STRING/DRUM SYNTHESIZER

By

Abhi Nayak
Joel Schurgin

Final Report for ECE 445, Senior Design,
Fall 2024
TA: Manvi Jha

11 December 2024
Project No. 21

Abstract

This project presents a compact synthesizer powered by the Karplus-Strong algorithm for real-time string synthesis. Utilizing an STM32 microcontroller interfaced with a Raspberry Pi, the system generates unique sounds controlled via emotive knobs, allowing users to manipulate synthesis parameters with precision. The design emphasizes portability and user-accessibility while achieving robust sound design capabilities. Testing validated the system's ability to produce diverse sound characteristics, showcasing its potential for creative applications in music production. Future enhancements include integrating a display for waveform visualization, further enriching user interaction and functionality.

Contents

1 Introduction

This project tackled the challenge of creating a cost-effective, portable, and expressive synthesizer designed for real-time sound synthesis. Its purpose is to provide musicians and sound designers with a compact, standalone tool for exploring unique musical possibilities. By incorporating intuitive, user-friendly features, the synthesizer simplifies the process of sound design, making it accessible to both beginners and seasoned users.

At the heart of the synthesizer is our modified Karplus-Strong algorithm, which emulates plucked string sounds and serves as a versatile foundation for creating rich tonal textures. The device features descriptive emotive knobs, transforming complex sound parameters into intuitive controls. This design enables users to shape their sounds effortlessly, regardless of technical expertise. Pairing the synthesizer with a MIDI keyboard enhances the creative process, allowing users to experiment by playing, adjusting, and refining sounds iteratively. This streamlined workflow fosters creativity and encourages exploration.

The accompanying report begins by outlining the project's motivations and objectives, discussing the design choices, and identifying the constraints that shaped the design. It then delves into the system architecture, detailing the implementation of the Karplus-Strong algorithm on the Raspberry Pi 4, the hardware design using the STM32 NucleoBoard for the emotive knobs, and the overall sound production process. The report features a section where the device’s final performance is evaluated, showcasing how it was tested and its current capabilities. Finally, the conclusion highlights the overall project's success and suggests future enhancements, such as adding a waveform display to improve user interaction.

This project demonstrates that a microcontroller-based synthesizer can deliver dynamic, high-quality sound outputs in a portable and affordable package. By blending core functionality with intuitive controls and an accessible workflow, the synthesizer effectively lowers technical barriers while inspiring creativity in sound design.

1.1 The Problem

Musical artists typically aim to evoke emotion with their art. They may lack the technical knowledge to produce their music. With sound design, the learning curve is steep, as it requires a comprehension of technical aspects, demonstrating the intersection of science and art. Shaping sounds in a way that is efficient and user-friendly is where our synthesizer comes in. The process of musical production can feel overwhelming, involving music theory knowledge, creativity, a huge time commitment, and a trained ear; hence, beginners, may feel lost amidst the jargon and the sheer number of options with regards to sound design.

1.2 The Solution

To address the complexity, we built a synthesizer that can produce a range of sounds but provides simple “emotive” knobs that describe musical qualities rather than technical parameters. Parameters such as filter cutoffs, oscillator frequency, low frequency oscillators, etc... become brightness, boominess, dampening, scratchiness, sharpness, and strike weight. These knobs would also control the internal parameters of the system to achieve the desired effects on the input sound. The intended workflow of this device for a musical artist is to record a sound, tweak the knobs, and play that sound by pressing a key on a MIDI keyboard. It is up to the musician to record the sound into their own recording software for further manipulation.

1.3 Performance Requirements

Listed below are the high-level requirements that were set at the start of the project to dictate what a successful design by the end of the semester would look like.

- Potentiometers are able to control synthesizer parameters with at least 8-bit resolution.
- Notes are played at the pitch of the key pressed on the USB MIDI keyboard with latency faster than 30ms.
- 110+/-1% ohm output impedance and sound can be heard with headphones.

1.4 Subsystem Overview

The String/Drum Synthesizer consists of several interconnected subsystems, each with a specific role in the overall functionality of the device. The **Synthesizer Subsystem** serves as the core of the design, generating sounds using a modified Karplus-Strong algorithm. This algorithm uses a combination of a pulse generator, and an echo chamber modeled with digital delay, feedback, and filtering. The subsystem processes MIDI input from a USB keyboard to produce notes at the correct pitch and latency, while allowing dynamic manipulation of sound through parameters controlled by the Emotive Controls subsystem. The output of this subsystem is a pulse-code modulation (PCM) wave, sent to the amplifier subsystem for further processing.

The **Emotive Controls Subsystem** enhances the user experience by providing a set of knobs labeled with descriptive terms such as “scratchiness” or “sharpness” that correspond to underlying synthesizer parameters like distortion, filter cutoff frequencies, and feedback levels. Each knob adjusts a voltage, which is read by the STM32 microcontroller and converted into digital values. These values populate a buffer and are sent directly to the Raspberry Pi, enabling real-time adjustments of the synthesizer’s parameters to shape the sound output.

The **Amplifier Subsystem** ensures that the audio signal is suitable for playback through headphones. It receives the PCM wave from the synthesizer, filters and amplifies it, and adjusts its impedance to match standard headphones.

The processed signal is then sent to the **Sound Output Subsystem**, which outputs the audio via a ¼-inch stereo jack. This jack allows the user to connect headphones or external speakers to hear the sound they have created.

All subsystems are housed in a physical enclosure that includes interfaces such as volume controls, power switches, USB ports, and an audio jack, enabling intuitive operation. The interconnection of these subsystems—where the synthesizer generates sound, the emotive knobs shape the parameters, and the amplifier prepares the signal for output through the audio jack—ensures seamless functionality.

