Simulating and Augmenting Melodica Performance on an Embedded System

Matt Tu

Advisor: Scott Peterson

Abstract: The field of computer music has grown significantly in the last half century due to rapid advances in computer hardware and incresaingly robust audio software. At the heart of computer music is digital audio synthesis, where programmers use a wide range of techniques to produce realistic sounds. A common application of digital audio synthesis is recreating the sounds of musical instruments. In this project, we aimed to digitally recreate the sound of the melodica, which is a musical instrument that combines a keyboard and a mouthpiece, and furthermore create a system that can simulate the physical performance of a melodica. Initially, the sonic profile of the melodica was analyzed using fourier spectrum analysis, in which harmonic frequencies and their amplitudes were recorded. We then used sound synthesis techniques, such as additive synthesis and filters, to digitally recreate its sonic qualities, and designed and built an embedded system that uses different sensors and input sources to emulate the operation and performance of a real melodica. The resulting digital melodica system's sound closely resembled that of a real melodica, and the user's interaction with the system was almost identical to that of the original instrument. Moreover, we developed additional features that expanded the musical capability of the electronic melodica system, including programming the melodica mouthpiece to modulate frequency instead of amplitude, and adding a brass-like synthesizer. Overall, this project gave more insight into the sonic profile of the melodica and explored the computational bounds of sound synthesis on an embedded system with limited resources.

INTRODUCTION

Computer music has seen substantial growth over the last half century due to advances in computer hardware and software. In particular, audio synthesis technologies have evolved over the course of the last forty years from hybrid systems, where digital controllers adjusted analog synthesizers, to fully digitalized systems with real-time input and synthesis [14] aided by protocols like MIDI [1] and synthesis software platforms like Supercollider [5]. Today, it is simple to apply these technologies and techniques to produce high-quality, realistic sound for a wide variety of artistic, academic and commercial purposes.

A specific application of computer music and digital audio synthesis is to analyze and reproduce the sound of physical musical instruments. The timbres and sound qualities of instruments such as the piano or violin have been extensively researched, analyzed, and digitally synthesized. This has made the development of high-fidelity electronic instruments possible, with the digital piano as an important example [3]. Not only do these instruments perform and sound in a similar fashion, but also have more musical features programmed into the system than their physical counterparts.

This project researches and explores this domain of audio synthesis and system design to recreate a digital version of the melodica. A melodica is a small instrument with a musical keyboard and a mouthpiece. Blowing into the mouthpiece passes air into the body of the instrument and produces sound based on which keys are pressed down and how much air is being blown into the mouthpiece [15]. The timbre of the melodica hasn't been as widely analyzed as the piano or other more well-known instruments, so a major focus of the project is to analyze and faithfully reproduce its sound.

BACKGROUND

Sound Synthesis Technique and Tools

Physically speaking, sound is a vibration which propagates as acoustic waves. Waves are periodic in nature, meaning sound can be modeled as a periodic signal that can change over time. Therefore, any audio signal can be expressed as the sum of sine and cosine waves, according to the Fourier series [2]. This is the basis of additive synthesis, one of the earliest and still widely-used synthesis techniques. Since every sound, from ambient noises to notes on an instrument, are composed of various sine frequencies at different amplitudes, these sounds can be

recreated by adding various sine oscillators together and outputting the resulting signal through a speaker. Typically, a Fourier transform, which converts a signal from time domain to frequency domain, is performed on a sound sample to determine which sine frequencies and amplitudes to use for additive synthesis [8].

Additive synthesis is combined with other synthesis techniques to produce sound in synthesizers and electronic musical instruments. Among the other synthesis techniques, modulation synthesis, notably frequency modulation, is widely used. In frequency modulation, the frequency of a waveform is changed by modulating its frequency with another modulator, usually a periodic oscillator. The original frequency of the waveform is called the "carrier frequency," while the frequency of the modulating signal is called the "modulator frequency". When the modulator frequency is very small compared to the carrier frequency, the result might sound like a vibrato effect, but once the modulator frequency is within a similar order of magnitude to the carrier frequency, audible sideband frequencies of the sums and differences of the two frequencies can be heard. This can result in harmonic and non-harmonic sounds based on the specific ratio of the frequencies. Moreover, the number of sidebands depends on the relative amplitude of the modulating signal, known as the index of modulation [9].

