APPLICATION PLAN

Group 13

1. Information for our group

Z5207250 Jack Ransome
Z5348946 Siyu Qiu
Z5319020 Yuchen Du
Z5158505 Zhengye Ma

2. Project overview (max 200 words)

This project uses the combination of a microprocessor and an FPGA, along with additional hardware such as a microphone and the wires required to connect them, to build a system capable of recording audio and saving it to a .wav file. On the hardware side of things, the microphone is attached to a chip that outputs the recorded sound data using the i2s protocol. This chip is connected to input ports of the FPGA, where the configuration of the hardware interfaces both with the microphone chip through I2S, and with the microprocessor's high-performance ports through the AXI protocol, allowing the transfer of data from one to the other. Inside the microprocessor, the incoming data is routed from the high-performance ports to our own program, where we plan to accumulate this data and then format it in the appropriate way, before saving it in the required .wav format. After these basic requirements are met, we will be able to extend the system to interface with additional hardware, for both input and output and process the data in new ways. This will be required to execute our vision for the application extension which is talked about in part 5.

3. Current vision for Application extension (max 300 words)

Our plan to extend the application beyond the basic requirements is to build it into an 8-step sequencer which will be able to sequence user recorded sounds. A step sequencer allows a user to arrange sounds on a timeline which is played back in a loop. The timeline is quantised into discrete steps, and on each step the user is able to specify which sounds will begin playing at the start of that step. This allows the user to compose short musical loops that will be played back through a speaker. The sounds that the user can choose to enable on each step are from a collection of sounds stored internally, which will be populated by sounds that the user records into the system using the microphone. There are multiple different ways to format the user interface for a step sequencer, in this case we will have one button for start/stop, one button for each step, one button for each sound, one LED per step, and one LED per sound. The sound LEDs will indicate which sound is currently selected, and the step LEDs, arranged in a horizontal row of 8, will indicate whether or not that sound is set to play, on each of the steps. When one of the sounds is

selected, the step buttons will be used to toggle that sound for that particular step. The start/stop button will begin or stop the looping playback of the sequence. This application is extensible, as it leaves room to incorporate sound effects to apply selectively to sounds, to output to multiple speakers, synthesise sounds from scratch, and add additional sequencing features, such as time stretching.

