Final Report

Team:
sddec19-15

Members:
Justin Shaver, Thomas Frye,
Will Pigg, Chandler Davis,
Daniel Bohlke, and Caleb Hendrickson

1. Final Project Design

Our design is for a guitar multi-effect pedal box with an LCD user interface and hardware controls, powered by a standard 9V wall plug Our box connects to a guitar via ¼ inch jack, takes in audio from the guitar, then applies effects on the audio based on selections on the device. It then sends that audio to a broadcast device. We designed our device in this fashion in order to allow the guitarist a selection of sounds without using multiple effect pedals.

Many guitar pedals only feature one effect. Given that these pedals are expensive, it is not very cost efficient for a guitarist to gain access to a variety of sounds by purchasing a variety of pedals. Our device includes 10 programmable effects that users can switch in and out of application to the original sound giving users easy access to a variety of effects.

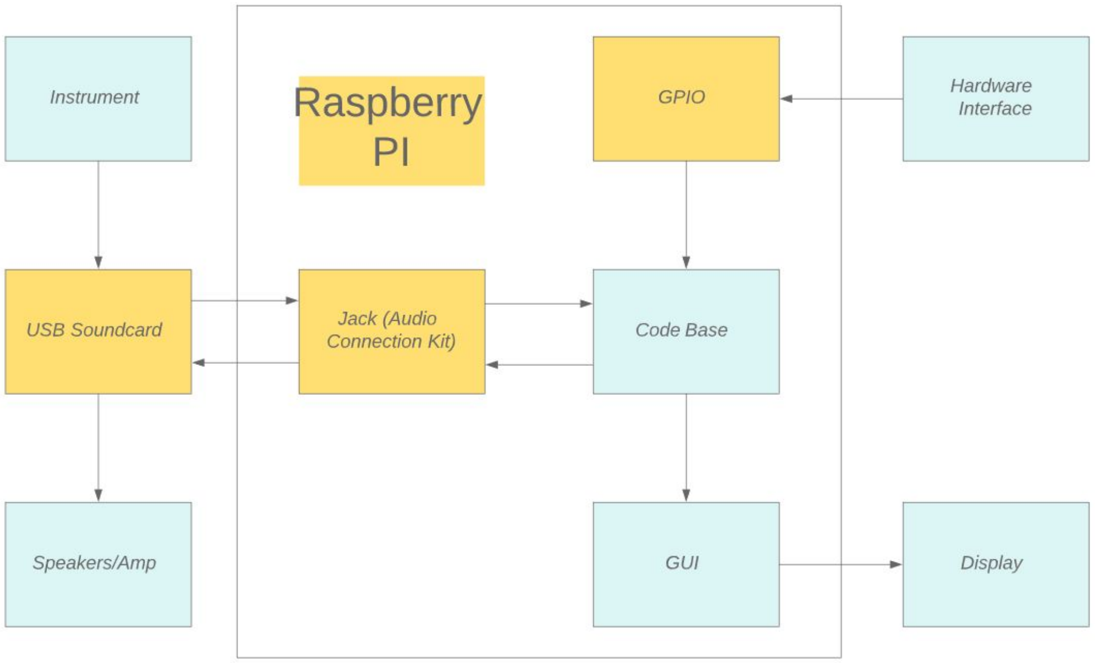
The Cyren provides a descriptive and easy to use interface. It can be used by beginners, so it includes an LCD gui interface along with the hardware interface. The LCD interface helps users inexperienced with audio effects understand what effects are being applied and how much of an effect(current effect level compared to max level) is being applied to the sound. Overall, giving users a smaller learning curve when first using the device.

We also wanted to make our design very durable. We made the device durable by designing the enclosure to be of a triangular prism shape and by deciding to build this enclosure using solid wood. It is likely to be much more durable than comparable devices in the market due to the triangular shape and solid wood material consistency.