There are other auxiliary synthesis techniques that help achieve more realistic sounds. A few notable examples include filters, which utilize feedback to emphasize or mitigate different ranges of frequencies. Additionally, reverberation, a useful tool for simulating a physical space for sound to exist in, can be accomplished using convolution kernels [17]. Overall, the existence of all these techniques illustrates that there is no one formula or algorithm that can be used to create realistic, natural-like sound. Most often, it is a matter of iterating until a satisfactory result is achieved.

Embedded Systems

We decided to model the electronic melodica's system design after an embedded system. Embedded systems cannot run any software other than the software already installed and running. Since the electronic melodica will only function as a melodica, this makes the programming model much simpler. Additionally, certain embedded systems fall into the subset real-time systems, meaning that they have actions that must be performed within a certain timeframe. Our electronic melodica system also falls into this category, since audio processing is extremely time sensitive. Moreover, embedded systems usually have fixed input-output, which

aligns with this project, as a musical instrument has well-defined inputs and outputs. Also, the electronic melodica should be completely wireless and require no external power, and instead use a battery to power the systems. Many embedded systems share this constraint as well, as they are used in automobiles, MP3 players, cellular phones, and other machinery and devices without a constant power source [4].

SOUND ANALYSIS

To analyze the sonic profile of the melodica, each note on a 37-key Eastrock Melodica [18] was played and recorded. The resulting recording was imported to an Audacity program, and fast Fourier transform (FFT) spectrum analysis was performed on each of the notes. Figure 1 shows a sample of one spectrum plot, with amplitude peaks at different harmonic frequencies. The full results, including peak decibel values for the 1-12th harmonics are linked in the appendix.

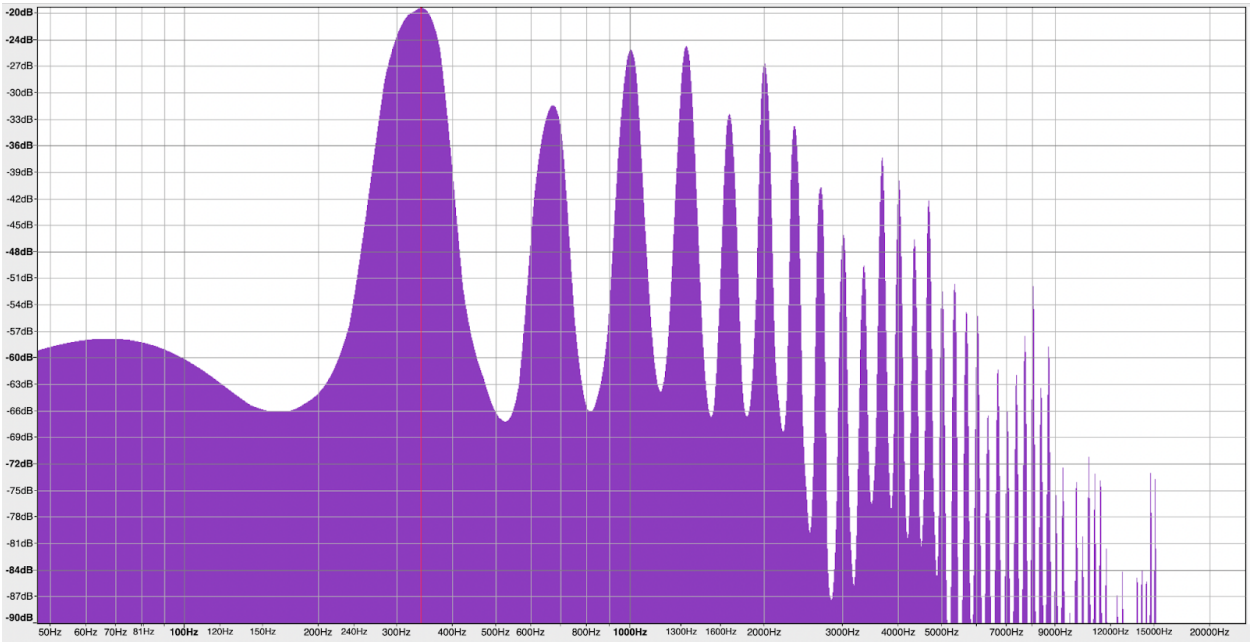

Figure 1: The frequency spectrum plot of a the Eastrock brand Melodica playing an F4 note

After careful examination, the relative loudness of the harmonics did not follow a set pattern. Even adjacent notes with similar fundamental frequencies sometimes had spectra with significantly different shapes. This phenomenon is consistent with findings from other instruments, as many physical factors of each instrument manifest in different ways [3].

