Final Report: Future Drum Machine

Rohan Lingala
rlingala@tamu.edu

Joshua Clapp
jay123q@tamu.edu

Joshua Chong
jchong99@tamu.edu

1. EXECUTIVE SUMMARY

In summation, the achieved goal was to create a hardware audio parser that could filter and adjust the sample rate of waveforms through a UI, and play back segments of the audio file through the use of buttons. We decided on this project because it had a uniqueness and complexity factor that we had never seen before: making an instrument from scratch, and with a Raspberry Pi as the brain behind it all.

Creating an instrument poses an interesting problem, this design needs to be efficient and fast, as we cannot have the user experiencing button delay or response issues, it must be tactile and quick. Not only that, but it must also be presentable and built with a strong sense of aesthetic and appearance.

What we were able to accomplish was a machine that is able to successfully playback waveforms via buttons and speakers, as well as building a UI that can modify the waveforms and a 3D model that meets our criteria for aesthetics, given the circumstances of time and production issues.

2. INTRODUCTION

Since the 1980's, hardware samplers have been a staple of the Electronic, Dance and Lofi genres of music. A huge part of the sound in these genres are the drums that provide a certain feel and texture to the music. These drum sounds are usually taken or "sampled" from other songs. A Lofi record will maybe use a kick drum or snare drum from other songs and re-used in their song. Maybe even a loop of sounds that match the tempo of the original record, are used in the new record that a music producer would make.

The understanding behind the design of these drum machines lead us to our question: is it possible for us to create a hardware based prototype that mimics the functionality of hardware samplers of old, however with the difference of having the insight of modern day technology, and a graphical user interface, as most older drum machines did not have one. Furthermore, could we add more functionality to our model, as many older samplers did not have time stretching and filter capabilities built in, as they would have needed to be routed through bulkier, and more expensive pieces of hardware back in the 80's and 90's.

We seek to address these issues in our Future Drum Machine, a lightweight solution and improvement to the samplers of a bygone era.

3. CHALLENGE SUMMARY

Our key challenges were beautifying the hardware, being able to efficiently read the buttons, and the server side backend.

Our main challenge with beautifying the hardware was that the 3D model was very tricky to work with because we had very little wiggle room in terms of box space, as well as making things look organized internally. We have achieved our goal of beautifying the hardware as much as we can, but we had to get very creative and had to have all hands on deck.

With our server side backend we had many tribulations regarding the setup and configuration of how it would interact with the frontend UI. We initially attempted a rest api method but realized we required a channel stream of data that provided our application with a more robust line of communication. Through this method, we are able to convey real time data of inputs and functionality of the hardware, multiplex buttons, and various pins. The Websocket method is not without obstacles, however. We hit various roadblocks in many aspects such as connectivity, resource pathing, event handlings, and callbacks along with having to both understand and manage how the single thread is handled. For a thread-efficient approach, the implementation of a server to host the web application, socketted communication, and running dynamic polling hardware required many trials, debugging, and software tracebacks.

While the software side was plowing through several issues, we also faced challenges of the initial hardware soldering implementation and hardware wiring in a sensible and efficient manner. In industry network communications for fiber optic cables, it's extremely difficult to manually test each cable to note whether or not it's operating with or without irregularities. In the same way, we faced issues upon hardware debugging when an expected output binary generated was either providing false positives or false negatives. With over 48 functioning cables of varying types and sizes, hardware health checkups would occur before, during, and after each run to ensure it is secure and that we are able to pinpoint exactly where our difficulties lie.

Furthermore, the software area was determining the equivalent sizes at which each virtual component must be such that it can fit upon the 480x800 along with software deprecations at every which way that would pivot our user visual interface into an alternative approach.

Finally, our last key challenge was creating a solution to the issue of how do we get 16 buttons to read into a Raspberry Pi efficiently. We had many different iterations of solutions before we settled on our final solution, which utilized a 5 GPIO pin setup to get unique button information to then map to code, and the server side backend.

More details about our challenges are listed in the next section.

4. PROBLEM SOLVING & CONTRIBUTION SUMMARY

Throughout the entire semester, we have faced many different difficulties and trials. The first issue we encountered was being able to parse the audio files, we originally made our own mp3 file parsers. We eventually realized that using a compressed audio format was not good for us and it came with a lot of overhead and debugging issues that were tricky to deal with. We then recreated the file parser with wav file formatting instead, and we were able to successfully parse and split the files, but the audio was very distorted because of missing information in the byte header. We eventually decided to utilize some libraries to help us parse the audio easier. We also had some minor issues with the frontend, we eventually settled on a frontend/backend web page so that we have better control of our UI design.

