Pi Piano

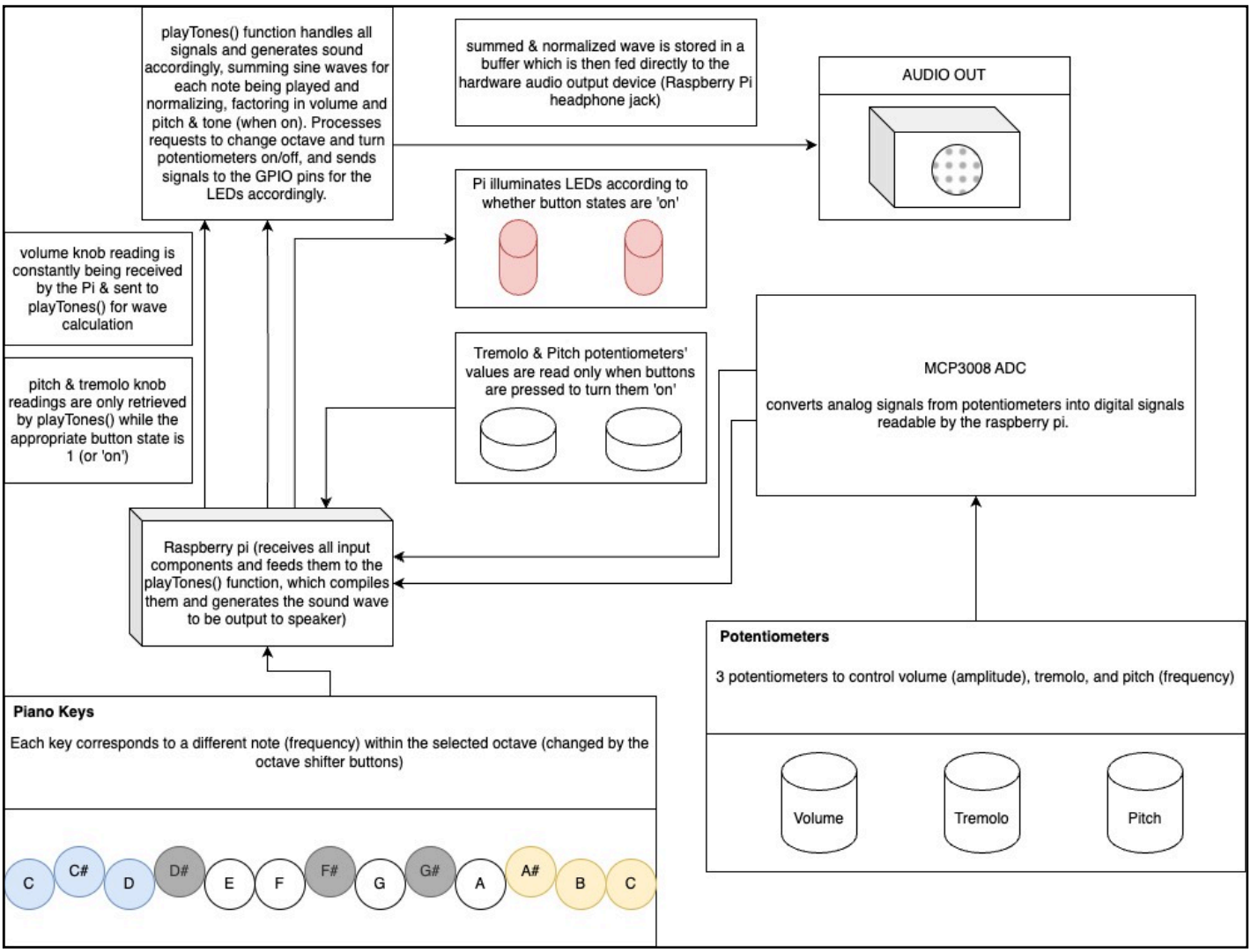
Abstract

This project constitutes the design and implementation of an electric keyboard using the Raspberry Pi 4, with real-time audio synthesis capabilities. The system utilizes GPIO inputs, analog potentiometers, and ALSA audio output to generate real-time polyphonic sound in response to user input. Features of the Pi Piano include easy-to-use pitch bending, overdrive, variable-frequency tremolo effects, and octave changes, in order to provide users with a low-cost, capable & versatile tool for music creation and experimentation. This report describes the hardware and software components & design, implementation details, experimental endeavors and results, and areas for improvement in future development.