In terms of block-level changes made during the semester, we made the most adjustments to the Power Block, Microcontroller, and Emotive Controls subsystems. Starting with the Power Block subsystem, at the beginning of the semester, we had two voltage regulators that would drop down the 5 Volts from our 5V Power Supply Adapter to 3.3 Volts and 2.5 Volts to be used in the Microcontroller and Amplifier subsystems. However, after making some changes to these subsystems, we realized that we only needed 5 Volts and 3.3 Volts for our entire system. Thus, we removed the voltage regulators, added a simple voltage divider circuit to produce 3.3 Volts, and added a few capacitors instead to stabilize the voltages. Next, in the Microcontroller subsystem, one major change we made was with the processor we decided to use. At the start, we were going to use the STM32 processor on our PCB to handle all the processing within our system. But, after encountering issues with our PCB design, we decided to instead switch to a Raspberry Pi 4 to serve as the heart of our Microcontroller subsystem. Next, in the Emotive Controls subsystem, we had planned to connect each knob to the inputs of an ADC chip and then send those converted values to the Raspberry Pi for further processing. However, after failing to get desired outputs from our circuit involving that chip, we switched to using the built-in ADC on a STM32 NucleoBoard and interfacing that with our Raspberry Pi to provide the correct functionality for this subsystem.

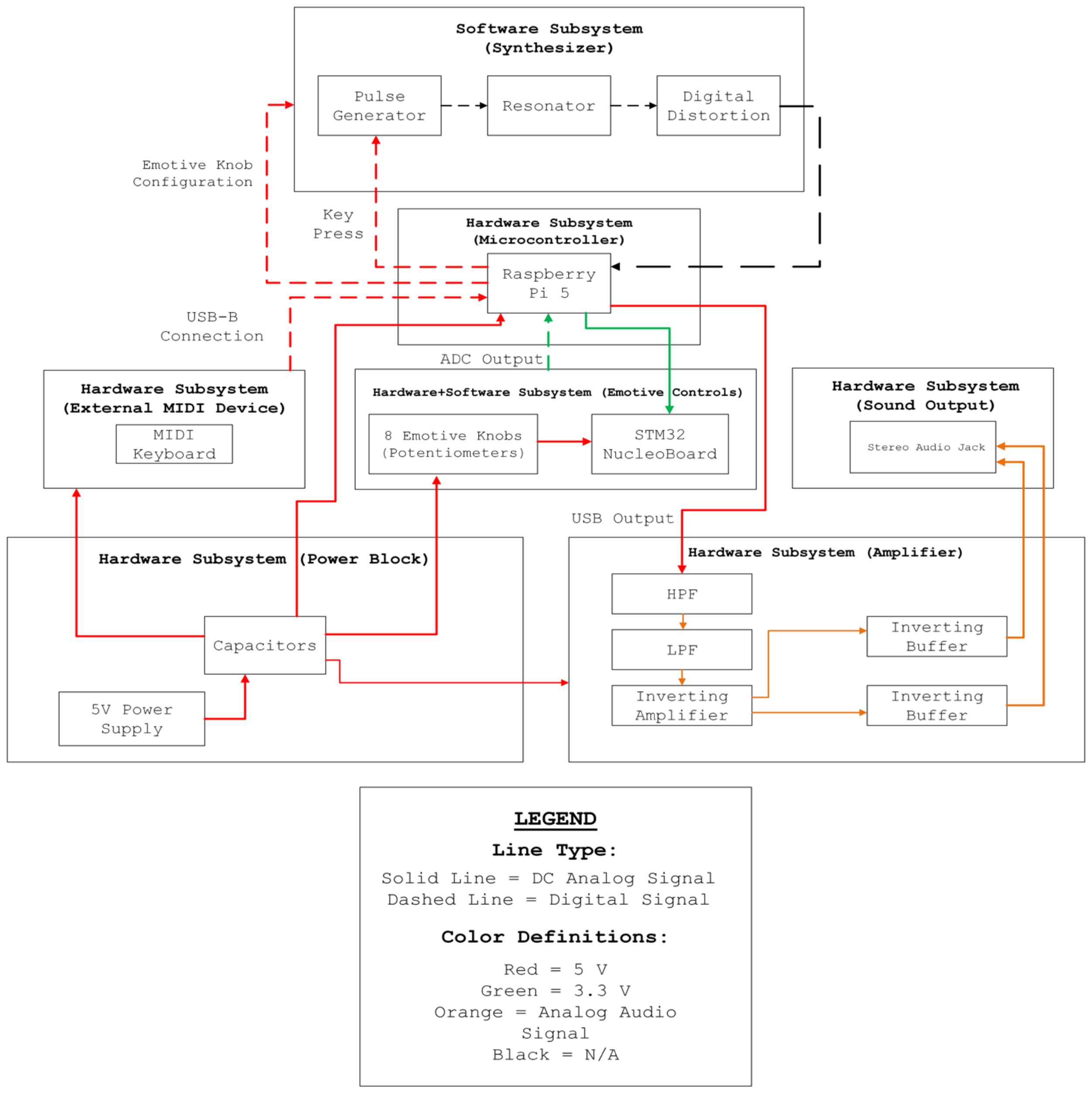


Figure 1 Top-Level Block Diagram

2 Design

2.1 Design Procedure

The synthesizer's design consists of several functional blocks, each selected to meet the project’s goals of cost-effectiveness, portability, and intuitive operation. For the sound generation block, the **Karplus-Strong algorithm** was chosen due to its computational efficiency and ability to produce high-quality plucked string sounds with minimal hardware resources.