4. A 1-page high-level "system diagram"
all completed aspects of your project + the future planned aspects

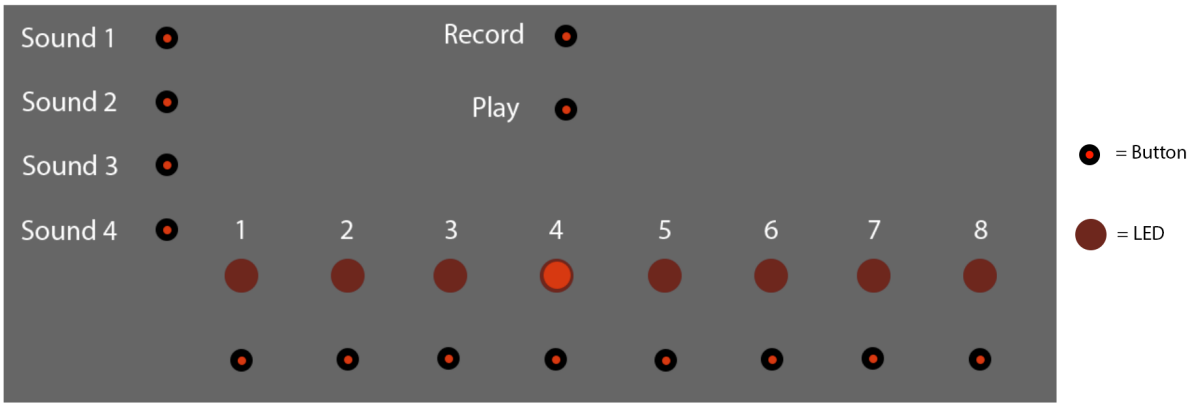

Figure 1: mockup of the user interface

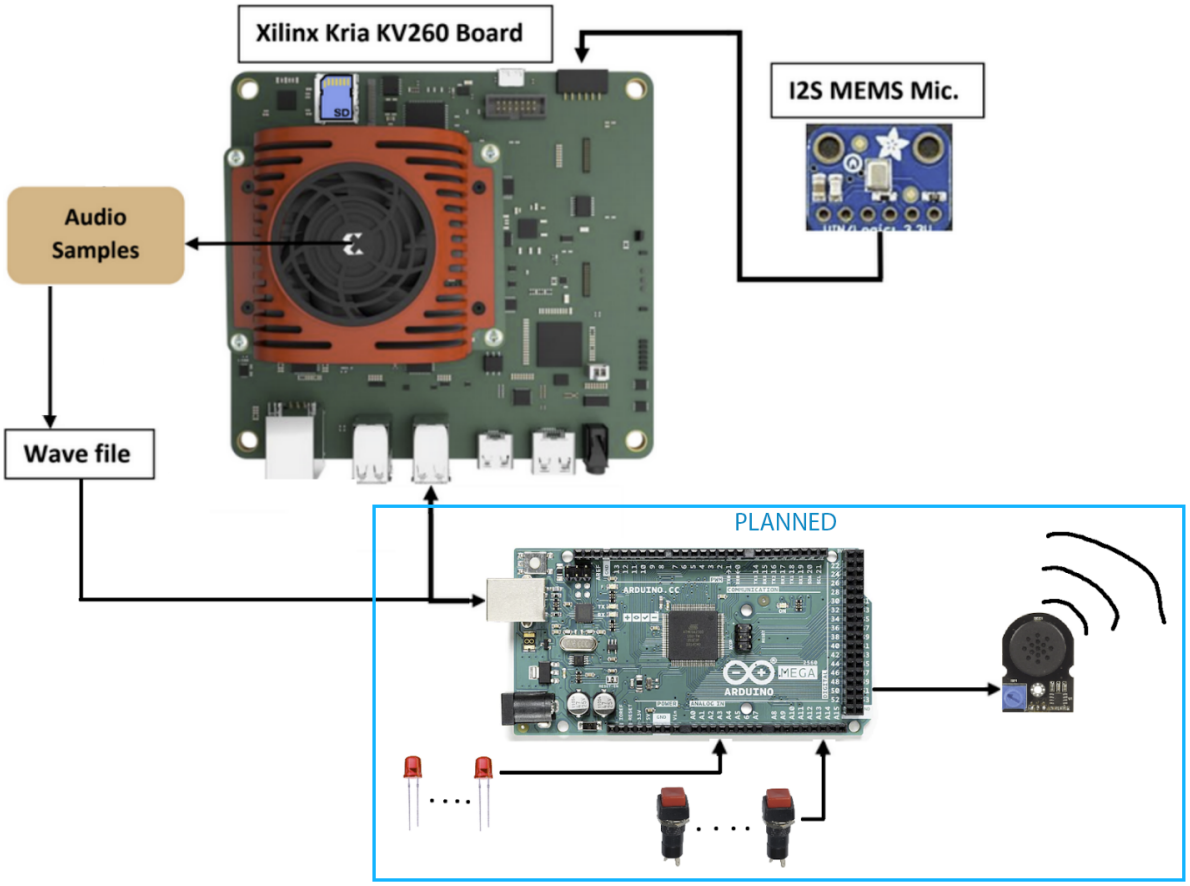

Figure 2: High level system diagram

5. A 1-page "architecture diagram"

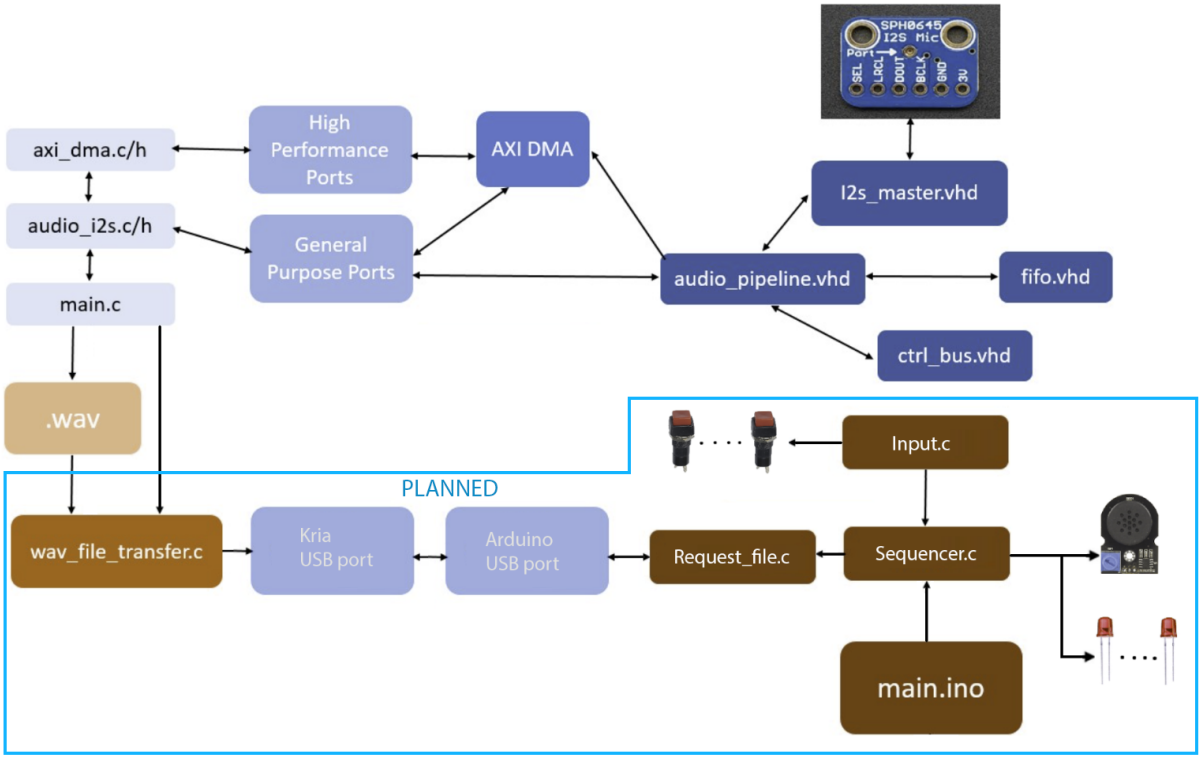

Figure 3: Architecture diagram

6. Explanation of project plan diagrams - (max 1 page, 12pt font)

System diagram:

The board waits for requests from the arduino to record new sounds. When it receives a request it records sound through the i2s microphone, saves it to a .wav file, and then sends this file over usb to the arduino, where it will be stored on the sd card. The arduino takes button inputs, has led outputs, a speaker chip output, and a usb connection to the board. The arduino has an internal state which stores the current configuration of the sequence (arrangement of sounds on a timeline) as well as 4 sounds on its SD card. Depending on the user input from the buttons, the code on the arduino will either request a new sound from the board, change the current configuration of the sequence, or play sounds through the speaker chip based on the current sequence.

Figure 1 is a mockup of the user interface, showing the user facing buttons and LEDs. The buttons on the left hand side select a sound. The buttons at the bottom place the selected sound at that point in the timeline. The 2 buttons in the middle play the sequence in a loop, or record to the selected sound. The LED's indicate which positions in the timeline the selected sound will play at, and which position in the sequence is currently being played, if the play button has been pressed.

Architecture diagram:

Input.c: sets variables representing the state of each button, based on input pins.