SYSTEM DESIGN

Hardware

To simulate the mouthpiece of the melodica, a simple breath controller was built. The breath controller's design was inspired by a similar USB MIDI Breath Controller project on a DIY electronic project hub [11]. Specifically, a board-mounted pressure sensor, able to handle readings from 0.00 to 0.87 PSI, was attached to the end of a melodica hose. Since the melodica hose formed an air-tight seal around the pressure sensor, blowing on the melodica hose would increase the pressure reading up to its upper limit. However, to mimic the gradual loss of pressure due to passing air in the instrument, a small hole was drilled into the side of the melodica hose, allowing for air to slowly escape during performance. To test how the output ranges of the sensor aligned with human breath, we initially used an Arduino Uno [6] to observe and record readings of the sensor based on different breath intensities. The observations indicated that with no blowing, the reading from the analog IO port ranged around ~0.15, and at the highest intensity of blowing, the IO reading managed to almost reach 1.00, the maximum reading. Thus, this confirmed that the sensor and particular configuration would be able to meet our needs. The figure below illustrates more details about the breath controller.

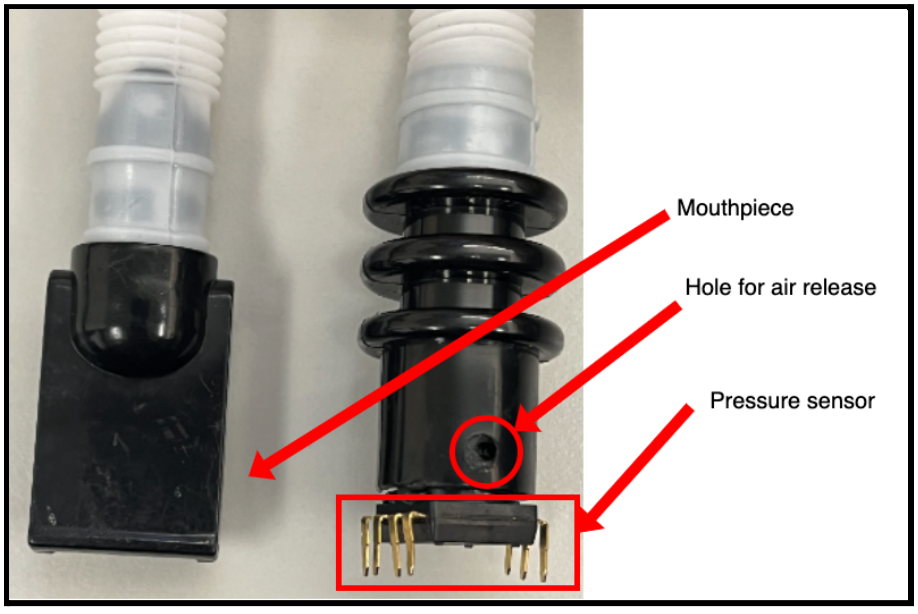

Figure 2: Breath controller diagram

In order to receive digital musical note input, a generic 33-key electronic keyboard was used. This keyboard supported the MIDI interface [1], which among other features included sending note-on and note-off messages to a USB host, making it possible to determine which set

of notes were being held down at a given moment. The initial testing of the keyboard was done by connecting it to a PC laptop. The latency from the note being pressed down to the computer receiving the MIDI message was negligible, suggesting that the default MIDI interface would be an ideal method to pass information about the state of the keyboard to the embedded system.