Alternative methods like additive or subtractive synthesis were considered but deemed too resource-intensive for a low-cost, portable device.

For the control interface, the synthesizer employs intuitive emotive knobs with descriptive ranges such as "soft to aggressive," implemented using potentiometers connected to an STM32 NucleoBoard. While alternatives like touchscreens or digital encoders were considered, potentiometers were selected for their cost-effectiveness, simplicity, and user-friendly design, ensuring accessibility for non-technical users. These knobs adjust parameters such as filter cutoffs with outputs digitized via the STM32 NucleoBoard's ADC.

The MIDI input block provides users with a familiar, expressive method to trigger sound generation. A wired MIDI keyboard was selected for its balance of simplicity and compatibility with standard devices. However, this proved challenging as the actual MIDI standard is confusing and ultimately required C++ library to use. The hardware platform combines a Raspberry Pi 4 for computationally intensive tasks and an STM32 NucleoBoard for real-time control, chosen over single microcontroller setups or higher-end single-board computers to balance processing power, cost, and modularity.

Finally, the audio output block includes low-pass and high-pass filters and an operational amplifier to ensure high-quality sound. The low-pass filter, with its cutoff frequency calculated as $f_c=12\pi RC$, attenuates frequencies above 20 kHz, ensuring a clear signal. The same equation is valid for the high-pass filter cutoff. Considering these factors, we wanted to create something that was functional and simple to use, such that the user would not need to understand any technical details; this way, our synthesizer will appeal to sound designers who may lack electrical engineering knowledge.

2.2 Design Details

As mentioned before, the detailed design of the synthesizer system integrates multiple subsystems, each playing a critical role in generating and outputting sound. The design process was performed with careful consideration of computational efficiency, sound quality, and user accessibility. The equations/approaches applied throughout the system ensure that the synthesizer remains both powerful and efficient, delivering an expressive, portable tool for sound design. Each component, from the external MIDI device input to the audio output, contributes to a cohesive system that meets the project's goals of creating a cost-effective and flexible synthesizer.

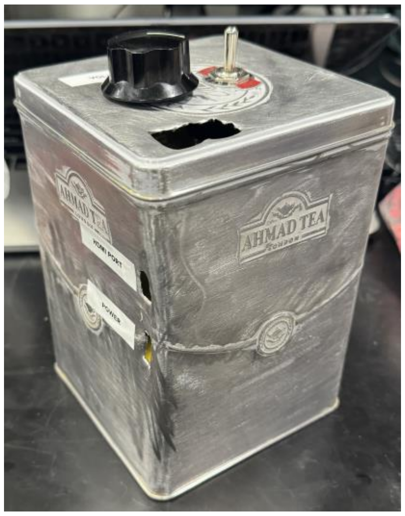
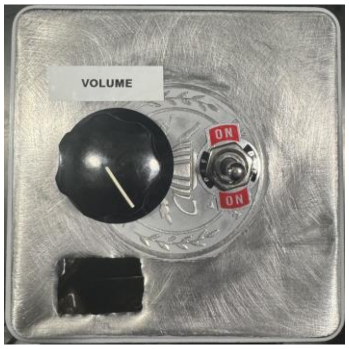
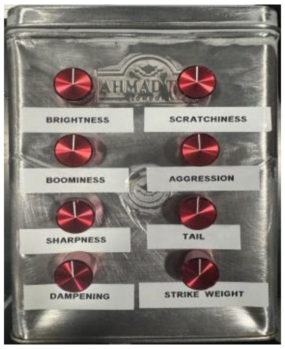
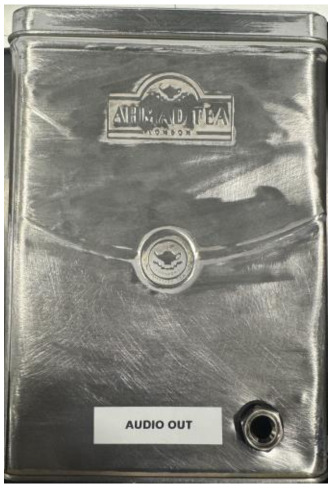
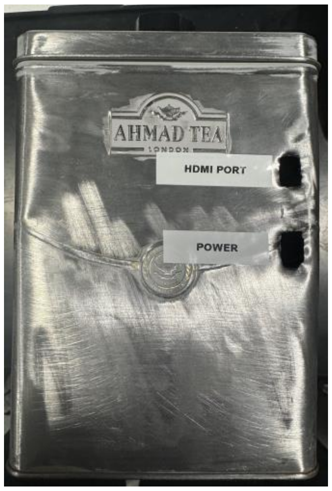
 Full Design	
 Volume Knob, On/Off Switch, and USB Ports	 Emotive Knobs
 Audio Output Port	 HDMI Port + Power Port

Figure 3 Physical Design of String/Drum Synthesizer

Shown in the figure above are the different features of the physical design of our project. To start, the synthesizer itself is contained within a metal box that measures 4.3 x 6.3 x 3.9 inches. The lid of the enclosure has three main components on it: a volume knob to specifically control the sound output, an on/off switch to control the entire system, and USB ports to connect to the external MIDI device. Each side of the container showcases different features of our project: the first side contains the physical knobs to be used in the Emotive Controls subsystem, another side contains the HDMI port for future developments of this project and Power port to be used by the Power Block subsystem, and lastly, an Audio Out port to be used by the Amplifier subsystem.

2.2.1 Synthesizer Subsystem

The Synthesizer Subsystem, which generates sound using the Karplus-Strong algorithm, utilizes a pulse generator that combines sine and noise signals. The echo chamber uses IIR filters to process the sound, including low-pass, high-pass, and bell filters. The filter coefficients are chosen to balance performance and computational efficiency. The synthesizer's parameters, such as pitch and distortion, are controlled using potentiometers in the Emotive Controls subsystem, which convert analog voltages into digital values via the STM32 NucleoBoard’s ADC.

