Rex Effects. “Replicator Tape Echo.” → T-REX EFFECTS ← Pedals for guitar and bass players! Accessed May 15, 2023. https://www.t-rex-effects.com/replicator.

[2] Bilbao, Stefan. *Electromechanical Effects*, n.d., 1. https://www.researchgate.net/publication/265203617_Electromechanical_Effects.

[3] Holmes, Benjamin Charles. “Guitar Effects-Pedal Emulation and Identification.” *Queen’s University Belfast Research Portal*, 2019. https://pureadmin.qub.ac.uk/ws/portalfiles/portal/169350312/main.pdf.

[4] Martinic. “THE HISTORY OF THE SCANNER VIBRATO.” Martinic.com. Accessed May 15, 2023. https://www.martinic.com/en/products/scanner-vibrato/history.

[5] Martinic. “SCANNER VIBRATO.” Martinic.com. Accessed May 1, 2023. https://www.martinic.com/en/products/scanner-vibrato.

[6] Apple Inc. “Scanner Vibrato Effect in Logic pro for Mac.” Apple Support. Accessed May 1, 2023. https://support.apple.com/guide/logicpro/scanner-vibrato-controls-lgcef266e44b/mac.

[7] Ritsch, Winfried, Fabio Kaiser, and Marian Weger. *Designing simulacra or the electronic replication of a mechanical instrument,* January 2012.

[8] Sandoz, Devin. “Simulation, Simulacrum (1).” The University of Chicago, 2003. https://csmt.uchicago.edu/glossary2004/simulationsimulacrum.htm.

[9] Ritsch, Winfried, Fabio Kaiser, and Marian Weger. *Designing simulacra or the electronic replication of a mechanical instrument,* January 2012.

[10] Giulio, Moro, Bin Astrid, Jack H. Robert, Heinrichs Christian, and Andrew P. McPherson. *Making High-Performance Embedded Instruments with Bela and Pure Data*, June 29, 2016, 1–5.

[11] Ljungberg, Jan. “Open Source Movements as a Model for Organizing.” *European Journal of Information Systems*, December 2000, 208–16. https://doi.org/DOI: 10.1057/palgrave/ejis/3000373.

[12] Boisseau, Étienne, Jean-François Omhover, and Carole Bouchard. “Open-Design: A State of the Art Review.” *Design Science* 4 (2018): 1–3. https://doi.org/10.1017/dsj.2017.25.

[13] Morreale, Fabio, Giulio Moro, Alan Chamberlain, Steve Benford, and Andrew P. McPherson. “Building a Maker Community around an Open Hardware Platform.” *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017. https://doi.org/10.1145/3025453.3026056.

[14] Harrison, Michael. “What Is Open Hardware?” Opensource.com. Accessed May 1, 2023. https://opensource.com/resources/what-open-hardware.

[15] “Definition (English).” Open Source Hardware Association, May 14, 2021. https://www.oshwa.org/definition/.

[16] Michon, Romain, Yann Orlarey, Stéphane Letz, and Dominique Fober. “Real Time Audio Digital Signal Processing With Faust and the Teensy.” *Sound and Music Computing Conference (SMC-19)*, May 2019.

[17] Barrass, Tim. *MOZZI : INTERACTIVE SOUND SYNTHESIS ON THE OPEN SOURCE ARDUINO MICROPROCESSOR*, January 2013.

[18] Kevin, “Arduino FM MIDI Synthesis with Mozzi,” Simple DIY Electronic Music Projects, May 17, 2021, https://diyelectromusic.wordpress.com/2020/08/18/arduino-fm-midi-synthesis-with-mozzi/.

[19] McPherson, Andrew P., and Victor Zappi. “An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black.” *Audio Engineering Society Convention 138*, 2015. https://www.aes.org/e-lib/browse.cfm?elib=17755.

[20] Michon, Romain, Yann Orlarey, Stéphane Letz, Dominique Fober, and Dirk Roosenburg. “Embedded Real-Time Audio Signal Processing With Faust.” *International Faust Conference (IFC-20)*, December 2020. https://doi.org/hal-03124896.

[21] Jacobs, Ben. “RPI OS, PureData, Audio Codec Subsystem: Details.” Hackaday.io, April 29, 2020. https://hackaday.io/project/171186-the-guitar-pedal-project-multi-effect-processor/log/176686-rpi-os-puredata-audio-codec-subsystem.