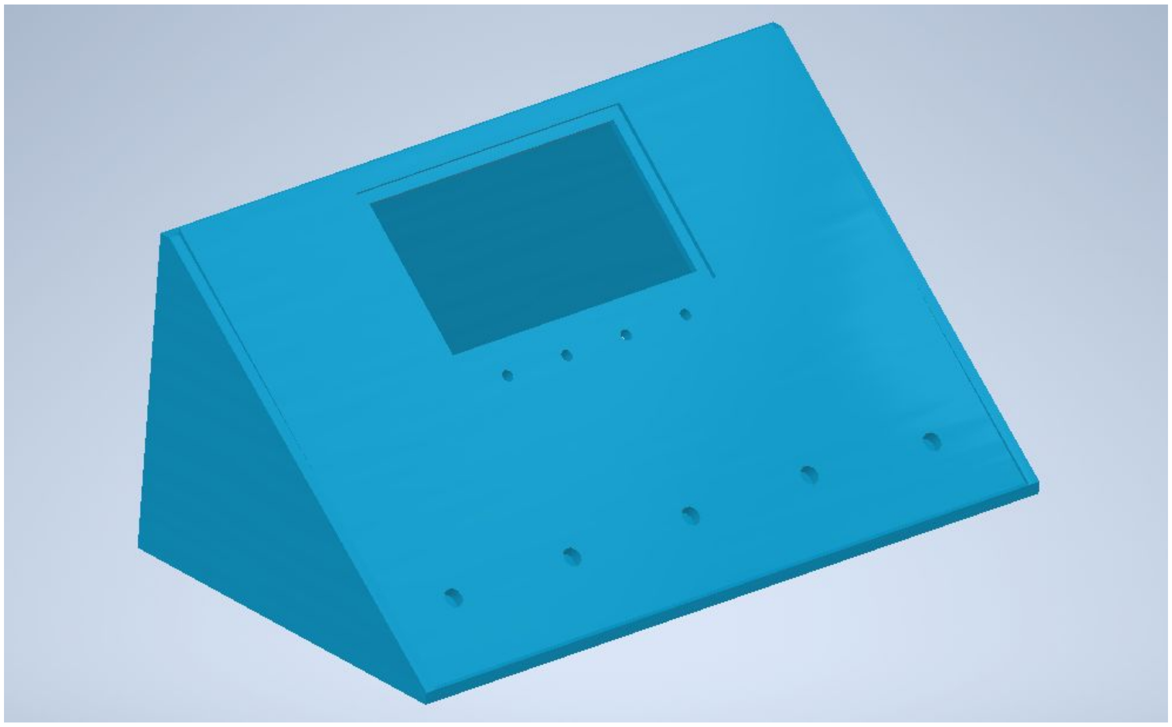
Light technical explanation of how the device works:

-Our device is powered by a 9V standard wall plug.
-Our design takes input from a guitar through ¼ inch jack in a usb interface,
-The interface transmits this via usb to the pi
-The jack utility establishes an input port, an output port, and determines the capture ports and broadcast ports. Once these are established, Jack listens to the data coming in from the capture port and copies it to the input port.
-Once the data has been copied to the input port, the data is available for alteration. The incoming sound is transformed in real time, and the characteristics of the transformation are determined by the buttons/parameters that are selected/adjusted on the hardware/gui.
-The transformed data is copied to the output port and through jack, transmitted back to the usb interface and guided to the broadcast ports(ie. Speakers, amp, etc.)