When a note is pressed, we receive a “MIDI Message” which consists of 4 bytes. The first byte signifies the type of message, such as note-on or note-off. We only care about note-on messages since the Karplus-Strong algorithm features a note that rings out on its own. The second byte signifies the pitch as an integer ranging from 0 to 127 where A (440 Hz) is 69. The third byte signifies note velocity, which is how hard the key was pressed (value from 0 to 127). We did not use this value. The fourth byte is also unused for note-on/off messages.

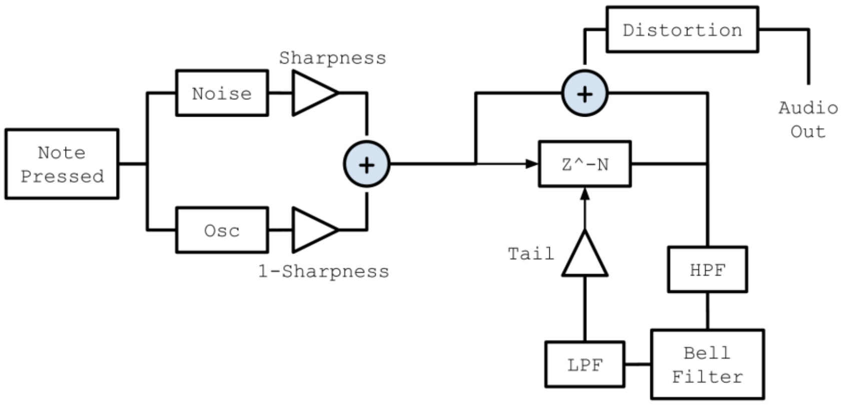


Figure 2 Modified Karplus-Strong Algorithm Block Diagram

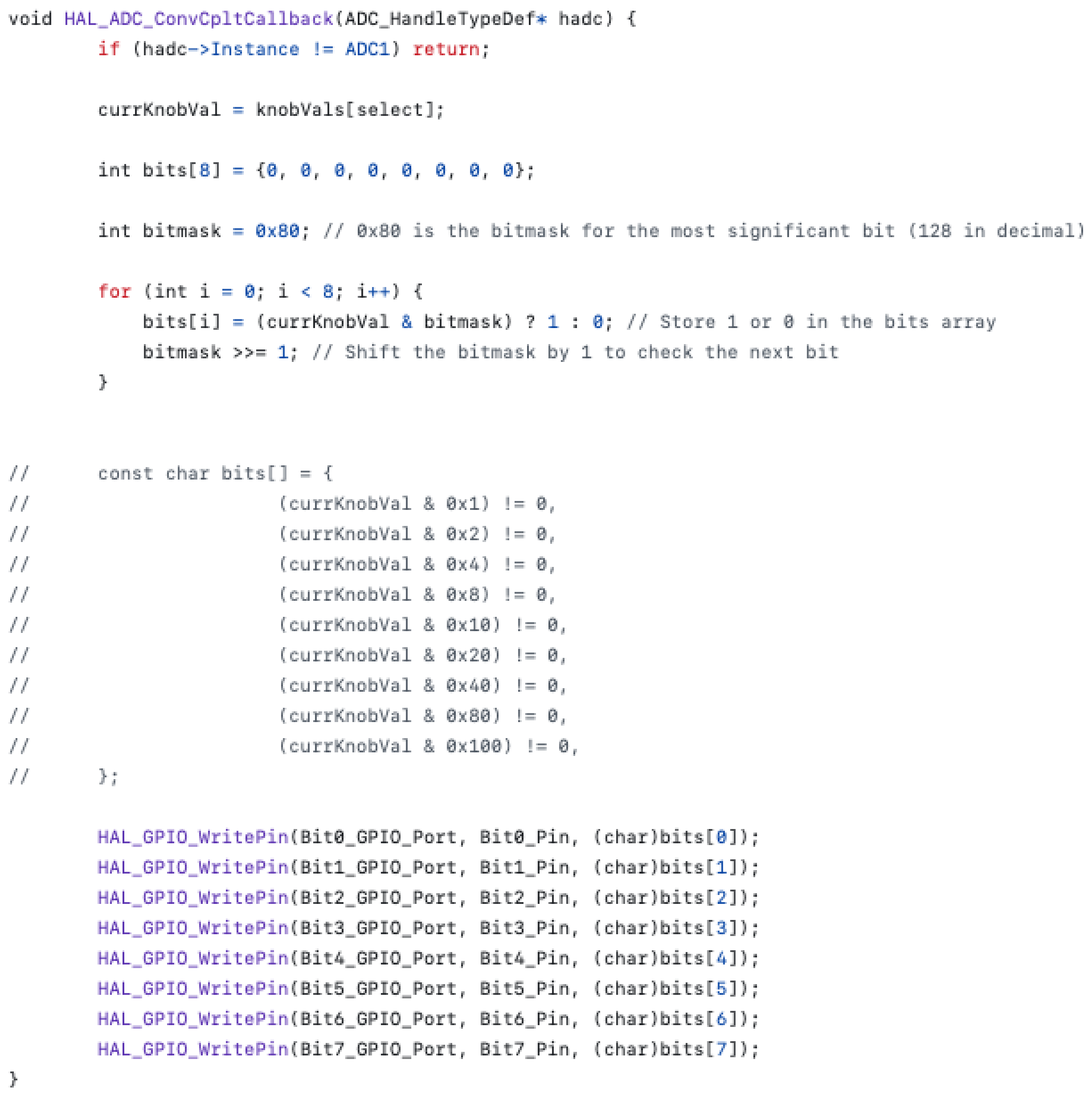
After reading the MIDI message, a short pulse is generated using a mix of white noise and a sine wave oscillator. The short pulse is repeated at the frequency corresponding to the pressed note. If the note that is pressed is an A at 440 Hz, we repeat the pulse 440 times per second. To repeat the pulse, we use a circular buffer which allows us to continuously write samples (i.e. repeat the pulse) while reading old samples. We can also read the old samples with some delay, which in the case of 440 Hz, would be the sample rate 48000 (samples/s) / 440 (1/s) = 109 samples. Each time the pulse is repeated it is filtered using a lowpass, highpass, and peak or bell filter. All three filters are implemented using a biquad filter and changing the filter coefficients. We did not implement the distortion as this is linked to the aggression knob, which did not work in the final product.