Our next big issue was dealing with hardware, specifically the CAD model and circuit design. Regarding the CAD, it was mostly just trial and error with the software and getting comfortable with using CAD, as well as meeting the design mock-up specifications. In hindsight, one thing with the CAD that we should've done was make the dimensions a little looser as later we ran into many issues with the construction of the circuitry into the hardware. Now for the circuitry. With the circuitry we had many different iterations of solving the issue of how do we get 16 buttons to respond quickly and efficiently with the Python code and the Raspberry Pi 4. Our first idea was to poll on all 16 pins but that would make our program very slow and the Raspberry Pi could not physically handle that much information that quickly, because it is only 4 cores. We also tried polling on 6 pins with diodes however that resulted in parasitic flow. Once we tried those, we eventually decided to use a model that involved LEDs instead of diodes to built a 16:5 MUX in hardware to minimize the use of GPIO pins to only 5, and maximize the use of the buttons without killing efficiency as well as making sure the Raspberry Pi did not work too hard. We had one pin represent the button on/off state, and the other 4 act as a 4 bit binary number, so we could represent up to 16 unique outputs for our 15 buttons.

However, even though we had our design, we still had many issues with implementation. We had to create unique routing and soldering to get the job done practically. We utilized male to female wires early on in the process and only soldered minimally as possible but we soon realized that a lot of our reliability issues came down to the male the female wires being loose, which made us go back and solder loose connections.

Finally, wrapping everything together. Once we finalized our CAD and circuit, we started putting everything together. Our main issue was making everything look neat and pretty. Initially, we did not have our design be very flush, we had wires sticking out of the sides and had to tape everything up, however we were able to find creative ways to beautify the drum machine, one of which being using mount tape and velcro to fasten the lid onto the machine, as well as using sleek black tape to seal off outside holes.

Surprise, there was a massive issue. The pin mapping that Joshua Clapp did on the buttons in pure python. Read differently using Joshua Chong's web socket implementation. This really was a massive stall on progress as retesting and remapping all the pins was a very demoralizing and time demanding process.

Once we had our hardware fully complete (beautified, and tested) we then had to finalize our software backend. Backend development refers to utilizing a websocket implementation to have real time inputs from the hardware be visualized and demonstrated on the front end ui design. The communication was chosen over an api depicted method as the connection should always be listening for any written messages denoted by the server. Various api and websocket implementations were tested with tornado's websocket proving to be more robust and better fit for the project's scope. The communication contains, (when a button is pressed and confirmed as a valid input) a json formatted list containing the state at which the current hardware is at along with the iterators for each list therein. Initial tornado websocket and app initialization and running contained several issues that refer to how the app is able to view the front end html/css/js files that was resolved through pathing.

5. SYSTEM DESIGN

Through an inspired binary encoded method, the design of the project is as follows: we have a button mapped system where each button corresponds to a unique 4 bit binary number and all 16 buttons are controlled by 5 GPIO pins. With the GPIO pin information being sent to Python, several options are allowed in terms of execution. We have 6 buttons for UI and hardware functions, and 9 buttons for audio playback. For the UI buttons we have a move left, move right, select, back button, and confirm settings buttons, and for the hardware we have one reset preferences button. The 9 playback buttons consist of one button which will play back the entire song, and 8 buttons that will play back the chunks of the song. The UI communicates with the buttons through an active server that hosts the web application, all its relevant resources, and retains all functionality of the hardware running within a tightly packed server.py. Hardware and Software : The server.py begins that hosts the web application that has an active websocket connection on the 8888th port. The server.py also has an instance of the hardware class that initializes itself fully and is continually checking for a change on GPIO 5 (The pin that indicates any button has been pressed) via buttonReady function (That also incorporates error handling on unknown pin inputs. Once GPIO 5 is hit, for a user-defined amount of time, the other GPIO pins are observed to determine the binary declaration on 4 bits to which a button's function has been mapped. The string generated serves as the key within a button dictionary that references the function as a dictionary value. Once the button function activates, the websocket message is formatted into a readable json that is broadcasted to the front end. The front end interprets the data and parses it for the appropriate page declaration in real time to select desired settings upon the audio file.