After testing the two main input sources of the system, the next main question was to determine what embedded platform to use for the processing of that input in addition to the eventual handling of audio output. The available options ranged from Arduino-based microcontrollers running single binaires [6] to complete computers like Raspberry Pi's running entire operating systems [16]. Ultimately, we ended up choosing the Bela platform, an embedded computing platform specializing in high-quality, low-latency audio output. Although built on top of a complete linux kernel, all of Bela's audio-related processes are automatically the highest priority in the system, allowing for significantly reduced latency [7]. Additionally, the platform supported various types of IO, including a native USB port which was important for connecting the MIDI keyboard [13]. The platform also included a built-in IDE that was accessed from a web browser, allowing for fast development and testing of code [12].

For audio output, the Bela physical board provided 2 audio output channels for stereo sound. While testing on the embedded Bela platform, we connected a pair of stereo headphones to the output channels using an adapter. However, the ultimate objective was to incorporate speakers that were powered directly by the embedded system. For this, we relied on the Bela board's built-in speaker amps. The speaker outputs support up to 1 Watt speakers with 8 Ohm impedance, but can only function if the board is connected to a DC power supply of 5 Volts [13]. To deliver exactly that amount, we used two 3.7V Lithium ion batteries in series, and attached an IC 7805 voltage regulator to the circuit [10]. The two batteries provide a raw voltage of around 8 volts when fully charged, which was reduced to between 4.8 to 5.2 volts by the regulator. Lastly, the ground and output voltage wires were connected to a barrel plug adapter, which was inserted into the barrel socket of the Bela board. The following figure illustrates the circuit that powers the speaker, in addition to the rest of the board.

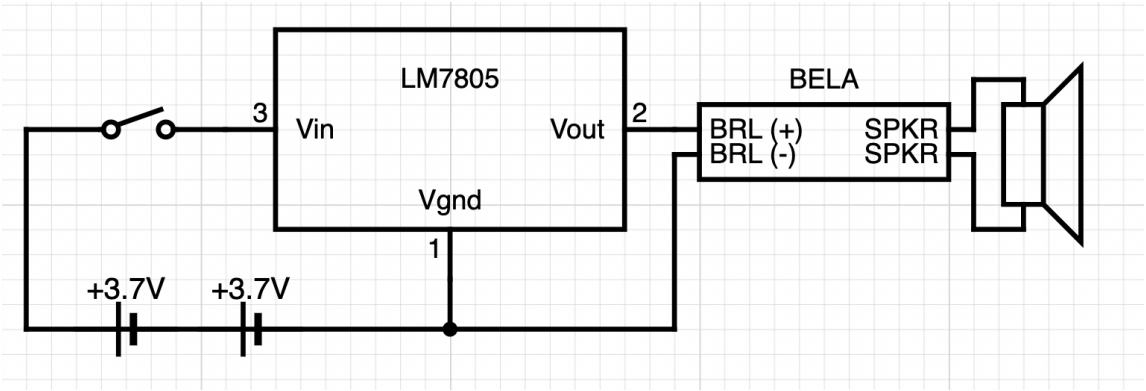

Figure 3: Speaker and battery circuit. The switch powers the Bela board on and off.

Software

Among the various audio-centric software platforms available, we chose to program in Supercollider, a feature-rich, end-to-end platform for audio synthesis. Supercollider contains three main components - “sclang”- the interpretive programming language used for defining unit generators, and other Supercolldier entities, “scide” - an IDE for sclang with extensive interactive documentation, and “scsynth” - the actual server that produces audio from sclang code [5]. The majority of the development phase was done on a main PC, as the Supercollider IDE provided the capability to instantly reinterpret, test and “listen” to partial modifications to the code. Once a new feature or sound was implemented on the PC, the code was then transferred to the embedded platform.

The Bela platform supported and included scsynth and sclang natively, so there was no extra process of installing libraries or dependencies, but there were a few significant differences in the embedded environment. For instance, the IDE did not support partial reinterpretation of the code, and instead had to run the entire file from start to finish, which made it necessary to refactor the code as if it was compiled and run once. Additionally, interfacing with the IO was slightly different. Since the Bela board included native analog IO pins, we could use Supercollider to directly poll the values as a unit generator. As a result, we used the PC platform to build and test new features, and then spent time transferring and refactoring that code to be compatible with the embedded Bela platform.

