EE 542
Fillipo Cheein
Izak Walker

Guitar Loop Pedal

Introduction:

This device takes the electrical audio signal from an electric guitar. It converts it to a digital signal to apply digital signal processing techniques to record and loop user determined lengths of audio. This recorded audio is then converted back to an analog signal to be used by a guitar amplifier. Upon recording the audio (the length of which is determined by two footswitch presses), it will continuously loop. This allows the user to implement many previously unattainable sounds, such as backing tracks and emulating multiple guitar lines. This device is widely used by musicians from amateaur to professionals.

Design Process:

We first did research to find what were industry standard loop pedals to see if we could realistically implement or replicate those features. This research informed some of the component selection that we made in order to create the system. Our main finding was that the resolution of the ADCs and DACs used were either 16 or 24 bits. We then decided to implement an ADC and DAC that use a 16 bit resolution to create a very high quality audio signal reconstruction. Additionally, we found that it was very standard to implement 5kHz anti-aliasing filters for improved ADC performance, as well as an output low-pass filter to further reduce unwanted signal content. In order to utilize the full range of the ADC, and prevent any negative voltage that could damage it, we needed to amplify the signal using an op amp and also offset it to half of the supply voltage. This meant we needed a rail-to-rail to use the full range without unwanted clipping due to supply voltage constraints. Since we needed to purchase an op amp we decided to implement a 3rd order Sallen-Key low-pass filter due to its performance and since there are a minimum of 2 op amps per chip typically. We also needed a footswitch inorder to communicate to the raspberry pi (via GPIO) that we were either starting or stopping a loop recording. To control the ADC and DAC data acquisitions and outputs, we used a C SPI driver library. We segmented our code processing into four different "states", bypass, recording, playback, and mixing. These are explained further in the software flowchart section.

System Architecture:

<u>High Level Application:</u>

This device was specifically created for electric guitars. In fact, the 5khz cutoff frequency for the antialiasing filter was chosen considering that electric guitar frequencies vary between 80Hz to 2kHz. The system recorded up to 4 minutes and 30 seconds of a sound. The absolute maximum duration was not tested, but we deemed this to be a sufficient metric. Most of the testing was done using the push button since most of the artists use the device by choosing when to start and stop the recording. The reaction time of the recording is seemingly instantaneous. No perceptible latency due to the pushbutton was noticed or measured. The mixing of the recorded sound with the incoming signal is smooth and reliable. Little distortion can be noticed. The reproduction speed of the recorded input is similar to the real time sound played. This allows the recorded sound and the incoming sound to feel synchronized to outside hears. The recorded loop is synchronized but has a slightly lower volume than the incoming sound. This is likely due to our mixing algorithm, but this is more ideal for a loop pedal since you want the loop to be more in the background to play over.

Low Level Implementation:

The sampling rate was measured to be around 35kHz. This was determined experimentally through checking the loop index values after a set duration. The SPI clock frequencies were tuned to produce the best loop audio quality (speed and minimal distortion). We used the SPI0 and SPI6 interfaces on the Raspberry Pi 4 by modifying the config.text file under the \boot directory. We had to specify that SPI6 was enabled, as well as set the CS pin (GPIO16) using dtoverlay and dtparam respectively. We used polling to determine check for new inputs from the foot switch to start and stop recording the loop. This was because it was easier to implement than interrupts and did not appear to degrade the system performance. The hardware implementation is described in the hardware schematic section and refers to what each stage is doing.

Real-Time Implementation:

In order to avoid potentially missing new ADC samples when outputting the loop samples, two SPI interfaces were utilized on the Raspberry Pi4B. This way, it could be ensured that mixed sample value for any new input with the current loop sample would not miss any data.

Decimation was necessary to approximate the proper timing of the loop playback to the timing of when it was recorded. Without the decimation, the recordings were consistently slower than the rate they were recorded at. This slow down is likely a combination of the sampling rate, the computation time of our program, and memory access delays we are encountering since we have so many samples, they likely cannot all fit within the caches.

