Review the assignment due date

final-project-skeleton

Team Number: 31

Team Name: 404 not found

Team Member Name	Email Address
Pranay Choudhuri	pranay24@seas.upenn.edu
Haoran Liang	liang9@seas.upenn.edu
Kefei Bao	kefeib@seas.upenn.edu

GitHub Repository URL:
https://github.com/upenn-embedded/final-project-f25-404-not-found

GitHub Pages Website URL: [for final submission]*

Final Project Proposal

1. Abstract

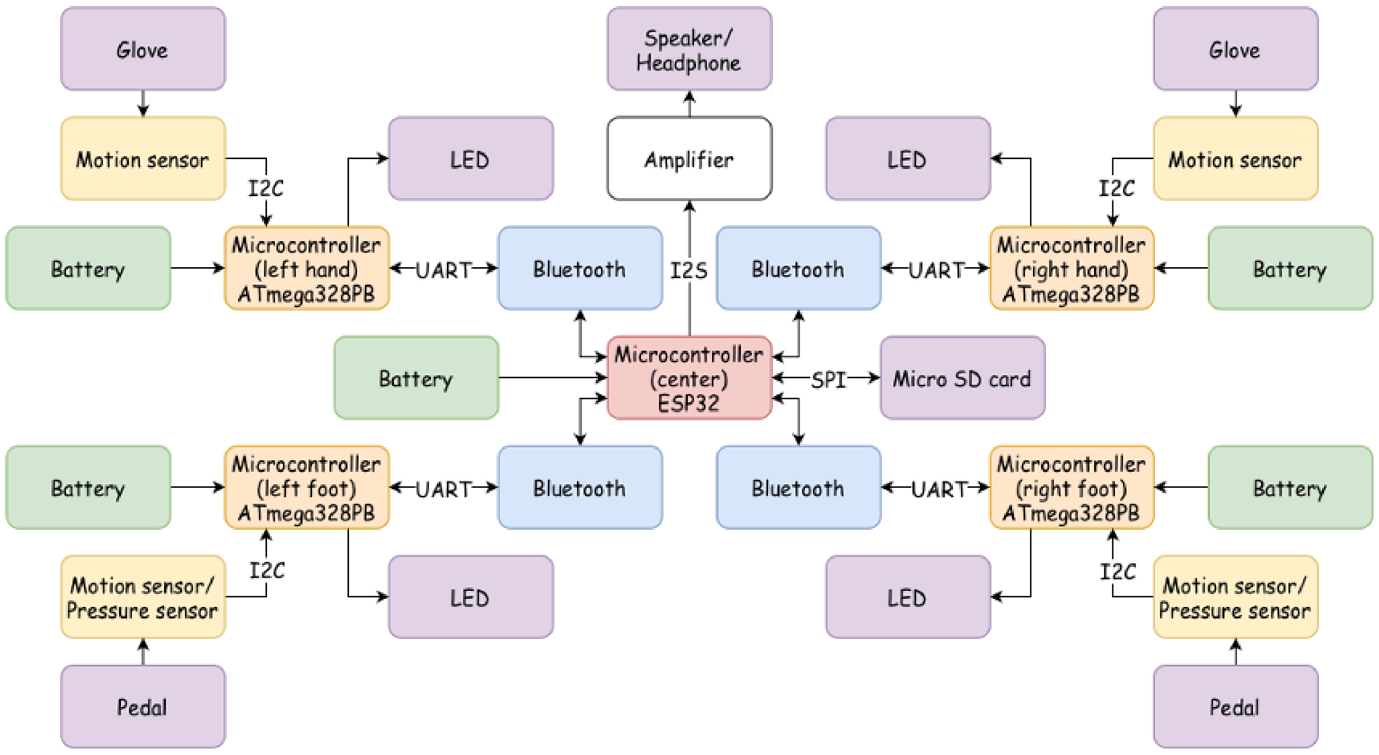
Our virtual drum kit system is built around multiple sensor nodes, each equipped with a motion sensor, microcontroller, and wireless communication module, all connected to a central hub that serves as the heart of the kit. The hub houses a microcontroller and the necessary hardware to manage communication with the nodes and generate the corresponding drum sounds. Each node is designed to be worn on the hands and feet, and when a motion sensor detects a downward strike, it sends a signal to the hub, triggering the playback of the appropriate drum sound and creating a seamless, immersive drumming experience without the need for a physical drum set.