[22] Ibid.

[23] Vignati, Luca, Stefano Zambon, and Luca Turchet. “A Comparison of Real-Time Linux-Based Architectures for Embedded Musical Applications.” *Journal of the Audio Engineering Society* 70, no. 1/2 (2021): 83–93. https://doi.org/10.17743/jaes.2021.0052.

[24] Webster, Thomas, Guillaume LeNost, and Martin Klang. “The OWL: An Open Source, Programmable Stage  Effects Pedal.” *KES Transactions on Innovation in Music* 1 (2013): 147–57.

[25] Webster, Thomas, Guillaume LeNost, and Martin Klang. “The OWL Programmable Stage Effects Pedal: Revising the Concept of the Onstage Computer for Live Music Performance.” *Proceedings of the International Conference on New Interfaces for Musical Expression*, June 1, 2014, 621–24. https://doi.org/DOI 10.5281/zenodo.1178979.

[26] “Owl Pedal Mkii.” Rebel Technology, June 3, 2023. https://www.rebeltech.org/product/owl-pedal-mkii/.

[27] Wakefield, Graham. “A Streamlined Workflow from Max/Gen~ to Modular Hardware.” *NIME 2021*, April 29, 2021. https://doi.org/10.21428/92fbeb44.e32fde90.

[28] Ibid., 6.

[29] “Pod - Electro-Smith.” Electro-Smith. Accessed July 1, 2023. https://www.electro-smith.com/daisy/pod.

[30] Berdahl, Edgar, and Spencer Salazar. “Making Embedded NIMEs with Satellite CCRMA Featuring the Raspberry Pi.” *13th International Conference on New Interfaces for Musical Expression*, 2013. https://www.nime.org/2013/program/workshops/1.pdf.

[31] Berdahl, Edgar, and Wendy Ju. “2011: Satellite Ccrma: A Musical Interaction and Sound Synthesis Platform.” *A NIME Reader*, 2017, 373–89. https://doi.org/10.1007/978-3-319-47214-0_24.

[32] Berdahl, Edgar, and Spencer Salazar. “Making Embedded NIMEs with Satellite CCRMA Featuring the Raspberry Pi.” *13th International Conference on New Interfaces for Musical Expression*, 2013. https://www.nime.org/2013/program/workshops/1.pdf.

[33] CRITTER & GUITARI. Organelle. Accessed May 1, 2023. https://www.critterandguitari.com/organelle.

[34] Ibid.

[35] Ibid.

[36] Sullivan, John, Julian Vanasse, Catherine Guastavino, and Marcelo  M. Wanderley. “Reinventing the Noisebox: Designing Embedded Instruments for Active Musicians.” *Embedded Instruments for Active Musicians.” New Interfaces for Musical Expression*, 2020, 5–10. https://doi.org/DOI:10.5281/zenodo.4813166.

[37] Tom, Ajin Jiji, Harish Jayanth Venkatesan, Ivan Franco, and Marcelo Wanderley. “Rebuilding and Reinterpreting a Digital Musical Instrument - The Sponge.” *Proceedings of the International Conference on New Interfaces for Musical Expression.*, 2019, 37–42. https://doi.org/DOI: 10.5281/zenodo.3672858.

[38] *Prynth Control Board attached to Raspberry Pi*. *Prynth Github.* Accessed May 1, 2023. https://prynth.github.io/doc/images/control_rpi.jpg.

[39] Meneses, Eduardo, Johnty Wang, Sergio Freire, and Marcelo Wanderley. “A Comparison of Open-Source Linux Frameworks for an Augmented Musical Instrument Implementation.” *New Interfaces for Musical Expression*, 2019, 222–27. https://doi.org/DOI:10.5281/zenodo.3672934.

[40] Ives, Dave. “Advanced Audio Effects with a BeagleBone Black and Bela Part 1: Getting Up and Running.” DesignSpark, April 17, 2019. https://www.rs-online.com/designspark/advanced-audio-effects-with-a-beaglebone-black-and-bela-part-1-getting-up-and-running.

[41] Moro, Giulio. “Beyond Key Velocity: Continuous Sensing for Expressive Control on the Hammond Organ and Digital Keyboards,” 2020.