SYSTEM IMPLEMENTATION

Additive Synthesis

As mentioned above, we used the Supercollider platform to synthesize audio for the electronic melodica. The main technique used was additive synthesis, in which the output signal was produced by summing up sine signals at different frequencies and amplitudes. For each note of the melodica, the frequencies of the sine oscillators were multiples of the note's fundamental frequency, and the amplitudes were calculated by converting the decibel values from the FFT spectrum analysis of that note into relative amplitudes (see Sound Analysis section). The analysis data was formatted into a CSV file, allowing Supercollider to read and convert it into an array. Each element of the array was an amplitude envelope, so that the amplitudes of pitches that weren't explicitly recorded were either interpolated or extrapolated.

In Supercollider, both frequency and amplitude values were stored in two separate array data structures. Because of multi-channel expansion in Supercollider, passing in both arrays into a `SinOsc` function created one sine oscillator unit generator per frequency amplitude pair. These inputs were further passed into the `Mix` unit generator to be combined into one output. All of the synthesis code was converted into a SynthDef - a blueprint for a synth which after definition, can be used to create synth objects synchronously on demand. In order to prevent the total output signal at any given time exceeding an amplitude of 1.0, each amplitude was divided by the number of notes that were being held down.

Keyboard Integration

To add support for the keyboard, we initialized Supercollider's MIDIClient class, which provided access to the Bela platform's MIDI subsystem. Following initialization, the MIDIIn class was used to connect to all available MIDI in devices, namely the keyboard. Once connected, Supercollider was able to receive asynchronous events whenever a key was pressed down or released. The note press event handler determined which note was pressed, and then immediately spawned a Synth whose fundamental frequency was the frequently associated with the MIDI note number, and amplitude array was calculated by iterating through each envelope in the array, and using the note number to determine what amplitude value should be associated with each harmonic frequency. The newly created Synth would then be added to a dictionary, with the key being the MIDI note number. This way, the note release event handler would then use the dictionary to retrieve the Synth at that note value, and then free the Synth, ending the

sound and releasing the memory used by that Synth. Altogether, when a key is pressed down, Supercollider would spawn a Synth to play that note, and stop playing the note once the key was released.

Breath Controller Implementation

When testing the Supercollider code on the PC, the breath controller's analog out pin was connected to the Arduino microcontroller, which had to normalize and output the readings to a serial out port. Supercollider was then used to create a unit generator from that serial out port, generating a sequence of values between ~0.15 (no breath) to 1 (max breath). The values were then mapped to the 0-1 range, and the amplitude of the entire sound of a synth was multiplied to that variable. This way, when there was no air blowing into the mouthpiece, the "breath" variable would be zero, and the resulting sound from any Synths would be none, even if the keys were being pressed down. Conversely, if the "breath" value was nonzero and no keys were pressed, no sound was made. Only when keys were pressed down and there was air blowing into the mouthpiece did the Synths produce audible sound. Additionally, the more pressure that was applied to the sensor by blowing, the higher the overall amplitudes were, contributing to a louder sound.

Furthermore, with all musical instruments, the louder an instrument is played, the "brighter" the sound gets, since the amplitudes of the higher harmonics increase. To simulate this change in tonal quality, we also passed the mixed sound through a low pass filter before outputting it. In Supercollider, the `LPF` unit generator is a second order Butterworth low pass filter with a cutoff frequency parameter [5]. Similar to what we did with the overall amplitude, the "breath" variable was used to modulate the cutoff frequency - as the breath increased, the cutoff frequency increased as well, allowing more high frequencies to be heard.

Modulating Pitch instead of Sound

Since one of the main objectives was to build more features and expand the musical functionality of the electronic melodica, we developed a new function to the mouthpiece of the system. In addition to modulating the amplitude (loudness) of the notes being pressed down on the keyboard, the mouthpiece could also modulate the frequency of the notes instead. This was accomplished by linearly mapping the breath output from 0-1 to 1-2 and then multiplying the base frequency associated with each note by that value. This resulted in a glissando effect as more or less pressure was applied to the mouthpiece. Furthermore, in order to discretize the new

frequencies to the nearest musical half-step, the resulting frequency was converted into a MIDI note, rounded to the nearest whole number, and converted back to a frequency value.