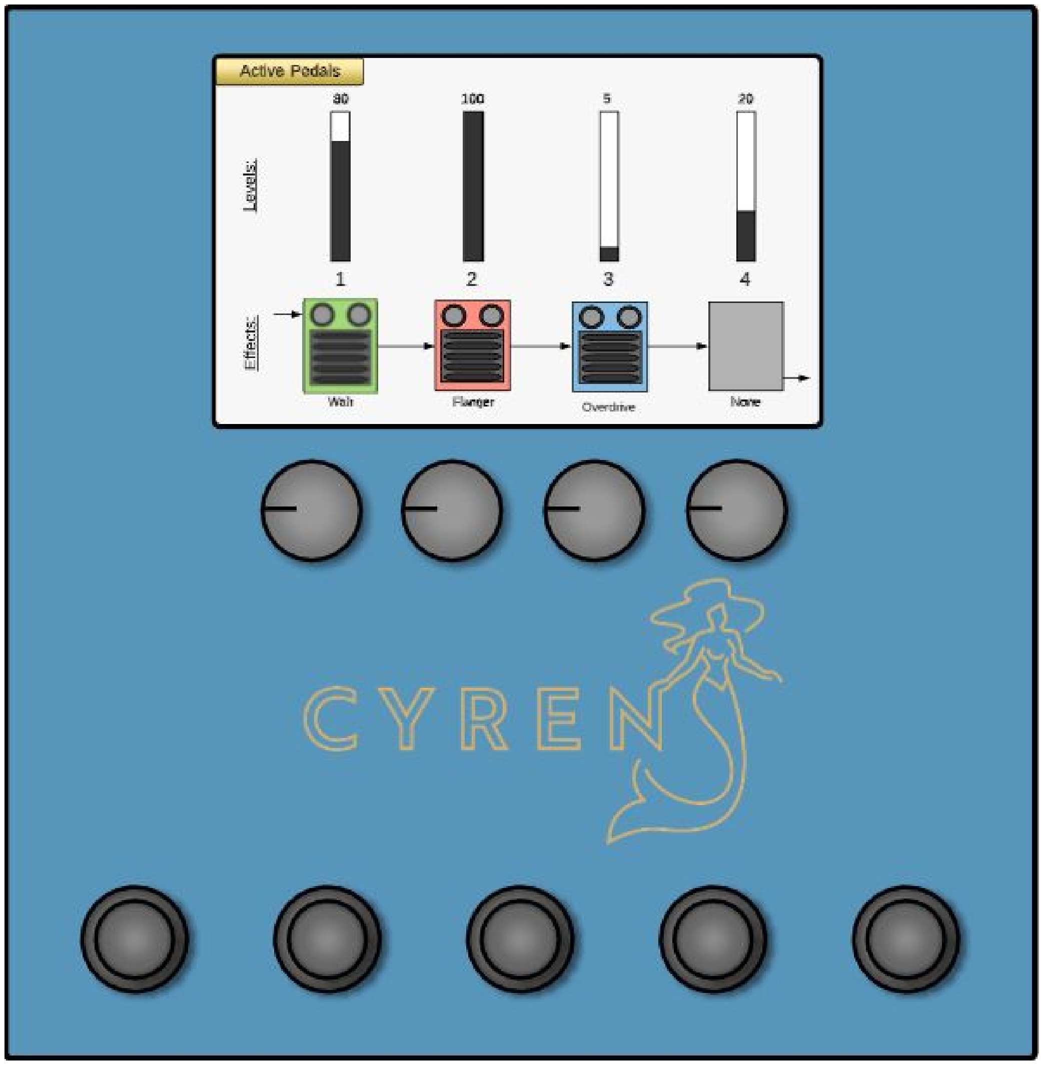
High Level Block Diagram



Enclosure Design



GUI Design



2. Implementation Details

Effects

The algorithms used for the audio effects used in this project were gathered from the book, Audio Effects: Theory, Implementation, and Application by Andrew McPherson and Joshua Reiss. As well as Digital Audio Effects, a document from the Stanford Center for Computer Research in Music and Acoustics. As well as a single effect approved by Analog Devices website. These effects were first tested for correctness in the test environment, and then integrated into the Jack simple client code. The simple client was a piece of example code that we found through the Jack website that demonstrated how to use Jack’s core functionality, which is reading audio data from an input port, and then copying the data to an output port where the data can be broadcast. Once we had this piece of code working, we integrated our effects one by one into this code, and added code so that the user can select an effect based on a keypress of the key associated with that effect.

Hardware/Electrical
Cyren includes a hardware interface that acts as one of the primary ways the user can interact with the device. The hardware interface features 5 stomp switches, 4 rotary encoders with push buttons, rear I/O and an LCD screen all connected to the Raspberry Pi’s GPIO bus. The stomp switches are wired using a pull down circuit so a signal is only sent to the Pi when one is pressed. The rotary encoders, requiring many electrical connections, were soldered onto circuit board and the connections were routed to a series of labeled pins to make it easier to wire the internals of the device. All of the hardware components are powered off of the 5V and 3.3V power pins on the Pi’s GPIO bus. The Pi will also be connected to a USB sound card that will handle the analog to digital conversion. Our prototype is using an audio interface, but any USB sound card would be work as long as it is compatible with the Linux ALSA driver.