The activation state and activate related button functions, the .wav file undergoes several operations we've implemented such as high pass, low pass, speedup, slowdown, fractional, etc. The left side of the casing provides the 9 buttons that then allow playback of the entire song, and 8 segments of that entire song all retaining user chosen modifications.

6. OUTCOMES

Our outcome produced by the prototype is a machine that is able to modify wav files, play them back, and utilize a UI that is controlled by a full stack backend.

Some experimental oddities and results of our model are:

- Occasional undefined behavior likely due to the insulators grounding some of the wiring. This sounds insane but a duct tape on a GPIO wire was causing very strange behavior where it was reading a zero when closed and a one when the lid was off. Removed the tape, the issues ceased.
- There are some restraints in the handling of the song files themselves. If the file is too large the compile time is unruly, or if it is too short the parsing will fail as it is too short.

7. APPLICATION ENVIRONMENT

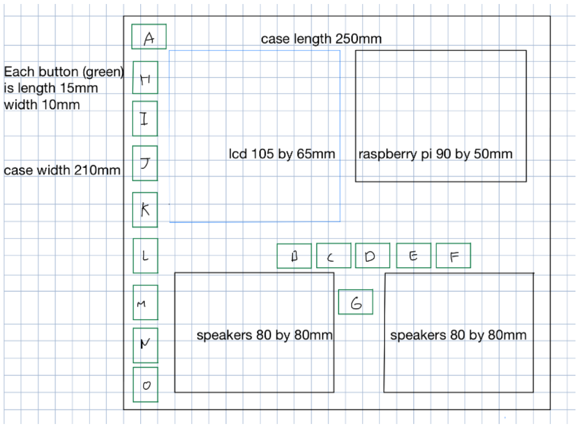
Our goal is to manipulate wav files using scipy.io.wavfile [1]. The goal is to partition the wav file into 8 wav files that are mapped onto buttons that we can use to playback the newly created wav files. We are going to attempt to time stretch and time shrink the sound files. We are also only using preset sound files. If the speedup works, we then go to highpass and lowpass. Finally, we then want to take our audio output and send it to a stereo out, that way we can get the audio output recorded onto a digital audio workstation. The goal is to just focus on making a functioning drum machine with these parameters.

8. DESIGN AND RATIONALE

Our solution is to use a signal processing algorithm in order to stretch or shrink a wav file from a song to match the target tempo. Then we will divide the song into 4 or 8 divisions where each division will get its own button to play back the chunk of the drum loop or song assigned to it.

Example 1 : The design here is that if the song Happy Birthday was playing, buttons 1-4 would become "Happy" , "Birthday", "to", "Jyh", and now pressing button 4 over and over would repeat "Jyh", but if the song was longer, we could do buttons 1 - 8.

The dimensions of the project can be seen below.



A. play whole song
B. back state
C. select left option
D. next state
E. select right option
F. confirm presets and parse
G. clear presets and parse
H. play 1st part of song on press with settings
I. play 2nd part of song on press with settings
J. play 3rd part of song on press with settings
K. play 4th part of song on press with settings
L. play 5th part of song on press with settings
M. play 6th part of song on press with settings
N. play 7th part of song on press with settings
O. play 8th part of song on press with settings

Current Idea rough estimate

Utilizing software editors and 3d printing, a design and physical case will be created to house the hardware. Playsound library will assist in pi to speaker output, buttons and external hardware are to be soldered, and alternative plans are in place in case of hardware failure in assembly (Example is 3d printing issues, woodworking issues, dimension issues, etc.)

9. SYSTEM AND CIRCUIT DESIGN

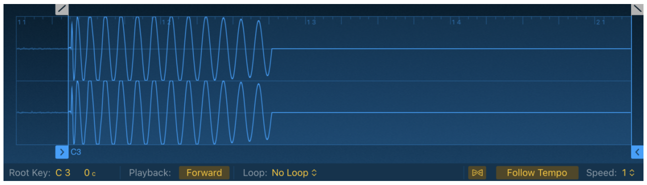
Overall Design :

Initially, the design will have two buttons on top that will pick which drum loop to start parsing. A press of any of these buttons marked [A, B] will change the drum loop type before runtime. A future design iteration will have these buttons go up and down a bank of sounds the user can pick from. This will be done with polling and each wav file will trigger itself on playback. This will also include a UI from an LCD screen in order for the user to see their selections.