Different Sound Profiles

In addition to altering the function of the mouthpiece, we also programmed and included another SynthDef, which had a sonic profile similar to a brass instrument. To create a brass-like sound, we used mainly frequency modulation, with some added microrandomness to the base frequency and overall amplitude of the sound. The modulator and frequency and carrier frequencies were equal, resulting in harmonic sideband frequencies. Moreover, the breath variable also controlled the index of modulation in addition to the overall amplitude of the sound. This way, as the sound got louder, it also got brighter due to the higher index of modulation enabling higher-frequency sidebands. The mouthpiece was also able modulate either amplitude or frequency like its melodica SynthDef counterpart.

SYSTEM DIAGRAM

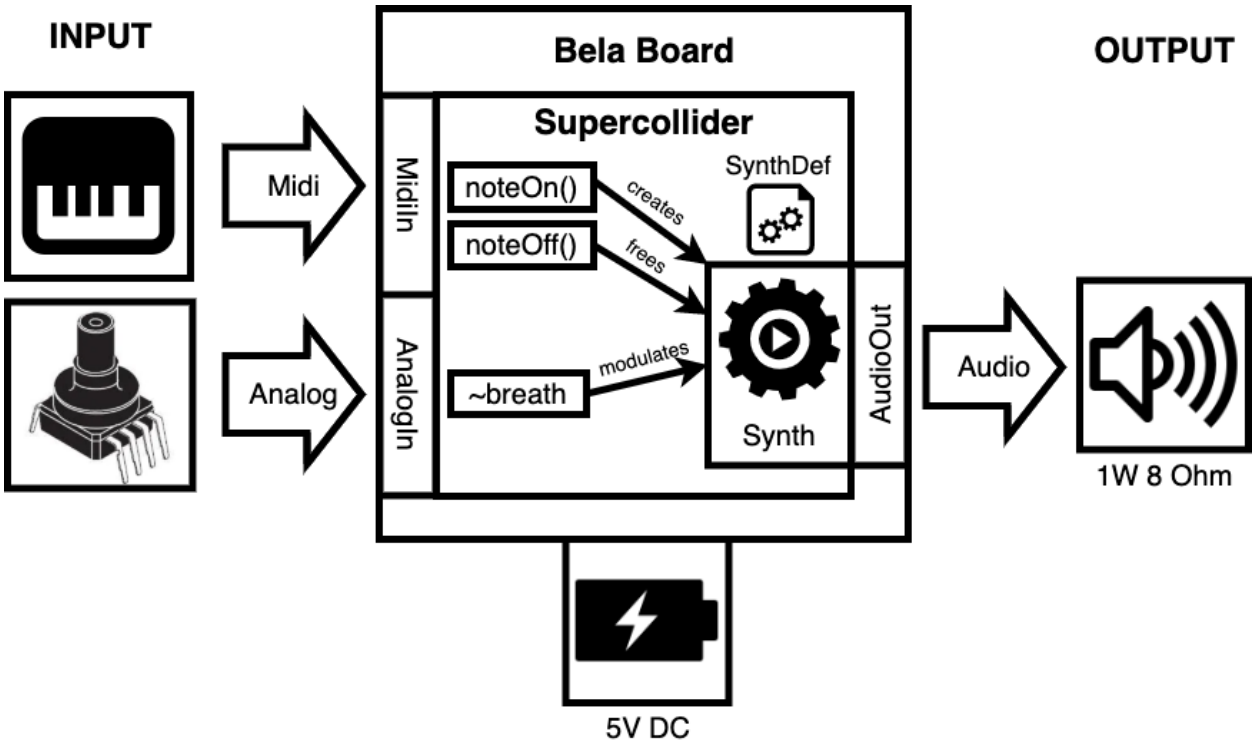

Figure 4: High-level system diagram of the electronic melodica system. The inputs include an analog input from a pressure sensor, and MIDI input from a keyboard. The Supercollider platform, which runs on the Bela board (powered by a 5V battery), processes the inputs, creating,

freeing and modifying Synths (based on predefined SynthDefs). The Synth objects produce audio output, which is sent to the speakers.