GUI
For programming the GUI, we decided to use a library called LittlevGL. This would allow us to make a functional GUI in C which isn’t particularly known for its ability to create graphics. Once it was decided that the GUI would be done in LittlevGL it was just a matter of getting a working demo. Following the documentation on the LittlevGL website we were able to get a working demo on PC and could download the code base from their website and start developing. At this point the development moved forward in 2 directions: 1- Developing the GUI itself inside the demo code to ensure it was working. 2- Creating our own LittlevGL using proper drivers so that the GUI code created in the demo could be ported to our own project outside of the demo code provided. Once these steps were completed, all that had to be done was port the GUI code to a Raspberry Pi, connect the Pi to the LCD screen and then test the display.
(See GUI Testing below)

3. Testing Process & Results

3.1 Tests Performed:

GUI Testing:

For GUI Testing we first used the computer screen to test the visuals. This way changes were easier to implement since the computer emulating the GUI was the same computer creating the code for the GUI.

After that we tested the GUI on the LCD screen while connected to a Raspberry Pi to make sure that it would work on the LCD screen.

Effect Testing:

For testing the effects, we decided to go with a simple manual two-phase functional testing procedure. First, the effects were implemented on a laptop in visual studio. One key thing to be mentioned about the testing in the visual studio environment is that the audio samples being tested were wav files, so they were of course, of fixed length. The first phase of testing began with a non-functional test by displaying the frequency content of the original sound file and transformed sound file. The functional portion of this test was determined a success or failure by whether the proper transformation had been performed on the transformed audio output into another wav file. This was done in the early stages of development to get the ball rolling on being able to test the accuracy of the effects while other crucial parts of the design in the audio signal processing realm were being developed. This worked well except for the fact that at some point we should have improved the test environment to support real time audio sampling and manipulation. This became a problem during the integration stage of development and phase two of testing. Phase two of testing was the same style of manual and functional testing and it was done on the Pi, which would be our production environment. Some of the effects were found to be useless in the continuous time domain because they were too computationally inefficient for real time data. Particularly the low pass & high pass filter effects and the fuzz effect. All of the other effects passed their tests on the first try save some minor bugs. The broken effects were redone and later passed testing on the Pi.

Hardware/Electrical Testing:

The hardware components of our system consists of a bank of stomp switches, as you would see on a guitar pedal, a bank of rotary encoders, and an LCD display. All communication between the components and the Raspberry Pi is done through GPIO pin connections. Thus, there were many testing points along the way to make sure that these pieces worked. All hardware was probed for continuity and proper voltage levels (0→5V,3.3V).

The rotary encoders act as a digital knob, not analog, that sends a pushbutton switch signal, as well as two two-bit signals to adjust levels in the software. This means that each encoder has three points in their circuits to test. These were done on Arduino to simply see if the signals were being reached.

For the stomp switch buttons the testing consisted of a simple 5V pulldown circuit going into an Arduino. An LED was then signaled to switch on and off. This then was implemented on the Raspberry Pi.

In order to manipulate the guitar signals in software the waveform is needed to check for voltage levels. Our initial design was to process the sound in our microprocessor, so the input needed to be between 0 and 3.3V. The guitar signal was directly probed from the guitar and sent to an oscilloscope in lab. The results ended up being skewed. This called for some design change (see results section below for more information)

Lastly the USB sound card implementation was just tested to verify that it would properly convert the analog audio signal to digital signal so that the Pi can retrieve the signal and have recognized by the operating system and JACK.

3.2 Test Results & Interpretations

GUI Testing:

The Computer testing went well, it allowed us to make quick changes to the code, test them, and then see them on screen. This meant that we could confirm the style, look, and feel of the GUI before moving to test on the LCD screen.

Effect Testing:

The tests for the first effect(the low pass filter) failed many times, but once we had figured out how to properly implement this effect, the majority of the tests for the rest of the effects went smoothly and successfully in the visual studio environment. This determined that the effects were fully operational on discrete time audio samples.

The second round of testing on the Raspberry Pi went a little less smoothly. Around 3 effects failed because they were only functional on discrete time samples and not continuous time samples for various reasons. This showed an oversight in the design of the first phase of testing and unfortunately debugging on the Pi was not as easy as debugging in the Visual Studio IDE.

Hardware/Electrical Testing

The testing process for hardware concluded a few things about the components: the stomp switches were simple, and straight forward, however, the encoders took some time to learn. The pushbutton signal often bounced and created a lot of noise on our GPIO. A debounce was created to solve this issue. On top of this, the selection between red and green LEDs led to some decision making. The device did not work while an LED was turned off, so we connected each LED color to a GPIO pin. This led to the issue of not having enough GPIO space on the MPU. We had to cut out a few design options in order for this to work, that is, the number of encoders and buttons being used. We are able to create our design with the minimal amount of components used.