Sequencer.c: maintains a set of variables that represent the current state of the sequence (4*8 array, since there are 4 sounds and 8 time steps). Reads variables

handled by Input.c, and edits the sequence array accordingly, or plays the sequence, or calls a function in Request_file.c to request the recording of a new sound.
Request_file.c: sends requests via USB, waits for an appropriate response then saves data received via usb into a new .wav file, until an end signal is received.
Main.ino: loops the code in sequencer.c, starting our arduino program.
Wav_file_transfer.c: waits for a request over USB, sends a signal back to confirm, then sends wav file data byte by byte, followed by an end signal when finished.
Main.c: manages the PL/PS system's operations, initialising hardware and managing data flow. Receives audio data from code in Audio_i2s.c, processes it, and saves it as a WAV file. Also calls wav_file_transfer.c to manage interaction with the arduino.
Audio_i2s.c: Provides an interface for initialising and interacting with an I2S audio system. Uses axi_dma.c to initialise and manage DMA transfers.
axi_dma.c: manage DMA operations between FPGA and processor memory, handling data transfer initialization, status checking, and cleanup routines.
AXI DMA: Links the PS to the PL for audio data movement using the AXI protocol
I2s_master.vhd: Captures audio data from a microphone via I2S interface and writes to a FIFO buffer.
Audio_pipeline.vhd: Manages data flow between I2S master, FIFO buffer, and the PS, including control signal handling via AXI4-Lite interface.
fifo.vhd: Implements a FIFO buffer to temporarily hold audio data during processing.
Ctrl_bus.vhd: Controls signal exchange within the PL for component coordination.

7. Plan to complete project with tasks and assignments (~250 words per task)

1. Hardware creation - Jack

Cut holes in the breadboard:

We have purchased a breadboard that will act as the top outer panel of our product, which will have all the buttons and LEDs of the user interface embedded on it. We need to cut holes in this breadboard so that the buttons we have purchased can fit into it.

solder LEDs and buttons:

Next, we need to insert the LEDs and buttons into one side of the breadboard, test to make sure they are also functional, glue them into place, and then solder them to wires on the underside of the breadboard.

Connect the speaker, UI, and SD to Arduino:

Since the Arduino will be handling the user input, deciding what to display on the LEDs, interfacing with the SD card for storage, and controlling the speaker, all of the above will need to be connected to it. Firstly, the speaker. First we will purchase an SD shield then plug it into the arduino, with the SD card inside. Then we will connect all the buttons to the board, each button has 2 ports, and 1 port of each button will be connected to the ground port of the arduino, while the other port of each will be connected to a unique input port of the arduino so that it may be read in pullup resistor mode.

2. USB transfer software - Siyu and Yuchen

Write USB serial communication code for the Linux system and Arduino, and implement file transfer and recording functions:

- Aim: To establish a communication channel between the Arduino and the board, allow the transfer of files from the board to the Arduino, store audio files for later playback, and enable the Arduino to remotely control the board to record and send audio files. This will all be done inside request_file.c so that sequencer.c can use this code to request new files.
- Outcome: A successful data exchange between the Arduino and the board over USB, files can be successfully sent from the board to the Arduino, a wav file can be successfully saved to and retrieved from the Arduinos SD card, and the Arduino can successfully send a request to the board to record a new wav file and receive it back. The completion of this task can be verified by sending test data back and forth between the two systems, sending a test file from the board to the Arduino and checking its integrity, saving a test wav file to the Arduino and successfully retrieving and playing it back, and initiating a recording request from the Arduino, checking that the board receives and executes it, and ensuring the newly recorded wav file is sent back to the Arduino.

3. Write sequencer code for Arduino - Jack and Zhengye

The sequencer code is the code written in input.c and sequencer.c, both of which run on the arduino. The first step here is to write the input.c code to read values off the pins connected to our buttons, these values are then written to variables (one for each button) that sequencer.c will be able to view and make decisions based on. This is likely to be a tricky task, as our group members have experienced issues with this type of code in the past, as cheap buttons may have a flickering on/off values especially on press and release of a button.

The next step is writing sequencer.c, which will carry out 3 main tasks: playing the loop, deciding when to send a new request, and modifying the current sequence data, all based on input captured by input.c. This file will store a 4*8 array of boolean values, called sequence_matrix, this represents the current composition of the sequence, where 4 rows of 8, one row for each sound, represents at which points on the timeline each sound is set to play at. It will also store a value that represents which sound is currently selected, called selected_sound, and will change this value based on which of the 4 sound buttons was pressed last. If the record button is pressed, this code uses request_file.c to fetch a new sound recording from the board, and associates this file with selected_sound. When the play button is pressed, a timing system will be started, which works its way through the step sequence 1 step at a time, playing each sound at that step according to the sequence_matrix.