2.2.2 Emotive Controls Subsystem

The emotive knobs subsystem provides a way to control the synthesizer parameters using more descriptive names for the knobs. We made a demo of the synthesizer algorithm using a digital audio workstation in order to play with parameters and determine the names of the emotive knobs.

The physical emotive knobs are connected to potentiometers that range between 3.3V and ground. As the user selects values for each of the knobs, their respective voltages are instantaneously sent to the GPIO Pins of the STM32 NucleoBoard. Then, a software program will rapidly retrieve these voltages one by one and pass them through one of the ADCs on the Microcontroller. These now digital values will populate a knob_value buffer, which will be used by the Synthesizer subsystem to make manipulations to the sound the user is building.

The primary equation used for this subsystem came from the method used to separate the binary value from the ADC output for a particular Emotive knob into 8 distinct bits. These 8 bits would then be sent in parallel to the Raspberry Pi for further processing.



```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
        if (hadc->Instance != ADC1) return;

        currKnobVal = knobVals[select];

        int bits[8] = {0, 0, 0, 0, 0, 0, 0, 0};

        int bitmask = 0x80; // 0x80 is the bitmask for the most significant bit (128 in decimal)

        for (int i = 0; i < 8; i++) {
            bits[i] = (currKnobVal & bitmask) ? 1 : 0; // Store 1 or 0 in the bits array
            bitmask >>= 1; // Shift the bitmask by 1 to check the next bit
        }


//      const char bits[] = {
//                      (currKnobVal & 0x1) != 0,
//                      (currKnobVal & 0x2) != 0,
//                      (currKnobVal & 0x4) != 0,
//                      (currKnobVal & 0x8) != 0,
//                      (currKnobVal & 0x10) != 0,
//                      (currKnobVal & 0x20) != 0,
//                      (currKnobVal & 0x40) != 0,
//                      (currKnobVal & 0x80) != 0,
//                      (currKnobVal & 0x100) != 0,
//      };

        HAL_GPIO_WritePin(Bit0_GPIO_Port, Bit0_Pin, (char)bits[0]);
        HAL_GPIO_WritePin(Bit1_GPIO_Port, Bit1_Pin, (char)bits[1]);
        HAL_GPIO_WritePin(Bit2_GPIO_Port, Bit2_Pin, (char)bits[2]);
        HAL_GPIO_WritePin(Bit3_GPIO_Port, Bit3_Pin, (char)bits[3]);
        HAL_GPIO_WritePin(Bit4_GPIO_Port, Bit4_Pin, (char)bits[4]);
        HAL_GPIO_WritePin(Bit5_GPIO_Port, Bit5_Pin, (char)bits[5]);
        HAL_GPIO_WritePin(Bit6_GPIO_Port, Bit6_Pin, (char)bits[6]);
        HAL_GPIO_WritePin(Bit7_GPIO_Port, Bit7_Pin, (char)bits[7]);
}
```

Figure 4 Emotive Controls Code Snippet

Shown above is a section of code from the Emotive Controls code used to send the current value of a knob to the Raspberry Pi. The equation used here is x & y == 1, where "x" is the binary value of the knob being read at a given time, "y" is a bitmask that gets shifted by a power of 2 during every iteration of the loop, "&" is the logical AND operator, and "==" is the relational equals operator. This equation is used to iterate through the bits of an Emotive knob's binary value, check if each bit is either a 1 or a 0, and individually write each bit value to a GPIO pin on the STM32 NucleoBoard. These GPIO output pins will be connected to pins on the Raspberry Pi, so that the processor can have a constant update on the values of each one of the Emotive knobs at a given time.

2.2.3 Amplifier and Sound Output Subsystems

The amplifier will modify the synthesizer's signal provided by the Microcontroller Subsystem so it can be heard with headphones. It receives the audio from the I2S system on the STM32 processor in the form of a PWM wave which must be filtered and amplified and have correct output impedance of 110+/-1% Ohms.