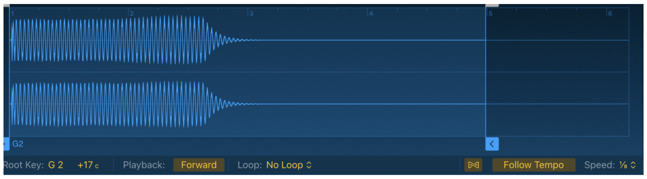
Once a letter button is picked, the wav file is partitioned into 8 "slices" that are able to be played back in any desired order. A future design iteration will possibly allow for time-stretching and time-shrinking the files to get different playback speeds. If the time-stretching is successful, we will then move onto creating low-pass and high-pass filters. The filters and time-stretching is all done before playback. The user may also see on the LCD screen a UI for picking a low pass or high pass filter.

Software will parse the wav files and then play them back into the speakers into the Raspberry Pi.

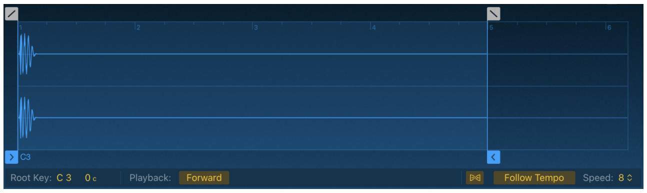
For the exact software, we will use Python to run on the Raspberry Pi, and use numpy, scipy.io.wavfile, and playsound, packages. Numpy is used for basic mathematics, scipy.io.wavfile will actually read and create the eight wav files to play, and playsound is used for audio playback. This will be what our base is to recreate and playback the sound files. After the sound files are properly parsed a speedup feature that increases the frequency of the playback. This speedup feature will be done in a while loop which will cause more errors as there is variation.


Root Key: C 3 0 c Playback: Forward Loop: No Loop Speed: 1

This is a 54 hz kick drum sin wave that we will time slow and stretch it to increase the playback speed and play it back on button press.


Root Key: G 2 +17 c Playback: Forward Loop: No Loop Speed: ⅛

This is a slow down of the same kick drum. It has scaled the sample rate by ⅛ of the time.


Root Key: C 3 0 c Playback: Forward Speed: 8

This is a speed up of the same kick drum. It has scaled the sample rate by 8 of the time.

Non-Computing Elements :

Housing of elements -> 3d Printed design is best crafted to allow stable structure of project. If 3d prints fail, a wooden design will be used. Mechanical elements will likely be used as locking mechanisms to secure the design if any. Structural elements pertain to the 3d housing and will likely implement a loose, holed structure in order to permit a cooling flow of air in and out of the system.

Computing and Electronics elements :

Raspberry Pi 4, LCD, Speakers, Wires, Buttons, Resistor, and other miscellaneous components are within current plans for the listed components but may change in the finalization of the project.

Software :

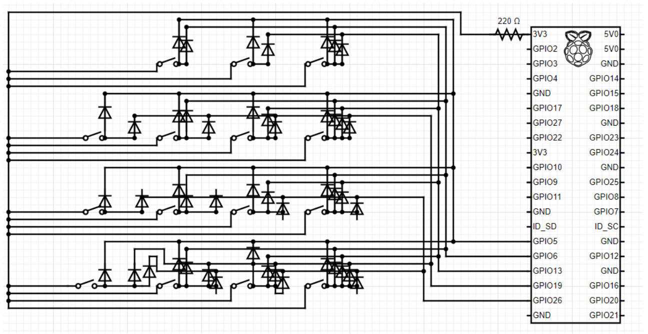
In creation of visual assets , circuit-diagram editor [3] and adobe substance stager 3d are in use to demonstrate and convey designs.

Python coding language and structure are to be used.

Relevant libraries are to be listed in requirements.txt (Example : GPIO, MatPlotLib, Numpy, Scipy, Playsound…). Note that current libraries are not finalized and library usage may be added or excluded in finalization + optimization of the project in case of latency, incompatibility, and other rising issues.