After each small circuit was verified we implemented them onto our chosen microprocessor. This includes the rotary encoders and stomp switches. One main issue we came across while manipulating audio was the quality through raw audio into the MPU. This called for a quick solution: the use of a sound card. We chose to use an audio processing module to make a clean signal for us. This connection is done through the USB port on the MPU.

4 Current Products & Literature

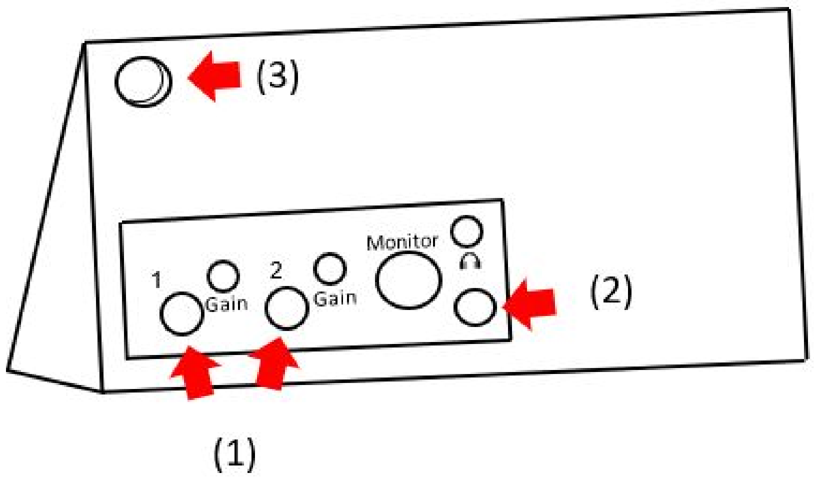
There are other pedals that can perform multiple effects on one device. They contain a set list of effects and have specific settings that can be changed for each effect. The other products on the market have a good selection of effects but not much scalability to include other effects. Our design offers more scalability and simplicity than other products. Our design would also likely be more affordable that the competing products (valued at ~$450). Since their effects are fixed, it would be much more difficult to add more effects to those products unlike our device with its effect selection and limitless combinations.

5 Appendix

5.1 Operation Manual

Setup

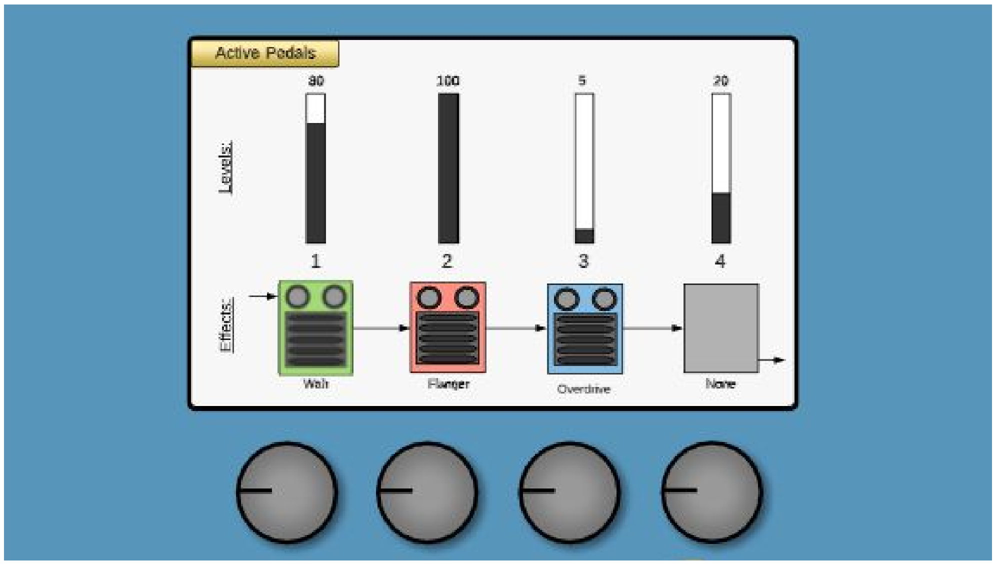
Step 1:
Plug in the guitar/instrument ¼” cable into ¼” input jack labeled “1” or “2” on the usb interface.