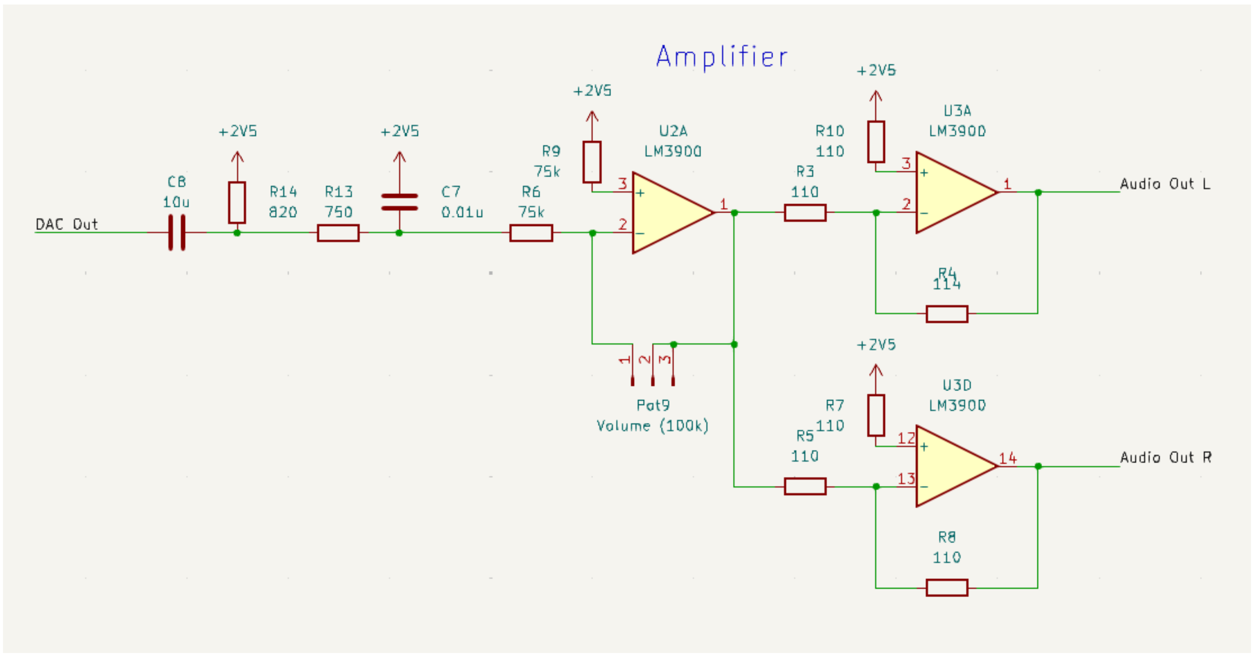


Figure 3 Amplifier Circuit Schematic

From left to right, the amplifier circuit starts with high and low pass filters, which are amplified then split into left and right signals that are connected to the headphone jack. Since the two rightmost op-amps are in the inverting buffer configuration, their output impedances are equal to R4 and R8.

To decide what the output impedance should be, we found that $Z_{Headphones} = 38\ \Omega$. While we could simply impedance match and have a transmission coefficient of 1, we chose to increase the impedance to 110Ω giving a transmission coefficient of 0.5 and a reflection coefficient of -0.5. This allows our circuit to also work well with higher impedance headphones.

2.3 Design Alternatives

We had several design issues such as the microprocessor overheating on our PCB, the DAC/ADC chips not working, and the need to write our own USB MIDI drivers. To address the overheated processor, we switched to using a Raspberry Pi 4 interfaced with an STM32 NucleoBoard for our primary processing. Taking this corrective action allowed for our Synthesizer Algorithm to not only be programmed into our system but also produce an accurate output. Next, to address the circuit with the ADC chip not properly digitizing the Emotive knob values, we switched to utilizing the ADC on the STM32 NucleoBoard. This corrective action served well as it allowed for the values of the Emotive knobs to be updated digitally in real-time to be used by the Synthesizer subsystem. Lastly, we were required to write USB MIDI drivers on the STM32 NucleoBoard. While there is a fair bit of help online for creating a USB Host using the STM32, there is almost no documentation besides the MIDI and USB standards themselves. A USB MIDI host is an audio class host meaning that the data is exchanged continuously. However, MIDI is a special case which would have taken us weeks to implement, understand, and debug. Our corrective action was switching to the Raspberry Pi so we could use RtMidi, a C++ library that uses the native audio and USB drivers of Raspbian (the Raspberry Pi operating system). Overall, we were able to implement a synthesizer on the Raspberry Pi, and we were able to incorporate the emotive knobs into the algorithm but were unable to do so using our PCB or original microprocessor.

3 Design Verification

3.1 Karplus-Strong Algorithm

Each block in the algorithm was implemented one by one, and previous blocks were used to test subsequent blocks. We would have used an oscilloscope, but our amplifier circuit was not working. When probing the headphone output directly from the Raspberry Pi, we did not see much change in voltage implying we had a current output instead. As a result, our verifications were done by ear with some basic sanity checks.

3.1.1 Noise Generator

We used std::uniform_real_distribution to generate white noise sample by sample. To test, we played continuous noise and listened. Our sanity check was that it sounded like the Ocean. We also tried std::normal_distribution as well, but a uniform distribution sounded more like the Ocean and (in theory) provides a much larger frequency spectrum.

3.1.2 Biquad Filters

After implementing noise, we tested our Biquad filters on the noise with varying cutoff frequencies. Our sanity check for the lowpass filter was setting it to 20 Hz and 20 kHz. When the filter was at 20 Hz, we heard nothing since pretty much the entire hearing spectrum is filtered out. Conversely, a 20 kHz cutoff resulted in the same sound of white noise as the unfiltered noise. The check for the highpass filter was the exact same except 20 Hz resulted in the same sound as the unfiltered white noise, and 20 kHz resulted in silence.

The peak filter was harder to verify by ear, but we could verify the Q parameter easily as a high Q results in an overall quieter sound as more energy is removed. In addition, changing the gain parameter also changed the volume. Our sanity checks also included setting the peak filter to a low frequency, low Q, and low gain to remove many low frequencies. This sounded like a mild highpass filter meaning that the low frequencies were simply attenuated a few dB and not completely removed. We repeated this with a high frequency and the same was true expect the results are comparable to a mild lowpass filter.