2. Motivation

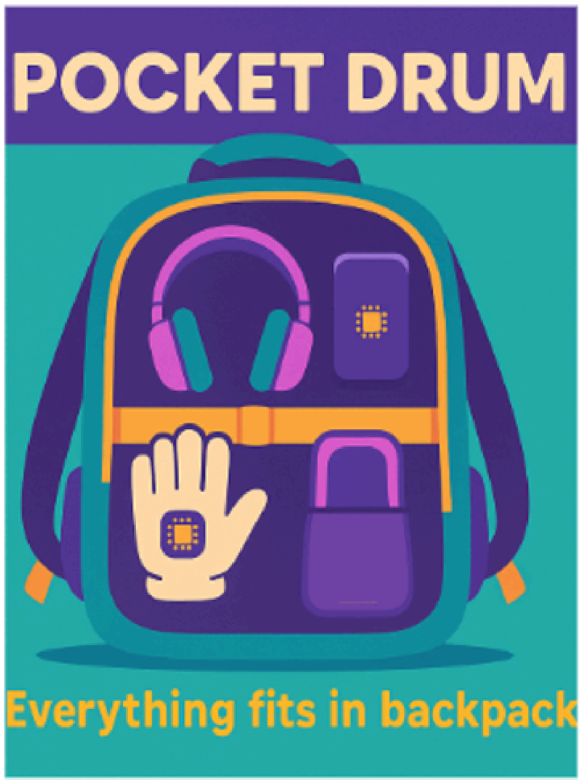
It's often the case that we vibe so hard to the rhythm of a song that we instinctively want to drum along, only to be reminded that traditional drum kits are expensive, bulky, and cumbersome to set up.

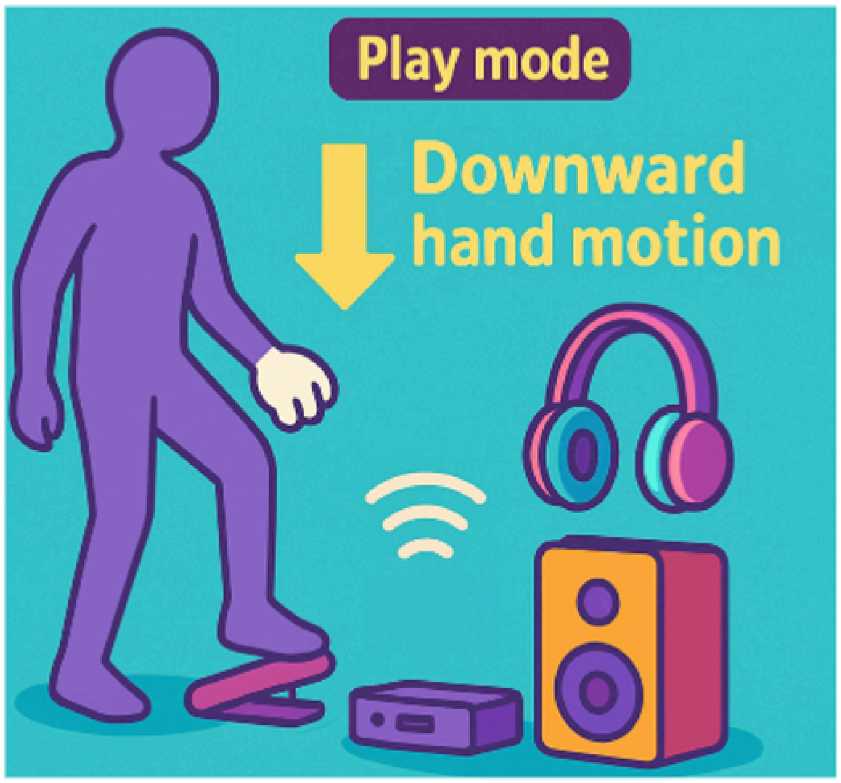
Our Virtual Drum Kit bridges this gap by offering an accessible, portable alternative.

3. System Block Diagram


Glove
Speaker/ Headphone
Glove
Motion sensor
I2C
LED
Amplifier
LED
I2C
Motion sensor
Battery
Microcontroller (left hand) ATmega328PB
UART
Bluetooth
I2S
Bluetooth
UART
Microcontroller (right hand) ATmega328PB
Battery
Battery
Microcontroller (center) ESP32
SPI
Micro SD card
Battery
Microcontroller (left foot) ATmega328PB
UART
Bluetooth
Bluetooth
UART
Microcontroller (right foot) ATmega328PB
Battery
I2C
Motion sensor/ Pressure sensor
LED
LED
I2C
Motion sensor/ Pressure sensor
Pedal
Pedal

4. Design Sketches





5. Software Requirements Specification (SRS)

Software Requirements Specification (SRS):

1. Low latency from hit to sound.
2. Reliable hit detection and low false positives on hit triggers.
3. Support stable Bluetooth communication with multiple nodes and seamless startup.
4. Multiple sound effects played depending on hand motion.