8. Justification of plan

The global market for musical instruments, which was valued at $14.20 billion USD in 2022, is growing quickly with a projected CAGR(compound annual growth rate) of 7.4%

between 2023 and 2030. Within this market, the electronic musical instruments sector is estimated to grow at a CAGR of 3.13% between 2022 and 2027, showing there is a growing market for digital music-making devices. Large companies like Korg, Moog, and Yamaha have established a demand for electronic instruments through their sales and market presence, proving the demand is already significant.

Our proposed hardware step sequencer is designed to simplify the process of recording, arranging, and looping sounds on an 8-step timeline in a portable unit. This design caters to both amateur and professional musicians who desire a straightforward music creation tool.

In comparison, the Cirklon sequencer, priced between $3,299 to $4,500, targets a high-end market segment with its 16-track capability and extensive connectivity options. However, this complexity and price point deters a large portion of the potential user base, especially those seeking simpler and more affordable sequencing solutions. Our hardware step sequencer, with its intuitive interface and simplicity, which will lead to lower prices, aims to fill this market gap. Its design still contains the essential sequencing functions, making the process of music creation accessible to a much wider audience.

Overall, our hardware step sequencer design fills a gap in a growing market. Its simplistic design and affordability, relative to high-end sequencers like the Cirklon, position it as a suitable product for a large user demographic. The existing market presence and success of electronic instruments from reputable companies further proves that the market opportunity for this hardware step sequencer exists.

9. Amendment

What we achieved:

Our current implementation successfully allows the user to record 4 different sounds, arrange them on a timeline, and then export to a final .wav file. As planned, the user is able to do all of this through the button + LED interface, and does not need to interact with any other parts of the system to use its full functionality. Figure 4 shows the button and LED interface, showing that we successfully executed our plan for this part of the system. Figure 7 shows 4 user recorded sounds, and an exported wav file our system has created out of the user recorded sounds and the user created composition.

What is different from the plan:

In contrast to our milestone 3 plan, the Kria board acts as the central hub of the system, rather than the arduino. This means that instead of the arduino running the logic to manipulate the composition, and requesting new sounds from the board, the board instead manages all the sequencer logic and simply listens for button events from the arduino, and sends back new LED configurations. This change occurred because we ran

into difficulty sending large files to the arduino, since it has a very small onboard memory, meaning we would need to give it an sd card and write software to do buffering so that we could transmit the larger files. The new configuration treats the arduino like a peripheral instead, and had allowed us to focus on doing the sound processing entirely on the kria board.

Another change from the original plan is that instead of doing serial communication between the arduino and the kria board over usb, we instead use an ethernet cable, as seen in figure 5, and set up a server on the arduino which the kria board connects to. After milestone 3 we did initially go down the path of writing our code for serial communication over USB, and wrote a testing program in c++ to run on a windows machine that would interact with the arduino, since uploading a new C program to the kria board every time we wanted to change something was time consuming. Unfortunately, when we decided to start testing on the kria board, we found that petalinux did not have the required drivers to handle the usb serial communication. We then decided that the easiest fix was to attach an ethernet shield to the arduino and connect it via ethernet to the board. The result is that the arduino runs a server which the kria connects to, and all communication is done over TCP, which simplified our code in comparison to our serial USB communication code.

What is missing:

The functionality that we originally planned that is currently missing from the system is playback of the composition over a speaker. This feature would allow the user to preview how their composition would sound before exporting it as a .wav file, by playing it out of a speaker connected to the arduino. The feature is now considered a stretch goal of the project, as our current implementation is deemed the minimum viable product, since we already allow the user to achieve the same task, although in a more time consuming way. This feature was cut because of time constraints.