3.1.3 Delay Line (Z^-N)

To implement the delay line, we created a short burst of noise (based on the scratchiness parameter and the pitch of the sound). Our sanity check for the delay line was that the burst of noise would be repeated at the right pitch (determined by the note pressed on the MIDI keyboard). We played a major scale to verify that the pitches are somewhat accurate relative to each other.

3.1.4 Oscillator

We used std::sin to calculate our oscillator and wrote continuously to the output buffer bypassing all previous DSP. We set it to high pitches such as 10 kHz and low pitches such as 100 Hz for our sanity check. High pitches sounded higher than low pitches, however we were unable to verify the accuracy beyond that.

3.1.4 MIDI Input
We printed the bytes of the received MIDI messages to the console of the Raspberry Pi. We used the raw values to decode which byte meant note-on and which byte was related to the pitch. When pressing the key, the first byte always reads 144 and when releasing, it reads 128. Neighboring notes such as C and C# differed by 1 in their second byte.

3.2 Emotive Controls
For the Emotive Controls subsystem, we needed to test its functionality from both a hardware as well as software perspective.

3.2.1 The Hardware
After connecting the potentiometers to the enclosure and connecting the pins of the potentiometers to 3.3 V, ground, and data, we verified whether the value of the data pin changed as the knob was turned and that its value would range from 0 to 3.3 V. To do this, we utilized a multimeter and connected it to the data pin. When turned in one direction, the data pin was 0 V and when turned in the other direction it read 3.3 V. We also verified that the voltage rose as we turned the knob.

3.2.2 The Software
Next, we verified that the knob values were being updated within the STM32 NucleoBoard in real-time by connecting each of the data pins of the knobs to different GPIO pins on the board configured as ADC's. In the main program, we created a global array to store the values being held in the ADC, using the line "HAL_ADC_Start_DMA(&hadc1, (uint32_t*)knobVals, 8)" in a while loop to constantly transfer the contents of the ADC into this array. Using GDB debugger, we checked the contents of the global array every time the program reached the breakpoint correspond to reading the ADC values. In between each run of the code prior to reaching the next breakpoint, we turned the knobs in different directions to see if the values in the buffer would change. After proving that all 8 knobs were changing in value after each cycle, we were able to verify the initial claim that the value of the knobs being held in the NucleoBoard were updating in real-time.

3.2.3 Discussion of Requirement & Verification Table
Please reference Table 1 in Appendix A to view the Requirements & Verification Table that was created in the early stages of this project for this subsystem.

After revisiting our R&V table for this subsystem and examining the procedures described above, we can see that this subsystem accomplished what it was intended to.

3.3 Amplifier
The Amplifier subsystem was developed using an oscilloscope to view the waveform at all stages.

3.3.1 Highpass and Lowpass Filters
We compared an input sine wave to the output of the two filters. We swept the sine wave from 20 Hz to 20 kHz and saw that the amplitude of the output remained constant as the frequency

increased. This suggests that the passband of the filters allow the entire hearing spectrum to pass.

3.3.2 Gain Stage
The gain stage featured a volume knob and an inverting amplifier. The expected output on the oscilloscope is an inverted wave that attenuates to 0 as the knob is turned. In addition, the signal was amplified when the knob was turned all the way up.

3.3.3 Output Buffer
The output buffer inverted the signal but did not affect the gain. We had no way of verifying the impedance.

4 Cost & Schedule

4.1 Component Costs

Table 1 Component Costs

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
Voltage Regulator (LM1117)	Texas Instruments	$1.15	$2.30	$2.30
Op-Amp (MC33097)	ST Microelectronics	$0.00	$0.00	$0.00
Microprocessor (STM32F722RCT6)	ST Microelectronics	$11.20	$22.40	$22.40
1/4" Audio Jack	Amphenol Audio	$2.43	$4.86	$4.86
USB-A Connector	On Shore Technology Inc.	$0.68	$1.36	$1.36
HDMI Port	Amphenol ICC (FCI)	$1.50	$3.00	$3.00
DC Power Jack	GlobTek, Inc.	$0.64	$1.92	$1.92
5V Power Supply Adapter	Tri-Mag, LLC	$5.31	$5.31	$5.31
Video Transmitter	Lattice Semiconductor Corporation	$10.10	$20.20	$20.20
STM32-NucleoBoard F401-RE	ST Microelectronics	$24.20	$24.20	$24.20
Raspberry Pi 4	Raspberry Pi	$45.00	$45.00	$45.00
Total	**-**	**$102.21**	**$130.55**	**$130.55**

4.2 Schedule

Consult Appendix B to see a detailed weekly schedule of our project.

5 Conclusion

5.1 Accomplishments

Although our project didn’t fully function as intended, we still had a lot of accomplishments from a subsystem perspective. Starting with the Power Block subsystem, we were able to produce a stable 5 Volts and 3.3 Volts for the entire system as we initially intended. Next, in the External MIDI Device subsystem, we were able to retrieve data from the keyboard upon keypress, such as the pitch of the key, with very low latency. Moving on to the Emotive Controls subsystem, we were able to both convert the analog values of the knobs into digital ones and also, send these values to the main processor within the Raspberry Pi as they were being updated in real-time. In terms of the final functionality, we were able to get our modified Karplus-Strong Algorithm working in practice as we could hear sound after each keypress and also tune the sound to our liking using the physical Emotive knobs.

5.2 Uncertainties

Unfortunately, our project did encounter some unsatisfactory results. Starting with the Amplifier subsystem, we weren’t able to produce a sound output from the “Audio Out” port and produce the correct impedance during our final demonstration. Also, when connecting an oscilloscope to this port, the signal did not match that of the sound we were generating. We have two assumptions as to why this was the case. The first being that a chip within the Amplifier circuit was fried and thus, was causing the rest of the circuit to malfunction. The second being that the entire circuit configuration itself was incorrect and needed to be readjusted.