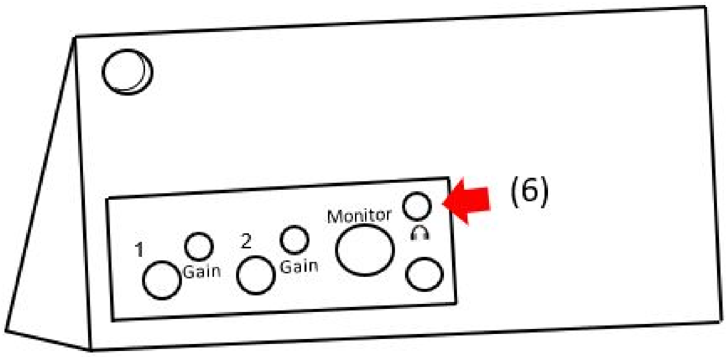
(above: Steps 1-3)

Step 2:
Plug in the speakers/amplifier ¼” cable into ¼” output jack on the bottom right of the usb interface.
Step 3:
Plug in the wall plug power cord (coming out of the hole in the top left of the back panel) into a grounded power outlet.
Step 4:
Choose the effect(s) to be applied to signal by pushing in one of the rotary encoders and selecting the effect via scrolling(turning the previously pressed encoder) through the generated submenu and again pushing in the encoder once the desired effect has been found.

(figure below: steps 4 & 5)

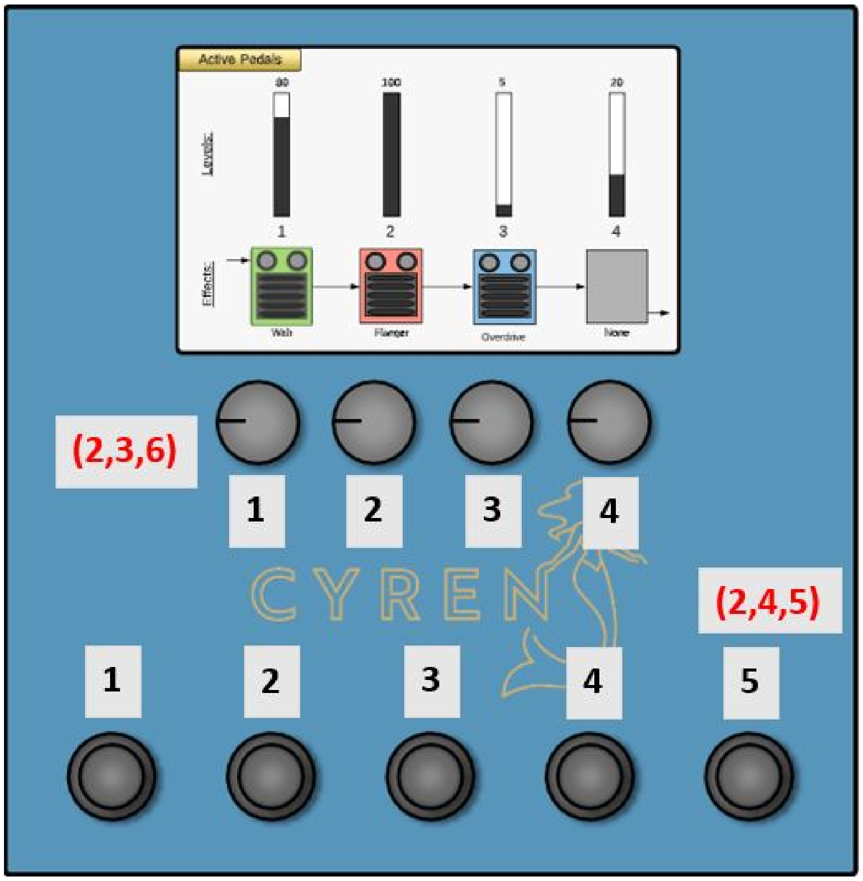


Step 5:
Adjust the application of the chosen effect by turning the encoder that was pressed in the previous step while still in the non-submenu mode(this mode is shown above).
Step 6:
Adjust the broadcast output knob (in the top right, labeled by a headphone symbol) to the desired volume(see below).