Hardware Schematic:

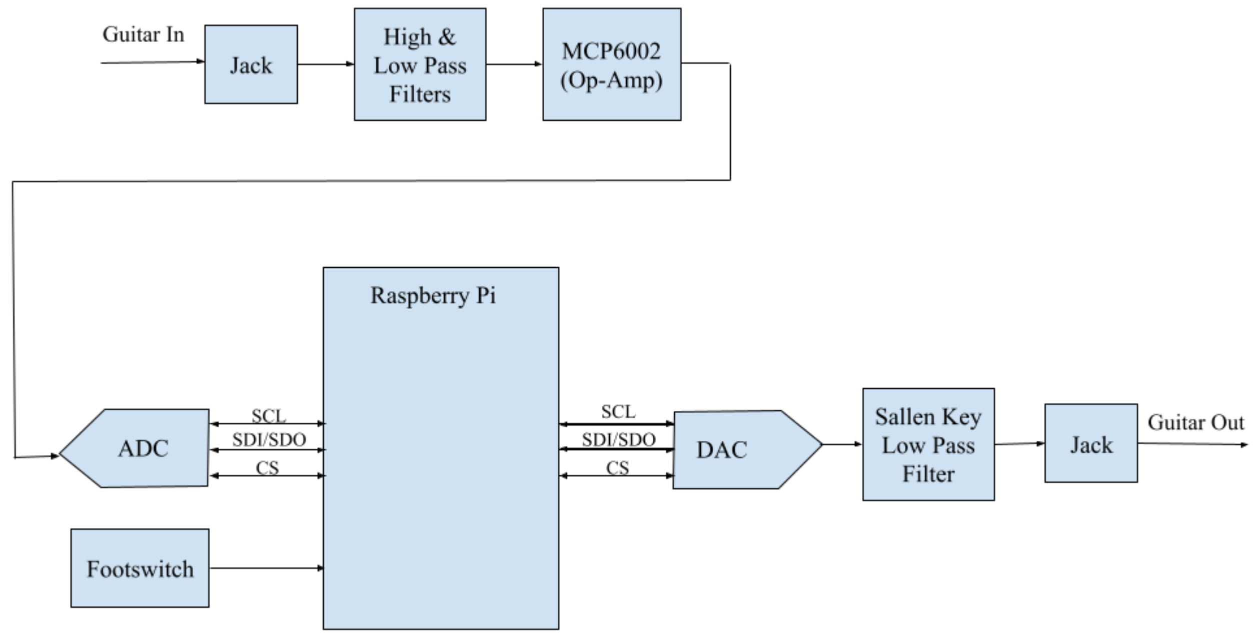

Figure 1: System Architecture of the Guitar Loop Pedal

<u>Schematic Description by Stage:</u>

Input Stage:

The low pass filters on the input stage of the will filter out high frequency harmonics that can contribute to aliasing. This reduces error in the ADC's signal acquisition. The high pass filters remove DC and any unwanted hum distortion present in the guitar's input signal. The op-amp is then used to increase the guitar's signal level to reach the rails in order to utilize the full scale range of the ADC.

Output Stage:

In the output stage, the DAC will receive the processed signal from the Raspberry Pi. This component will transform this digital signal into analog. The signal will then go to the Sallen Key Low pass filter which will remove the 5th and greater harmonics from the signal being outputted. This will help out with limiting harmonic distortion. Once the signal is sent to the Jack, it is ready to be sent to a guitar amplifier.

Button Switch:

The footswitch controls whether the effect is active, and the duration between the presses determines how long that the audio is recorded for.

Software Flowchart:

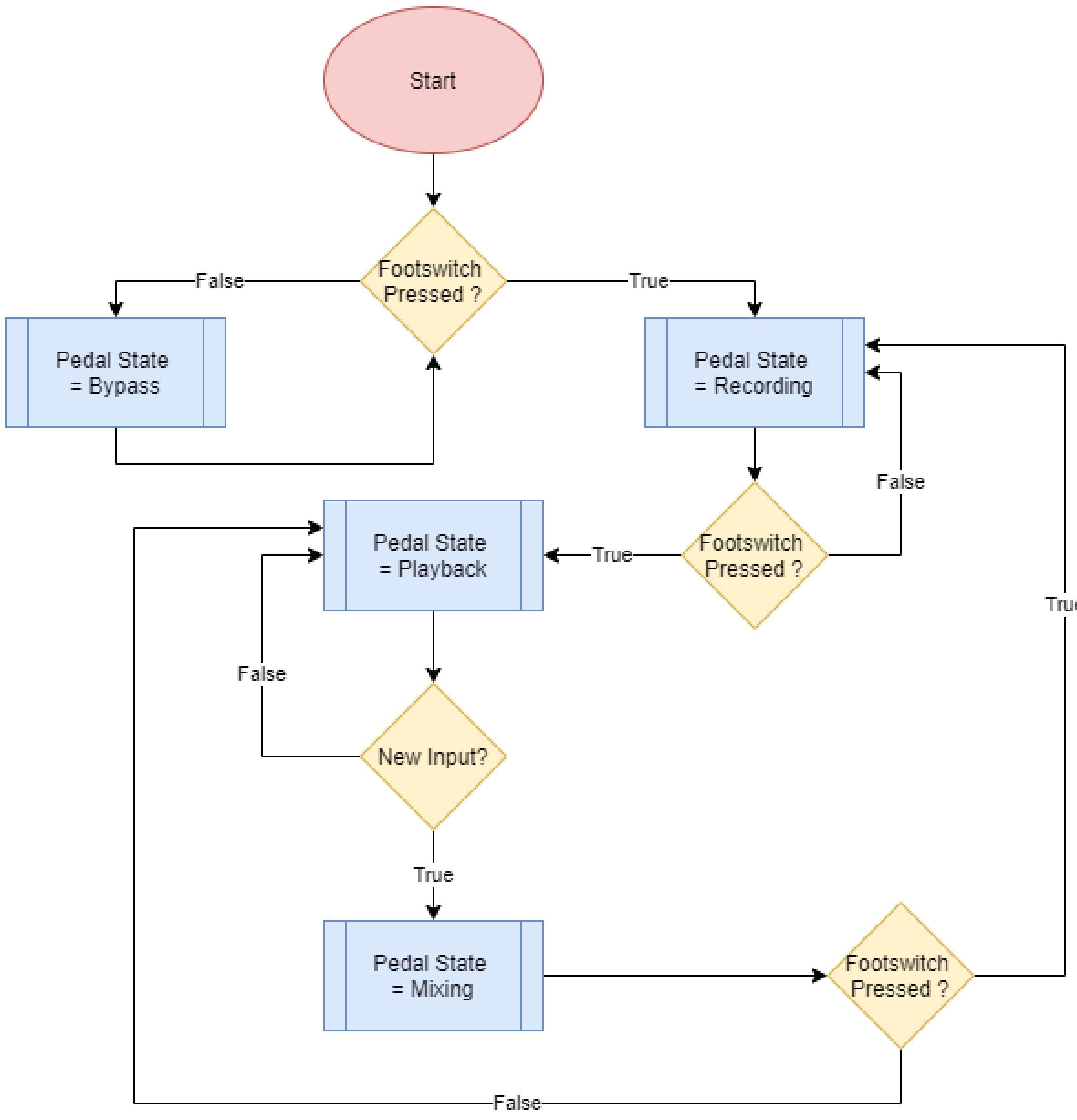

Figure 2: High-level Software Flowchart

<u>Bypass State:</u>
The Input is connected to the Output. Whatever is captured by the ADC will be sent to the DAC directly. No data will be recorded and processed.

<u>Recording State:</u>
Upon the button press, the input will be recorded through the ADC and stored in memory ready to be processed. This state will include a signal processing step.

Playback State:
The recorded audio will be played back and output through the DAC. Upon finishing the recorded audio, it will be looped until there is a new footswitch press to start a new recording.

Mixing State:
This state will allow the device to store and reproduce additional sounds. The mixing equation used is:

$$\begin{aligned} &if\ A < 512\ and\ B < 512 && output = \frac{AB}{512} \\ &else && output = 2(A + B) - \frac{AB}{512} - 1024 \end{aligned}$$

Challenges:

There were several challenges throughout the project. Hardware related problems concerned the ADC and DAC selected. These ICs were surface mount and we did not have the instruments to aid us at home. Through some phone zooming, thin solder, solder wick and solder flux it was possible to successfully solder the ICs. In addition, for the chosen ICs there were little support and user documentation. Despite spending numerous hours on datasheet and support chats it was not possible to make the ADC and DAC work properly. Despite receiving a neat DC signal, the ADC was not able to capture and write to the Pi any AC signal. The DAC was not able to be turned on despite setting it up as advised by the company support. This issue was solved by rerouting toward simpler and more commonly used ADC and DAC. The MCP3008 was used as an ADC. This ADC is simpler in its use and it's widely used in DIY projects. MCP4921 was used as a DAC. This DAC was familiar since it was used in a previous class. The only down-side to using the different ADC and DAC was that they had reduced resolution. The ADC only had a resolution of 10-bits and the DAC had a resolution of 12-bits. The lower resolution leads to some added signal distortion that our original ADC and DAC would not have created. The output Sallen-Key low pass filter was able to mitigate a majority of this signal distortion, so a clean signal could be heard when the gain of the input op amp was lower. There was some clipping and fuzz-like distortion that occurred at higher gain, but this could be used to add to the sound rather than detract from it. The code used to implement our initial ADC and DAC is not present in our final code. Software related challenges included the use of SPI libraries. The Raspberry Pi 4 is a fairly new device and it is not as used as the previous editions. As a consequence, few libraries were developed for its use in peripheral communication. Most of the libraries were developed for the Broadcom BCM2837 processor used in previous Pis rather than for the Broadcom BCM2711 processor used in the Pi 4. This issue was solved by using an SPI and GPIO library that had little documentation and had to be tailored for our use. Additionally, we had trouble using two spi interfaces with this library. We overcame this by modifying config.text file with dtoverlay and dtparam to specify that we were using SPI0 and