1. Introduction

I was drawn to this project due to my interest in playing around with making music in GarageBand. For a long time, I have wanted to invest in a little keyboard to aid in my musical endeavors, but have been hesitant to buy new equipment due to costs, and not knowing exactly what will end up resonating with me. Developing an electronic keyboard with the Raspberry pi provided me a low-cost way to experiment with making music and generating different & interesting sounds, without having to invest in expensive music equipment. To this end, the 'Pi Piano' aims to provide a compact, cost-effective solution for musical synthesis. Utilizing the computational power of the Raspberry Pi and its GPIO capabilities, my project implements digital audio generation, real-time input handling, and interactive sound effects. The project serves as an example of hardware interfacing and digital signal processing, utilizing the versatility of embedded systems, and providing a low-cost method for musical experimentation.

2. Design



The high-level design of my project has remained fairly consistent throughout the development process, while details of its implementation have changed over time. The plan was always to have a set of keys that send signals to the Pi to trigger different notes, to include functionality for octave changes, to include potentiometers to adjust certain sound effects (though what those effects were going to be, specifically, certainly fluctuated over the course of the development process) which would be applied to the notes being played, and then to output that sound to a speaker of some sort. Initially, I had also planned to include an LCD display of some sort. I had only planned on being able to implement monophonic sound (one note can be played at a time), and output that sound to a small speaker connected to a GPIO pin.

Over the course of the implementation of this project, while the overall design did not much waver, the specifics of the project certainly evolved into something unlike exactly what I had anticipated (dare I say, something better than I had previously imagined). I iterated through different wave forms and audio output systems, in order to achieve the highest-quality sounding output that I could. I scrapped the display in favor of putting my time into better implementation of the sound generation & processing, and I ended up (somewhat to my surprise) discovering that I could implement polyphonic sound output, which I had sort of assumed would just be too complicated to get around to. I ended up finding that certain sound effects (vibrato, wah) were harder to implement than others (volume, tremolo, pitch), which somewhat informed my decisions, but these were all effects that I had been considering. In the future, I'd like to implement more effects, and maybe even a way to change which effects the pots are controlling (or just add more potentiometers… there are 5 more channels on the MCP3008 I'd like to utilize).

Hardware Components:
Raspberry Pi 4: The central processing unit.
GPIO Buttons: 13 'keys' (buttons) representing musical notes across multiple octaves, as well as buttons for turning effects on/off, and a power button to terminate the program.
Potentiometers: Analog controls for volume, tremolo, and pitch bending effects.
LED Indicators: Display the status of pitch bending and tremolo effects (on or off).
MCP3008 ADC: Converts analog potentiometer signals into digital values.
Audio Output: Configured via ALSA library, sent directly to the Pi's headphone jack.

Circuit Design:
The circuit connects 13 buttons to GPIO pins configured with pull-up resistors. Analog potentiometers interface with the Raspberry Pi through the MCP3008 ADC, while LEDs are connected to GPIO pins for status indication. Audio output is routed through the Raspberry Pi's built-in audio hardware.

Libraries:
ALSA: Used for real-time audio playback & interfacing with the Raspberry Pi's built-in audio output/ hardware.
pigpio: Handles GPIO inputs and SPI communication.
math.h: Provides mathematical functions for signal processing.

Algorithm:
1. Initialize GPIO pins, SPI, and audio hardware.
2. Continuously poll GPIO inputs to detect key presses.
3. Read potentiometer values to adjust volume, pitch, and tremolo effects.

4. Generate and normalize sine wave audio signals.
5. Output audio through ALSA to the Raspberry Pi audio output device (headphone jack).
6. Check for special button combinations to handle octave changes or power off.

3. Build Process

1. To begin with, I wrote some basic C programs to generate different types of waves (square, triangle, sine) using basic GPIO high/low signals and PWM. At this point, I just planned on using a basic square wave, as that was what sounded best on the little GPIO-connected speaker I was using at the time.
2. I configured some buttons to GPIO pins, and got to the point where I could, in conjunction with an array of frequencies corresponding to different notes, play different notes (monophonic) with different buttons (accessing frequencies in the array and generating square waves appropriately).
3. At this point, I started adding potentiometers, and working on implementing the effects whose parameters could be adjusted with the potentiometers
4. During the development process, I decided that I wanted to be able to send audio directly to the Pi hardware (headphone jack) rather than the little GPIO speaker, so that I could produce higher quality sound with a more versatile interface (nice to be able to use headphones, or hook the piano up to a louder speaker).
5. In learning how to interface with the audio output/hardware, I discovered that using a buffer, I could generate a sine wave much more clearly than I had been able to using PWM, so I switched to using sine waves instead of square waves, as they achieved a sound much closer to the smooth tone I had initially envisioned for this project.
6. Due to the way I was storing sound waves in a buffer, and the fact that I was using sine waves rather than square waves, I discovered that I could implement polyphony by summing sine waves. This discovery was a thrill.
7. Had to figure out how to normalize this output (for instances where the amplitude of summed sine waves grew too great for decent audio output.
8. Finalized the software aspects of playing multiple notes, and generating the appropriate sound output — at which point I went back and reworked my effects/potentiometer readings to interact with the new algorithm for generating sine waves.
9. I iterated through a number of attempts at different sound effects, and ended up implementing and keeping the volume, tremolo, and pitch-bend effects, as they were the ones I had most successfully implemented, and I was quite pleased with the versatility they provided the piano.
10. While working on the potentiometers and related sound effects, I added buttons to turn these effects ‘on’ or ‘off’— allowing the user to select whether effects are in use, and also making the program more efficient by only reading the potentiometers when in use, rather than constantly.
11. I attempted to implement an LCD display using I2C communication, but ran out of time. I plan to implement this in the future.
12. Finally, I cleaned up my code and added the functionality for the power button (to safely exit the program via hardware input) and added to the rc.local script on my pi to run the program on startup, so that the piano program could be run headless. All you need to do is plug it in and wait for the program to start up, and then press the ‘power’ button to safely terminate the GPIO and exit the program.