Demo/Test
Step 1:
Play the instrument

Step 2:
Turn effect(s) off by pressing the corresponding encoder knob and navigating the submenu to the “no effect” option or by pressing the corresponding stomp switch



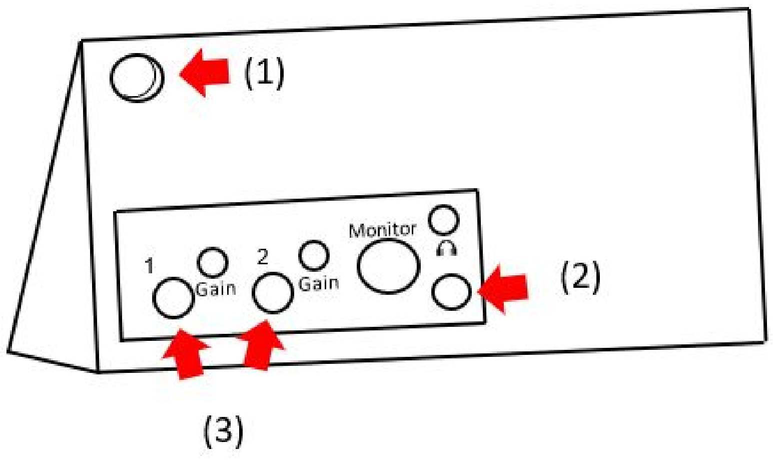
(above: steps 2 - 6)

Step 3:
Turn effect(s) on by pressing encoder knob(s)(labeled 1 through 4) and scrolling through submenu and pressing the knob once desired effect has been highlighted in submenu.
Step 4:
Instantly Select/Deselect an effect and its configuration using the corresponding stomp switch (found at the bottom of the front panel of the device, labelled 1 through 4).
Step 5:
Instantly Select/Deselect all effects and their configurations by pressing the last stomp switch(labelled 5, on the far right, when facing the device).
Step 6:

Adjust the application of the effect(s) on the input signal in real time by turning the corresponding encoder knob.

Deconstruction

Step 1:
Unplug the wall plug(coming out of the top left corner of the back panel) from the of the grounded power outlet.



(above: Steps 1-3)

Step 2:
Unplug in the speakers/amplifier ¼” cable out of the ¼” output jack on the usb interface.
Step 3:
Unplug in the guitar/instrument ¼” cable out of the ¼” input jack labeled “1” or “2” on the usb interface.

5.2: Alternative or previous versions of the design

Design Change 1:

Removal of the looper

We decided to remove the looper from our specifications because the looper added so many requirements to the device such as storage requirements, creation of a database, a compression algorithm, alternative routing, etc. that it would be too time intensive. Our group would not be able to put enough time into the live-looping functionality without sacrificing quality in other, more essential, aspects of our design. So, in the interest of time and resources, we decided to drop the looper functionality and focus our efforts purely on creating a real time audio effects device.

Design Change 2:

Switching from RockPro to Raspberry Pi, USB interface, and Jack

We found that our current design at the time was not feasible. We learned that we were going to have a hard time interacting and programming the RockPro's onboard sound card.

To fix this, we chose to take the RockPro out of our design. To replace the features we needed from the RockPro, we had to employ new hardware and new technologies. The new microprocessor we replaced the RockPro with was the Raspberry Pi. We chose the Pi because it met the computational requirements we needed and it was the microprocessor that the majority of our team was the most familiar with. Along with the Raspberry Pi for processing, we needed another piece of hardware to handle audio transmission in and out of the device. To do this, we used the Focusrite Scarlett Solo USB Audio Interface (Gen 3). Lastly, we needed a software component that could listen for and capture the incoming audio from the USB input port and also transmit that audio back to the USB output port. To accomplish this, we chose the Jack Audio Connection Kit Utility. Jack allowed for us to set up a server that listens for incoming audio on specified ports. We are also able to capture that audio, manipulate that audio, and transmit it to some sort of broadcast port. Because of these reasons, we were able to change the design in a way that made development possible without sacrificing any requirements or specifications.

Design Change 3:

Switch in GUI frameworks

We ended up switching the GUI libraries we were using late in the second semester, because we found that the GUI library we were using, GTK, was intended for desktop environments and is structured so the GUI acts as the top-level "runner" for the rest of the code. We switched from this library to LittlevGL - designed for embedded software. LVGL can run on a separate thread from the main program, essentially running "on top" of the rest of the code.