5.3 Ethical Considerations

One ethical concern in this project is the potential for loud noises or ringing pitches that could cause hearing damage, as outlined in IEEE Section 7.8, Rule 1. To address this, we will ensure that the synthesizer’s volume is always started at a low level and gradually increased during testing. The final product will be designed to avoid unintended loud noises or harmful ringing pitches, mitigating the risk of hearing damage.

Another ethical concern is the possibility of producing sounds that already exist, which could lead to copyright infringement, as per IEEE Section 7.8, Rule 5. To resolve this, we will advise users to submit their music to the United States Copyright Office before publishing their work. This step serves as a formal recognition of their creation and provides legal protection against potential copyright infringement claims.

5.4 Future Work

The enclosure we chose was too small to fit all our components. We would like to transfer the components of our prototype to a larger enclosure for a cleaner design.

The Amplifier Subsystem did not end up working or providing the correct impedance. As this is crucial to the quality of the sound, we would make sure it meets specs. Also, we could add a switch to change the output impedance, since the ideal impedance changes based on what we plug the synthesizer into.

We were unable to implement all 8 Emotive Knobs. The Aggression knob still needs to be implemented into the Synthesizer Algorithm, but we can get rid of the Tail knob entirely. The length of the output sound is affected by Dampening, Brightness, Boominess, and Scratchiness as these all affect the energy in the system. As such Tail is redundant.

In addition, we would like to build the Screen subsystem such that the DSP output can visualized in real-time to allow users to have a better understanding of how their sound is being developed. We could display a few periods of the waveform as well as the spectrum. We could also display a virtual version of each knob to show the user how it's being turned.

References

[1] “IEEE Code of Ethics.” IEEE - Advancing Technology for Humanity, June 2020, www.ieee.org/about/corporate/governance/p7-8.html.

[2] “Professional Monitor Headphones: ATH-M50X.” Audio-Technica, www.audio-technica.com/en-us/ath-m50x.

[3] Dengate, Paul. “When Milliseconds Matter: The Reality behind Latency in Networked AV.” HARMAN Professional Solutions Insights, 23 Sept. 2022, pro.harman.com/insights/harman-pro/when-milliseconds-matter-the-reality-behind-latency-in-networked-av/#:~:text=Most%20experts%20a gree%20that%20visual,and%20the%20user’s%20visual%20acuity.

[4] Purves, Dale. “The Audible Spectrum.” Neuroscience. 2nd Edition., U.S. National Library of Medicine, 1 Jan. 1970, www.ncbi.nlm.nih.gov/books/NBK10924/#:~:text=Humans%20can%20detec t%20sounds%20in,to%2015%E2%80%9317%20kHz.

Appendix A. Requirement and Verification Tables

Table 2 Emotive Controls Subsystem Requirement and Verification

Requirement	Verification	Verification status (Y or N)
1. Processor correctly reads the value of all potentiometers.	1. A) Turn one potentiometer and print the value on a computer screen/debugger. B) Repeat with the other potentiometers.	Y

Appendix B. Schedule

Table 3 Detailed Weekly Project Schedule

Week	Task	Person
10/07 to 10/11	Prepared for Design Review Presentation	Everyone
	Attended and filled out form for Peer Design Review Presentation	Everyone
	Decided on what components to order for our project	Everyone
10/14 to 10/18	Ordered components	Abhi
	Worked on Teamwork Evaluation Form	Everyone
	Started building subsystem circuit schematics	Joel
10/21 to 10/25	Fixed errors in circuit schematics (Screen Subsystem)	Abhi
	Fixed traces in PCB design	Joel
10/28 to 11/01	Finalized PCB design	Everyone
	Placed PCB order	Joel

11/04 to 11/08	Worked on Individual Progress Report	Abhi
	Fixed the Design Document based on TA feedback	Everyone
	Created a physical circuit for Amplifier subsystem to verify functionality	Joel
11/11 to 11/15	Soldered Components to the PCB	Joel
	Tested Amplifier and Power Block Subsystems on PCB to check functionality	Joel
	Tested Microcontroller and Emotive Control Subsystems on PCB to check functionality	Everyone
	Started reading through STM32 Documentation	Abhi
	Started writing a test script to check if the Microcontroller is working (writing "Hello World!" to the console and sending a voltage to a GPIO pin)	Abhi
11/18 to 11/22	Finished building Power Block and Amplifier Subsystems	Joel
	Created double buffer system for Audio to output with I2S protocol on STM32 microprocessor	Joel
	Finished writing the test script and learning how to program the STM32 microprocessor	Abhi
	Mock Demo	Both
11/25 to 11/29	Fall Break (Worked on the Synthesizer algorithm and wrote code for reading Emotive Knob	Joel

	digital inputs from STM32 NucleoBoard)	
12/02 to 12/06	Built Enclosure for Synthesizer	Everyone
	Wrote script on STM32 to convert analog Emotive Knob values to be digital inputs to the Raspberry Pi	Everyone
	Finished Synthesizer Subsystem on the Raspberry Pi	Joel
	Finalized Design Document with all new changes	Abhi
	Final Demo	Everyone
	Attended and filled out form for Final Demo Peer Review	Everyone
	Worked on Final Presentation Slide Deck	Everyone
12/09 to 12/13	Final Presentation	Everyone
	Attended and filled out form for Final Presentation Peer Review	Everyone
	Completed Final Report	Everyone
	Cleaned up Lab Notebook for Submission	Everyone
	Lab Checkout with TA	Everyone