[42] “Bela: An Embedded Platform for Ultra-Low Latency Interactive Audio.” Augmented Instruments Laboratory. Accessed May 1, 2023. http://instrumentslab.org/research/bela.html.

[43] Ibid., 64.

[44] Bela.io. “About.” All about Bela. Accessed May 1, 2023. https://bela.io/about.

[45] McPherson, Andrew P., and Victor Zappi. “An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black.” *Audio Engineering Society Convention 138*, 2015. https://www.aes.org/e-lib/browse.cfm?elib=17755.

[46] Bela.io. “Pure Data.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/languages/pure-data/.

[47] Meneses, Eduardo, Johnty Wang, Sergio Freire, and Marcelo Wanderley. “A Comparison of Open-Source Linux Frameworks for an Augmented Musical Instrument Implementation.” *New Interfaces for Musical Expression*, 2019, 222–27. https://doi.org/DOI:10.5281/zenodo.3672934.

[48] Bela.io. “Meet the IDE.” The Bela Knowledge Base. Accessed May 1, 2023. https://learn.bela.io/the-ide/meet-the-ide/.

[49] Ibid.

[50] Giulio, Moro, Bin Astrid, Jack  H. Robert, Heinrichs Christian, and Andrew P. McPherson. *Making High-Performance Embedded Instruments with Bela and Pure Data*, June 29, 2016, 1–5.

[51] Bela.io. “Bela Mini.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/products/bela-boards/bela-mini/#bela-vs-bela-mini.

[52] Bela.io. “About.” All about Bela. Accessed May 1, 2023. https://bela.io/about.

[53] Manzano, Chuck. *Application Note - CE Series Mechanical Encoders*, n.d.

[54] Handson Technology. *KY040 Rotary Encoder User Guide*. Accessed June 1, 2023. https://www.handsontec.com/dataspecs/module/Rotary%20Encoder.pdf.

[55] “KY-040 - Rotary Encoder Module.” Components101. Accessed June 1, 2023. https://components101.com/modules/KY-04-rotary-encoder-pinout-features-datasheet-working-application-alternative.

[56] Bela.io. “Digital Input.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/tutorials/pure-data/connecting/digital-input/.

[57] Ibid.

[58] Handson Technology. *KY040 Rotary Encoder User Guide*. Accessed June 1, 2023. https://www.handsontec.com/dataspecs/module/Rotary%20Encoder.pdf.

[59] Pete. “What’s the Difference between Latching & Momentary (Non-Latching) Footswitches?” DIY Guitar Effects Pedals, August 28, 2021. https://diyeffectspedals.com/difference-between-latching-and-momentary-footswitches/.

[60] JB Magoncia. “How RGB Leds Work and How to Control Color - Electronics Tutorials.” CircuitBread, July 7, 2019. https://www.circuitbread.com/tutorials/how-rgb-leds-work-and-how-to-control-color.

[61] “Adafruit Industries LLC 1312.” Digi-key electronics. Accessed June 1, 2023. https://www.digikey.at/de/products/detail/adafruit-industries-llc/1312/6565388?s=N4IgTCBcDaIIIBMCGAzATgVwJYBcAEAkgHYIYDOOaWApmXgDL0DCeAjAMysQC6AvkA.

[62] Bela.io. “Digital Output.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/tutorials/pure-data/connecting/digital-output/.

[63] Bela.io. “Using an OLED Screen.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/bela-techniques/using-an-oled-screen/.

[64] Bela.io. “Powering Bela.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/using-bela/bela-techniques/powering-bela/.

[65] “Bela Mini Usb Host Power Only on Usb Supply.” Bela Forum, November 2021. https://forum.bela.io/d/1166-bela-mini-usb-host-power-only-on-usb-supply/13.

[66] Bela.io. “Powering Bela.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/using-bela/bela-techniques/powering-bela/.

[67] Ibid.

[68] “Bela Mini Usb Host Power Only on Usb Supply.” Bela Forum, February 2020. https://forum.bela.io/d/1166-bela-mini-usb-host-power-only-on-usb-supply/2.

[69] “What Are Expression Pedals?” That Pedal Shop. Accessed June 1, 2023. https://www.thatpedalshop.com/pages/what-are-expression-pedals.