To follow up, the factors that led up to our design changes included:

Feasibility- We switched microcontrollers because we were unable to interact with rockPro sound card which was necessary to the functionality of our device

Time and Resources- We felt that we did not have enough time to implement the live-looping functionality without sacrificing quality and development hours in other aspects of our design.

Time and Resources- The switch in GUI libraries was made to make the development and integration into the codebase run more smoothly, preserving time and resources.

Our design changes addressed the new information in one case by preserving our time and resources and losing a feature from our original design. The other addressed the new information we received, by preserving time as well, but did not sacrifice any functionality of the design.

5.3: Other Considerations

Lessons Learned:

- Give more weight to testing
  For example, for testing the software effects, our test engineer should have tried to make the testing environment for the software effects as similar to our production environment as possible. The main issue being that our test engineer was testing with discrete time audio samples. Whereas on the actual device we were working with non-discrete time samples. This caused issues in a couple areas. Mainly, in the low & high pass filter effects. Our test engineer spent a lot of time testing, developing, and perfecting these effects in the test environment with discrete samples. However, once our test engineer moved to real time audio, the

effects for the low & high pass were rendered useless. The algorithms our test engineer had used were for filters with finite impulse response which worked for the type of data our test engineer was receiving in the test environment. But in production, the impulse response of the data was infinite(being continuous). Once our test engineer realized that the effects he had spent so much time on were useless, there was not much time left in the semester, so he had to quickly come up with two simple and easily implementable algorithms that would work with an IIR filter and our data.

- Scheduling
  Scheduling was something that wasn't too difficult for our team in the first semester. We had two meeting times, one with our faculty advisors and one as a team each week. And all of our teammates could be present at the meetings. Unfortunately, this semester, during the weekdays, only 3 of our teammates lived in ames and on the weekends only 2 of our teammates were present in ames. The rest of our team is scattered elsewhere across Iowa. So needless to say meeting in person every week was not practical. And to make matters even worse, due to all of our team members busy schedules, there wasn't a single time slot in the entire week where everyone in our team could meet and participate in the meeting. This is something that we should have taken more seriously and tried to combat more accurately to stay organized and keep the project moving forward. We should have moved to some type of a multi-meeting schedule, perhaps 3 meetings a week where a portion of the group that could make that meeting time would attend/voice call/video conference. This would make sure that everyone in the group had attended at least one meeting that week and that everyone was on the same page with the status of the project's development.

- Research
  We believe that if we would have put more time into the first semester of the class, we would have had fewer issues in the second semester. Mainly, that being the choice of our microcontroller, the RockPro. We probably should have looked into the amount of documentation and how easy it would be to interact with the RockPro's onboard sound card. If we had done more research on this device, we would not have had to switch

microcontrollers and design late in the development life cycle like we did. We also could have done a lot more research in the theory side of audio signal processing and implementation as a group and that would have made the project progress much more efficiently. For example, understanding things like sample rate, bit depth, and fourier transformation, etc. at the start of the semester would have made the development of the project flow much more smoothly. We also could have put more time into the research of our GUI because that was another aspect of the project that was changed in the second semester.

- Do less, think more
  This lesson goes along with doing more research, but can be applied even more granularly throughout the development of our project. Like most things in life, the act of doing something is actually pretty simple. Typing code for example, typing code is very easy but if you don't know what to type, creating something significant is near impossible. We believe that throughout the entire 2 semesters of the project, we should have thought more about our design decisions because of the impact that these decisions have on the flow and probability of success of the project. There were quite a few instances where a little extra time and thought toward remembering things and planning things could have gone a long way. A lot of time and effort could have been saved by finding the most effective solutions to the requirements we wanted to fulfill in our project.

- Communication
  Communication is the cornerstone of a group project. We could have improved the communication practices between group members considerably during the course of this project, by applying standards toward how communications should be practiced, such as requiring team members to update the group on their progress weekly and requiring team members to respond to questions or topics that they are involved in/ associated with in a timely manner. Communication between team members working in different areas of the project should have been had more frequently about how to integrate their respective sections into the final project so that combining the different pieces of our project in the final stages of the project could have run more smoothly. Also, communication of

design changes should have been done in a timely manner at the beginning of the semester.

- Accountability
  Because everyone in our group was very comfortable with each other, we failed to hold each other accountable to adhering to project timeline deadlines and to gaining progress on the project in a timely manner as individuals.

5.4: Code

https://git.ece.iastate.edu/sd/sddec19-15