RESULTS AND LIMITATIONS

Overall, the completed electronic melodica system was able to produce sounds closely resembling a real melodica. The 33-key keyboard and breath controller made out of the same melodica hose provided a playing experience that was nearly identical to performing on a real melodica. Furthermore, the additional features, including the ability to change what sound property the mouthpiece modulated and a new SynthDef, enabled the electronic melodica to perform musical feats that a normal melodica would be physically unable to. It would be very straightforward to develop additional features on this embedded system, such as more SynthDefs for the user to select from, and different ways that the breath controller can modulate the audio output. A comprehensive video, which includes an overview of the system design in addition to a demonstration of the electronic melodica being performed, is linked in the appendix section.

However, there were also several noteworthy limitations to the project. Most of these limitations were tied to the processing capabilities of the Bela board. For instance, the additive synthesis we used to recreate the melodica sound was limited to the 1st through 12th harmonics, totalling 12 amplitude readings per note. Although some notes when played and analyzed had high amplitudes tied to their 13th harmonic frequencies and higher, we suspected that the Bela CPU would potentially have trouble handling more sine oscillators per note. Similarly, we experimentally discovered that the most amount of notes that could be played at a time was around 6-7. Thus, there was a tradeoff between how many harmonic frequencies additive synthesis could use and the number of notes that could be played at once. In addition, the speakers that the Bela board could drive were only a maximum of 1 Watt. Although the speaker provided ample loudness for a small room, the loudness of a real melodica is noticeably greater. However, this limitation can be remedied by including a discrete speaker amplifier.

CONCLUSION

In this project, we have successfully designed, built, and ultimately performed on an electronic melodica system that sounds and performs similarly to a real melodica. In addition, the electronic melodica system features musical capabilities that cannot be done on regular

melodicas, and there is potential to develop even more capabilities on the embedded platform. Through the process, we gained insight into the sonic properties of the melodica, which has not been thoroughly studied, and used several industry-standard synthesis techniques to recreate its sound. Although there are limitations to the system, it is feasible that they can be resolved with improved hardware in future iterations.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Scott Peterson, whose expertise in computer music and embedded systems helped me tremendously throughout the project. Additionally, I would like to thank the Computer Science Department at Yale University, especially my current and former professors, and the faculty responsible for managing CPSC 490. Lastly, I would like to thank my suitemates, who dealt up with the loud sounds coming from my room while working on the project.

APPENDIX

Demonstration Video

Melodica Frequency Spectrum Analysis CSV

- Note: First column indicates MIDI note number, columns 2-13 indicate loudness (dB) at 1st through 12th harmonic

Source Code (for Bela board)

References

1. Anderton, Craig. "The MIDI protocol." *Audio Engineering Society Conference: 5th International Conference: Music and Digital Technology*. Audio Engineering Society, 1987.
2. Bracewell, Ronald Newbold, and Ronald N. Bracewell. *The Fourier transform and its applications*. Vol. 31999. New York: McGraw-hill, 1986.
3. Koenig, David M. *Spectral analysis of musical sounds with emphasis on the piano*. OUP Oxford, 2014.
4. Tanenbaum, Andrew. *Modern operating systems*. Pearson Education, Inc.,, 2009.
5. Wilson, Scott, David Cottle, and Nick Collins. *The SuperCollider Book*. The MIT Press, 2011.
6. https://arduino.cc/
7. https://bela.io/about
8. https://ccrma.stanford.edu/~jos/sasp/Additive_Synthesis_Early_Sinusoidal.html
9. https://cymatics.fm/blogs/production/fm-synthesis
10. https://electronicsforu.com/technology-trends/learn-electronics/7805-ic-voltage-regulator
11. https://hackaday.io/project/161678-usb-midi-breath-controller
12. https://learn.bela.io/the-ide/
13. https://learn.bela.io/using-bela/about-bela/bela-hardware/
14. https://musicainformatica.org/topics/groove.php
15. https://www.patmissin.com/history/melodica.html
16. https://raspberrypi.com/
17. CPSC 432 (Computer Music: Sound Synthesis)
18. https://www.amazon.com/EastRock-Melodica-Instrument-Mouthpiece-Beginners/dp/B0894HC2DH/ref=sr_1_5?crid=285HL1PX05LEV&keywords=eastrock+melodica&qid=1651111333&sprefix=eastrock+melodica%2Caps%2C182&sr=8-5