How the rest of the plan would be implemented:

Adding the missing functionality of playback of the composition over a speaker would require sending the final wav file from the board over tcp to the arduino, where it would be buffered and saved onto an sd card, which it would then be played over a speaker from. An alternative implementation here could be instead streaming the audio data over ethernet via UDP, which the arduino would receive and play in real time.

10. Results and photos:

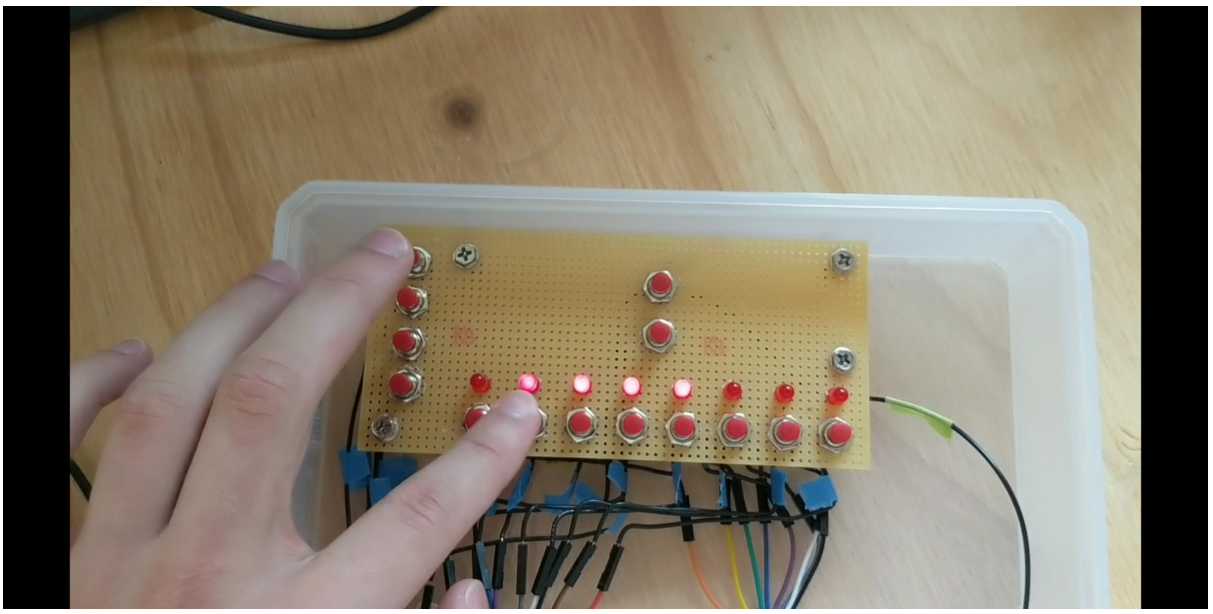

Figure 4: User interface featuring buttons and LEDs