[70] “EX-P Universal Expression Pedal.” Kenny’s Music. Accessed June 1, 2023. https://www.kennysmusic.co.uk/m-audio-ex-p-universal-expression-pedal-p5922.

[71] Bela.io. “Analog Input.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/tutorials/pure-data/connecting/analog-input/.

[72] “How Expression Pedals Work.” All about expression pedals. Accessed June 1, 2023. http://expressionpedals.com/how-expression-pedals-work.

[73] Ibid.

[74] bmoe24x. “Expression Pedal TRS Input Requirements.” Arduino Forum, May 16, 2018. https://forum.arduino.cc/t/expression-pedal-trs-input-requirements/526113/3.

[75] Bela.io. “Bela Hardware.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/using-bela/about-bela/bela-hardware/#audio-io.

[76] “2 Audio Adapter Cables.” eu.shop.bela.io. Accessed August 9, 2023. https://eu.shop.bela.io/products/audio-adapter-cables-3.

[77] Bela.io. “Resources.” The Bela Knowledge Base. Accessed June 1, 2023. https://learn.bela.io/community/resources/.

[78] Török, Lehel. “effect_cape_for_bela_mini/Hardware /Schematic and PCB.” GitHub, 2023. https://github.com/leheltorok/effect_cape_for_bela_mini/tree/main/Hardware/Schematic%20and%20PCB.

[79] Török, Lehel. “Interactive BOM.” Dropbox, 2023. https://www.dropbox.com/s/17bvmpcd8jzs7nz/ibom.html?dl=0.

[80] Török, Lehel. “Effect_cape_for_bela_mini/Case.” GitHub, 2023. https://github.com/leheltorok/effect_cape_for_bela_mini/tree/main/Case.

[81] Török, Lehel. “Effect_cape_for_bela_mini/Projects /Delay_Chain.” GitHub, 2023. https://github.com/leheltorok/effect_cape_for_bela_mini/tree/main/Projects/Delay_Chain.

[82] Ritsch, Winfried. “Does Pure Data Dream of Electric Violins?” Chapter. In *Bang: Pure Data*, 1st ed., 11–18. Hofheim, Germany: Wolke Verlag, 2006.

[83] Bela.io. “Pure Data.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/languages/pure-data/.

[84] Ibid.

[85] Brinkmann, Peter. “Libpd: Pure Data Embeddable Audio Synthesis Library.” GitHub. Accessed July 1, 2023. https://github.com/libpd/libpd.

[86] Bela.io. “Pure Data.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/languages/pure-data/.

[87] Farnell, Andy. “Pure Data Essentials.” Chapter. In *Designing Sound*, 219–36. Cambridge, MA, England: MIT Press, 2010.

[88] Barta, Andrew. “The Truth about True Bypass.” Tech 21 NYC, October 30, 2006.

https://www.tech21nyc.com/the-truth-about-true-bypass/.

[89] Bela.io. “Pure Data.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/languages/pure-data/.

[90] Bela.io. “Combining Pure Data and C++.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/tutorials/pure-data/advanced/custom-render/.

[91] Török, Lehel. “Effect_cape_for_bela_mini/Projects/Delay_Chain /Render.Cpp.” GitHub, 2023.
https://github.com/leheltorok/effect_cape_for_bela_mini/blob/main/Projects/Delay_Chain/render.cpp.

[92] Farnell, Andy. “Abstraction.” Chapter. In *Designing Sound*, 192-203. Cambridge, MA, England: MIT Press, 2010.

[93] Bela.io. “Pure Data.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/languages/pure-data/.

[94] Ibid.

[95] Giuliomoro. “O2O.” GitHub. Accessed July 1, 2023.
https://github.com/giuliomoro/O2O.

[96] “OSC Messages to and from Pd Lists.” Pure data - [oscparse]. Accessed July 1, 2023. https://pd.iem.sh/objects/oscparse/.

[97] Bela.io. “Using an OLED Screen.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/bela-techniques/using-an-oled-screen/.

[98] Bela.io. “Running a Program as a Service.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/bela-techniques/running-a-program-as-a-service/.

[99] Bela.io. “Crafting GUIs.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/the-ide/crafting-guis/.

[100] Ibid.

[101] Ibid

[102] Ibid.

[103] Bela.io. “Connecting Bela to Wifi.” Connecting Bela to Wifi - The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/bela-techniques/connecting-to-wifi/.