Blending hardware and software :

Button and circuit implementation event listening via interrupt handling takes resources and reduces accuracy of user input detection and searching through all 4 (Excluding detection GPIO pin) pins would waste resources. To prevent this, a method of binary buttons is implemented with the last button being the only button checked every cycle. Once any button is active, GPIO5 will be active and the program will then proceed to the next state of analyzing the other 4 pins briefly to determine the binary equivalent that corresponds to a user defined function implementation. delete and replace with One of the main issues faced by the team was how to poll on the pins to receive system output. If every pin was used the Raspberry Pi would need to poll on sixteen pins at the same time. This is unfeasible. Ordering a MUX gate to perform the same logic would have taken too long to arrive so the solution was to route each button to a combination of pins. To check for any button input a check on pin 5 is done and assigned arbitrarily. Once pin 5 is checked and receives a positive then listeners can be opened up on other pins. This works very well, but makes soldering and cable management a nightmare that Joshua Clapp routinely wakes up in cold sweats from. To make the routing for a custom MUX, we used solder routers, where we take male pins and add solder connecting them all and this allows us to connect male pins and route to the GPIO board. This means if yellow is pressed it routes to a three male pins and provides voltage to all, this then sends to the GPIO pins and provides voltage read on the Pi. Backflow is prevented using LED diodes, and is vital to the success of the project. Several issues stem from loss solders and mainly it was just testing and finding the poor connections, violently shaking wiring to test loose outputs and trial and error. Debugging this was very painful as with so many wires it was any number of connections to the boards, connections to the solder routers, and caused the deaths of twenty four leds as Josh Clapp burnt them out by accident looking in the wrong area.



Circuit implementation of 15 buttons.

Active States interfacing to Raspberry Pi			GPIO5	GPIO6	GPIO13	GPIO19	GPIO26
0	Default - Nothing	na	🟥	🟥	🟥	🟥	🟥
10001	confirm presets and parse	F	🟩	🟥	🟥	🟥	🟩
10010	play parsed 2	J	🟩	🟥	🟥	🟩	🟥
10011	play parsed 4	L	🟩	🟥	🟥	🟩	🟩
10100	play parsed 3	K	🟩	🟥	🟩	🟥	🟥
10101	select left option	C	🟩	🟥	🟩	🟥	🟩
10110	next state	D	🟩	🟥	🟩	🟩	🟥
10111	play parsed 6	N	🟩	🟥	🟩	🟩	🟩
11000	play all parsed parts	A	🟩	🟩	🟥	🟥	🟥
11001	select right option	E	🟩	🟩	🟥	🟥	🟩
11010	play parsed 0	H	🟩	🟩	🟥	🟩	🟥
11011	play parsed 5	M	🟩	🟩	🟥	🟩	🟩
11100	reset all presets	G	🟩	🟩	🟩	🟥	🟥
11101	select left option	B	🟩	🟩	🟩	🟥	🟩
11110	play parsed 1	I	🟩	🟩	🟩	🟩	🟥
11111	play parsed 7	O	🟩	🟩	🟩	🟩	🟩

1. Team Efforts Breakdown

Rohan L.	> Song splicer, he finished debugging and got it working > Wav creation > Main idea guy > Case design > High/low filters
Josh Cl.	> Overhead management, boilerplate > Hardware GPIO triggering songs > Case Cad Modeling > Speedup slow down > 2 speed up buttons
Josh Ch.	> Overhead management, boilerplate > Visuals and circuit design > Case design > High/low filters > Documentation Design > User Interface visual > Software-relevancy

2. WORK PLAN

Phase 1	October 14th	> Software splicing of audio .wav file into 8 segments. >> Determine quality of segments > Software recombination of 8 segments and replay.
Phase 2	October 28th	> Software functions to correlate to hardware buttons >> Time-stretching and time-shrinking of audio file(s). >> Low Pass and High Pass filters > Hardware breadboard implementation test and development >> Modeling the 3d print
Phase 3	November 11th	> Hardware stalled week, raspberry pi 4 passed > Rohan and Joshua Clapp soldered > Software overhead UI implementation >> Audio choice movement >> Time shift selection >> Low/None/High pass selection
Phase 4	November 21st	> Built final product > Audio changes at runtime > Hardware button assignment > Stereo jack output + Audio interface and recording on DAW.
Phase 5	November 28th	> Beautifying final product > Final Recording > Final Report > Final Presentation

10. REFERENCES

[1]Scipy, “Scipy/scipy at V1.9.2,” GitHub. [Online]. Available: https://github.com/scipy/scipy/tree/v1.9.2.

[2]“Playsound,” PyPI. [Online]. Available: https://pypi.org/project/playsound/.

[3]“Circuit diagram,” *Circuit Diagram Web Editor*. [Online]. Available: https://www.circuit-diagram.org/editor/.