4. Experimentation

1. I experimented with different wave forms— tested out some different C programs that would generate square, triangle, and sine waves using different methods. I tried generating a sine wave using PWM, which did not sound so good, and eventually by calculating the wave piece-wise and storing it in a buffer, which ended up sounding much better and was the method I ended up keeping in my project implementation.
2. I Experimented with different methods of audio output (which also informed/was informed by the wave form(s) used). Initially I had planned to use a little speaker hooked up to a GPIO pin, but in an effort to provide higher-quality sound output, ended up looking into & implementing sound output directly to the Pi hardware.
3. Not necessarily 'experimentation,' but as a method for testing my software as I developed it, I used lots of print statements to observe the values being read by different inputs (buttons and potentiometers), as well as to determine where in my code errors were occurring.
4. I conducted many experiments while developing the audio effects— I tried implementing a number of different effects (wah, vibrato, tremolo, pitch bend, noise/gain) to varying degrees of success. Throughout the process of developing these different effects, I did lots of testing that largely involved changing something, running the program, and seeing how the sound had changed. I don't really have a good way of including these tests/results in written form, but the entire development process involved lots of these sort of experiments where the result/output was monitored by listening to the sound produced.

5. Experimentation Results

The results of my experimentation, to put it simply, are the Pi Piano as it exists today. As the program output is pretty much entirely audio, I don't have many results to show for my experiments. My experimentation largely involved tweaking things until they sounded the way I wanted them to, and while I went through many stages of experimentation and many iterations of sounds produced by the project, I don't really have anything to 'present' in a written report for all that evolution.

6. Conclusion & Future Work

This project demonstrates a low-cost, functional electric piano built on a Raspberry Pi. By integrating real-time audio synthesis with hardware controls, it highlights the potential of embedded systems in musical applications. Future improvements could further enhance its usability and feature set, making it a more versatile tool for musicians and hobbyists alike.

Successfully-Implemented Features:
Polyphonic Sound: Supports simultaneous key presses.
Real-Time Effects: Smooth pitch bending and tremolo modulation, as well as volume adjusting which also functions as an overdrive effect at higher levels.

Octave Switching: Easy transition across five octaves.
Headless Operation: Program runs on boot, and can be safely terminated before shutting of the Pi using the power button provided.

Challenges
1. Configuring audio output to hardware.
2. Determining optimal wave form and generation method
3. Calculating and successfully summing sine waves to provide polyphonic sound output.
4. Implementing different effects and their control by the potentiometers.
5. Resolving sound clipping by adding normalization to prevent audio distortion.

Future Work
Advanced Audio Effects: I would love to add more features/effects like reverb, wah, vibrato, noise, or chorus in addition to the tremolo and pitch-bending effects I've included in the project.
Multi-Channel Output: It would be interesting to explore stereo or surround sound configurations.
User Interface/display: I would like to have implemented an LCD display to communicate to the user certain parameters, such as note(s) being played, current octave, potentiometer status/reading, etc.
Improved Design & Refined Software: As I continue to develop this project, I would like to continue to refine my code, and ideally improve upon the hardware configuration by making it more compact and designing a better enclosure for it (the cardboard box works well for now, but may not hold up over time). Eventually I could look into 3-D printing some sort of case for this, once the design is more finalized.

8. References

1. ALSA Project Documentation: https://www.alsa-project.org
2. ALSA PCM (digital audio) Documentation: https://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html
3. pigpio Library Documentation: http://abyz.me.uk/rpi/pigpio
4. MCP3008 ADC Datasheet: https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf
5. Raspberry Pi Documentation: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
6. ChatGPT assisted me with certain aspects of developing this project, including providing a framework/parameters for interfacing with the Raspberry Pi audio hardware via the ALSA library, generating the formulas for calculating, summing, & normalizing sine waves, and providing guidance on how certain audio effects might be implemented within those formulas (to varying degrees of success). https://chatgpt.com/