[104] Vorkoetter, Stefan. “Overhauling and Improving the Hammond M-100 Series Vibrato System.” stefanv.com, January 24, 2009. http://www.stefanv.com/electronics/hammond_vibrato_mod.html.

[105] “Service Manual - the Hammond Vibrato.” Benton Electronics, April 12, 2013. https://bentonelectronics.com/service-manual-the-hammond-vibrato/.

[106] “Vibrato Scanner.” Electronic Music Wiki. Accessed July 1, 2023. https://electronicmusic.fandom.com/wiki/Vibrato_scanner.

[107] Werner, Kurt  James, Ross W. Dunkel, and François G. Germain. “A Computational Model of the Hammond Organ Vibrato/Chorus Using Wave Digital Filters.” *Proceedings of the 19th International Conference on Digital Audio Effects (DAFx-16)*, September 5, 2016, 271–78.

[108] Apple Inc. “Scanner Vibrato Effect in Logic pro for Mac.” Apple Support. Accessed May 1, 2023. https://support.apple.com/guide/logicpro/scanner-vibrato-controls-lgcef266e44b/mac.

[109] “Bela/Examples/Salt/Tape-Delay.” GitHub. Accessed July 1, 2023. https://github.com/BelaPlatform/Bela/tree/master/examples/Salt/tape-delay.

[110] “Freeverb: Physical Audio Signal Processing.” DSP. Accessed July 1, 2023. https://www.dsprelated.com/freebooks/pasp/Freeverb.html.

[111] Katjav. “Freeverb in Vanilla PD.” PURE DATA forum~, August 12, 2020. https://forum.pdpatchrepo.info/topic/6247/freeverb-in-vanilla-pd.

[112] Bela.io. “Understanding Real Time.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/using-bela/about-bela/understanding-real-time/.

[113] Ibid.

[114] Bela.io. “Block-Based Processing.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/tutorials/c-plus-plus-for-real-time-audio-programming/block-based-processing/.

[115] Bela.io. “Latency.” The Bela Knowledge Base. Accessed July 1, 2023. https://learn.bela.io/tutorials/c-plus-plus-for-real-time-audio-programming/latency/.

[116] Ibid.

[117] Wessel, David, and Matthew Wright. "Problems and Prospects for Intimate Musical Control of Computers." *Computer Music Journal* 26, no. 3 (2002): 11–22. https://doi.org/10.1162/014892602320582945.

[118] Bela.io. "Bela-Online-Course/Lectures/Lecture-10 /Code-Examples." GitHub. Accessed July 1, 2023. https://github.com/BelaPlatform/bela-online-course/tree/master/lectures/lecture-10/code-examples.

[119] "Let's Create an Effects Pedal/Unit With Daisy Pod!," *YouTube*. Sound Simulator, 2022. https://www.youtube.com/watch?v=KTM8pHN3_mM&ab_channel=SoundSimulator.

[120] "Pod - Electro-Smith." Electro-Smith. Accessed July 1, 2023. https://www.electro-smith.com/daisy/pod.

[121] JoseFuzzNo. "Owl Patch Library." Rebel Technology, July 5, 2021. https://www.rebeltech.org/patch-library/patch/Filtremolo.

[122] "Owl Pedal Mkii." Rebel Technology, June 3, 2023. https://www.rebeltech.org/product/owl-pedal-mkii/.

[123] CRITTER & GUITARI. "LFO Delay." Critter & Guitari, December 5, 2017. https://www.critterandguitari.com/organelle-patches/lfo-delay#.

[124] CRITTER & GUITARI. Organelle. Accessed May 1, 2023. https://www.critterandguitari.com/organelle.

[125] Massat, Pierre. "Making a Looper with Pure Data." Guitar Extended, August 5, 2013. https://guitarextended.wordpress.com/2013/08/05/making-a-looper-with-pure-data/.

[126] Massat, Pierre. "Raspberry Pi Multi-Effects : Overview of the Setup." Guitar Extended, January 31, 2013. https://guitarextended.wordpress.com/2013/01/31/raspberry-pi-multi-effects-overview-of-the-setup/.

[127] Török, Lehel. "Effect Cape for Bela Mini (Demo)." YouTube, July 18, 2023. https://youtu.be/DcWgzVZQZMs.