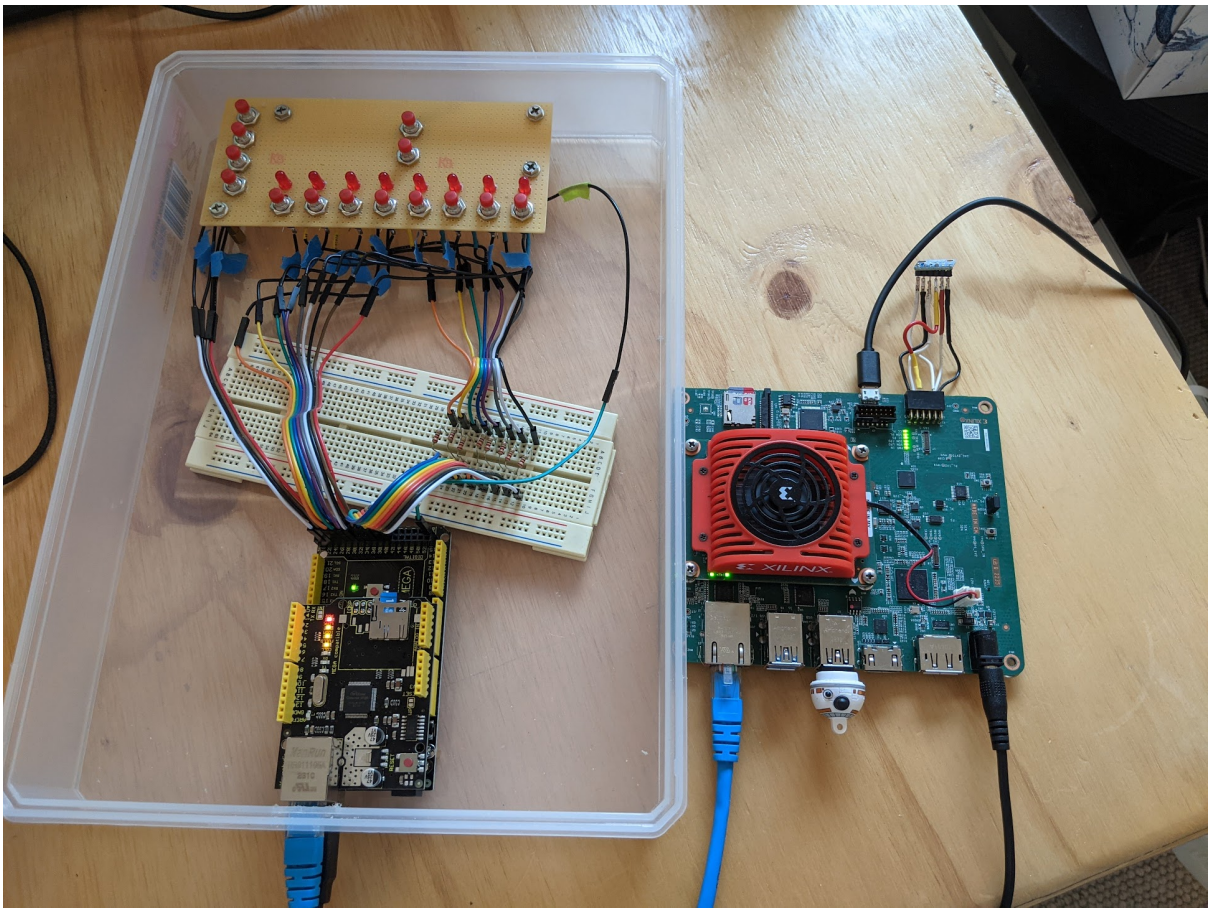

Figure 5: The entire system currently

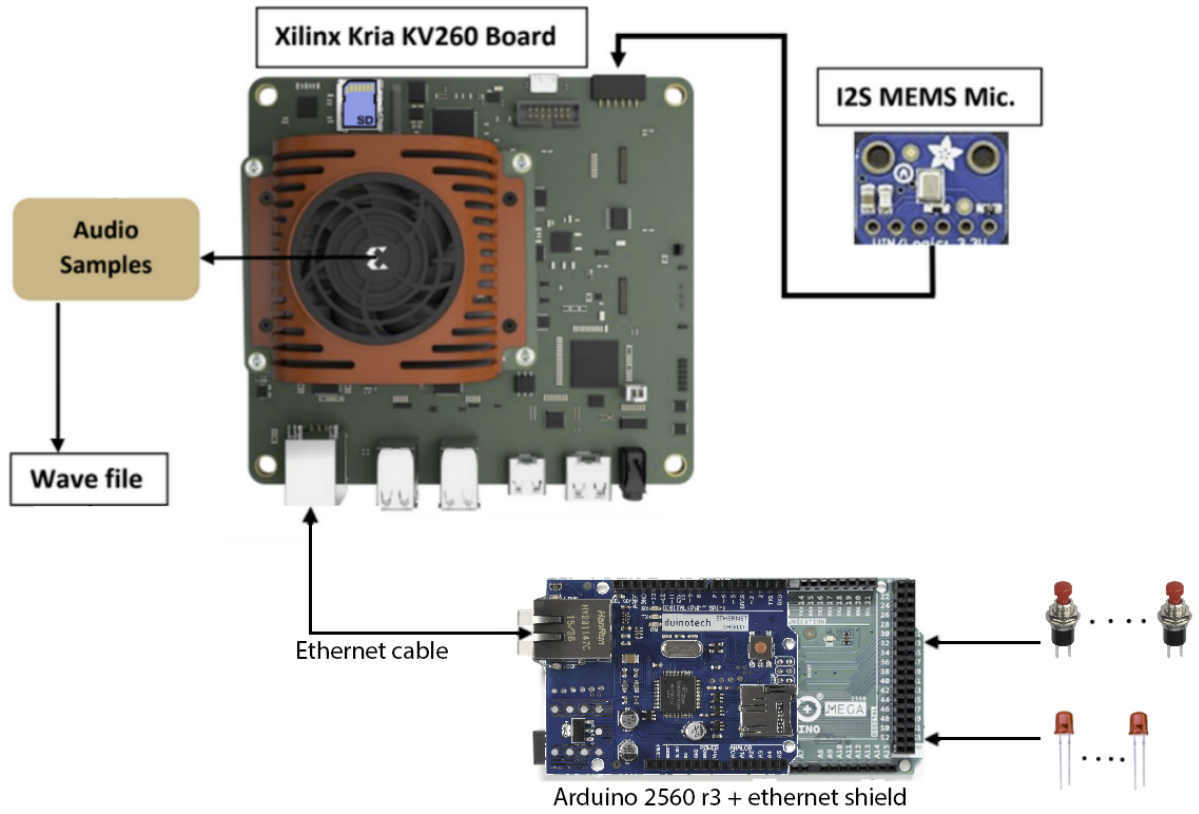

Figure 6: Updated System diagram to reflect current progress

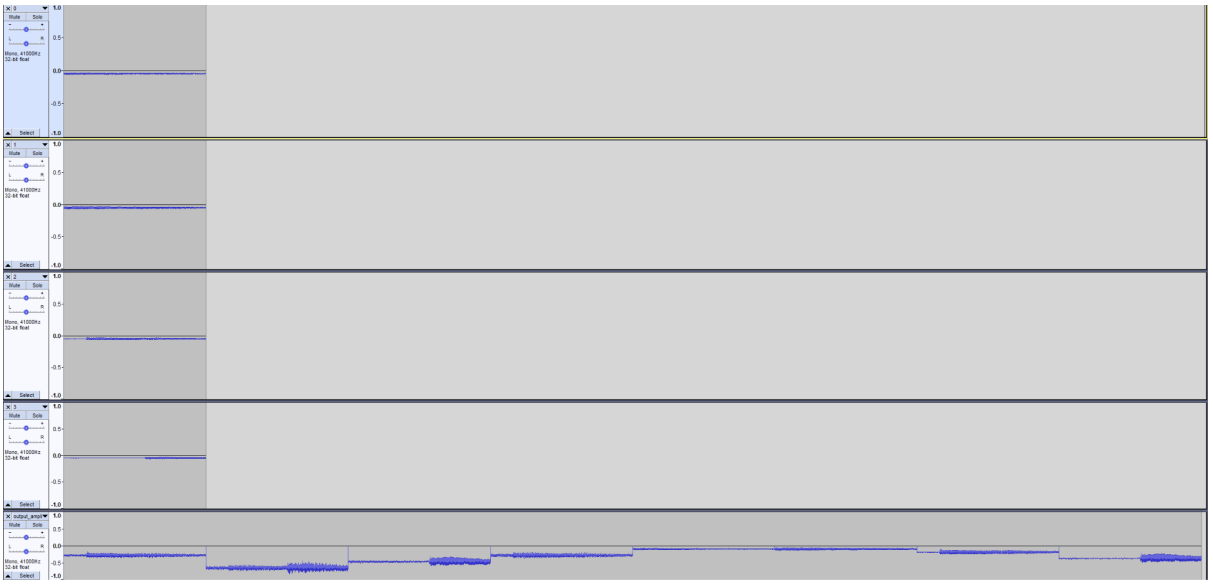

Figure 7: audacity screenshot showing 4 recorded sounds, and 1 final composition made from them