SPI6. We also had to use different device numbers that corresponded to the chip selects that were used for the library to work with two spi interfaces.

Components

Description and Usage:

A 1/4" audio jack will be used to input and output the audio signal. Input will go through an anti-aliasing filter before reaching the ADC MCP3008. This anti aliasing filter is made of 2 low pass filters and 2 high pass filters. The low pass filters will remove harmonicas that could create aliasing and set the cutoff frequency to 5kHz. These two filters are built using a combination made of a 4.7 kohm resistor and a 6.8nF capacitor and a combination made of a 100kohm resistor and a 270pF capacitor. The high pass filters will remove the DC signal. These two filters are made of a 1Mohm resistor paired with a 100nF capacitor and a 4.7 kohm resistor paired with a 4.7nF capacitor. This anti-aliasing filter is connected to an MCP6002 op-amp that will be used to amplify the signal coming to the ADC, so that the sound does not get any negative signed signal. A Potentiometer is added to the filter so to choose the gain of the amplifier from 1 to 21.

The ADC is connected to a Raspberry Pi 4 that processes the output and sends it to the DAC MCP4921. This output signal goes through a Sallen & Key 3rd order low pass filter so to remove harmonics above 5 kHz. This filter is built by combining two 4.7 kohm resistors and three 6.8 nF capacitors.

In addition, a footswitch is connected to GPIO pin 12 of the Raspberry Pi so to determine when it should record the sound and when to stop.

BOM:

	Items	Type	Quantity	Reference #	Price per unit	Exit Price
I/O	ADC	MCP3008	1	579-MCP3008-I/P	$2.32	$2.32
	DAC	MCP4921	1	579-MCP4921-E/SN	$2.07	$2.07
	Op-amp	MCP6002	2	MCP6002T-I/SNVAO	$0.33	$0.66
	Jack	1/4" audio jack	2	NMJ6HCD2-RM	$1.52	$3.04
Switches	SW0	3PDT	1	612-FS573PLT2B2M2QE	$5.64	$5.64
Potentiometer	RV0	500k Reverse Log	2	858-P160KNP0C20C500K	$0.91	$1.82
Resistors	R0	1 Meg	2	MFR-25FRF52-1M	$0.13	$0.26
	R1	300 k	1	MFR-25FRF52-300K	$0.12	$0.12
	R2	100 k	2	603-MFR-25FRF52100K	$0.13	$0.26
	R3	50 k	1	603-MFR-25FRF5249K9	$0.12	$0.12
	R4	10K	4	603-MFR-25FRF5210K	$0.12	$0.48
	R5	4.7 k	6	660-MS1/4DCT52R4701	$0.12	$0.72
	R6	300	2	603-MFR-25FRF52-300R	$0.12	$0.24
Capacitors	C0	220 uF	4	140-REA221M1CBK0811P	$0.13	$0.52
	C1	4.7 uF	3	667-ECE-A1EKA4R7B	$0.21	$0.63
	C2	100 nF	5	810-FG28X7R1H104KNT6	$0.20	$1.00
	C3	6.8 nF	5	81-RPEE41682M2S2A03A	$0.33	$1.65
	C4	270 pF	1	581-SR211C682MARTR1	$0.22	$0.22
Misc.	Breakout Board	SMT	1	485-1206	$4.50	$4.50
Total						$34.17

Reference:

Toth T. Viktor, “Mixing Digital Audio,” Available: ,Viktor T. Toth - Mixing digital audio (vttoth.com). [Accessed February 12, 2021].

Note: Any code libraries used are referenced with the code itself