5.1 Definitions, Abbreviations

- When we talk about **latency**: time from a downward motion to sound being played.
- When we talk about a **trigger**: downward motion that has acceleration larger than 2g.
- When we talk about **connection**: Bluetooth built in with ESP32 / Bluetooth used between ESP32 and ATmega328PB.

5.2 Functionality

ID	Description
Low latency	Hit-to-sound latency less than 100 ms.
Reliable hit detection and low false positives on hit triggers	Any motion that is upward shall not trigger any sound, and only downward motion with acceleration larger than 2g is considered as valid input.
Support stable communication with multiple nodes and seamless start up	Once power for all nodes is turned on, wireless communication shall automatically engage and, if it fails, shall report an error code.
Multiple sound effect being played depends on hand motion	Like an actual drum kit, our system detects downward motion with direction and plays different instrument recordings correspondingly.

6. Hardware Requirements Specification (HRS)

Hardware Requirements Specification (HRS):

1. Reliable wired connections that can sustain sharp movements.
2. Minimal power consumption.

3. Generate audio that is loud enough.
4. IMU responsiveness.
5. Lightweight.

6.1 Definitions, Abbreviations

- IMU = SparkFun LSM6DS0 6DoF
- ATmega = ATmega328PB
- Amplifier = 1528-1696-ND
- Speaker = 668-1234-ND
- Power = 1000 mAh power bank

6.2 Functionality

ID	Description
Reliable wired connections that can sustain sharp movements	Soldered connections in sensor and center node for final demo.
Minimal power consumption	With a power bank of 1000 mAh, the system shall work for 1 hour straight.
Generate an audio that is loud enough	Output sound level larger than 60 dB.
IMU responsiveness	Any motion that is upward shall not trigger any sound, and only downward motion with acceleration larger than 2g is considered as valid input.
Light weight	Hand nodes under 200 g × 2, foot nodes under 300 g × 2, and hand nodes, foot nodes, center hub, and speaker under 500 g total, for a < 1.5 kg kit.

7. Bill of Materials (BOM)

What major components do you need and why? Try to be as specific as possible. Your Hardware & Software Requirements Specifications should inform your component choices.

In addition to this written response, copy the Final Project BOM Google Sheet and fill it out with your critical components (think: processors, sensors, actuators). Include the link to your BOM in this *section.*

https://docs.google.com/spreadsheets/d/1jcCn7PMXPBg7CsjnHoBdqLJhqJybd3HvZRBbgPiIp-4/edit?gid=2071228825#gid=2071228825

Node:

- MPU: ATmega328PB
- Communication module: HC-05 (Bluetooth module)
  - Low-cost, low-power Bluetooth module
- Sensor: LSM6DS0
  - IMU that we already have prior experience with
- Power: Power bank

Central Hub:

- MPU: ESP32
  - Provides enough compute power along with built-in Bluetooth.
- Amplifier: 1528-1696-ND (Digi-Key)
  - Receives the audio from the MCU in digital format over I2S. Handles conversion and amplification.
- Speaker: 668-1234-ND (Digi-Key)
  - Supports the output of the amplifier.
- Power: Power bank

8. Final Demo Goals

How will you demonstrate your device on demo day? Will it be strapped to a person, mounted on a bicycle, require outdoor space? Think of any physical, temporal, and other constraints that could affect your planning.

Our final demo shall be fairly simple. We will strap on the components and play a groove while sitting on a chair.

9. Sprint Planning

You've got limited time to get this project done! How will you plan your sprint milestones? How will you distribute the work within your team? Review the schedule in the final project manual for exact dates.

Milestone	Functionality Achieved	Distribution of Work
Sprint #1	Acquire hardware and start setting up base code	Pranay: reading ESP and ATmega for wireless communication; Haoran: reading manual for motion sensor; Kefei: reading manual for amplifier and speaker
Sprint #2	Develop full code for one node and the amplifier/speaker drive	Pranay: coding for wireless communication; Haoran: coding for motion detection; Kefei: coding for sound effect storage/call and amplifier/speaker driving
MVP Demo	Scale up to 4 nodes / putting it all together	Debugging together
Final Demo	Motion detection fine-tuning and more sound effects	Debugging together

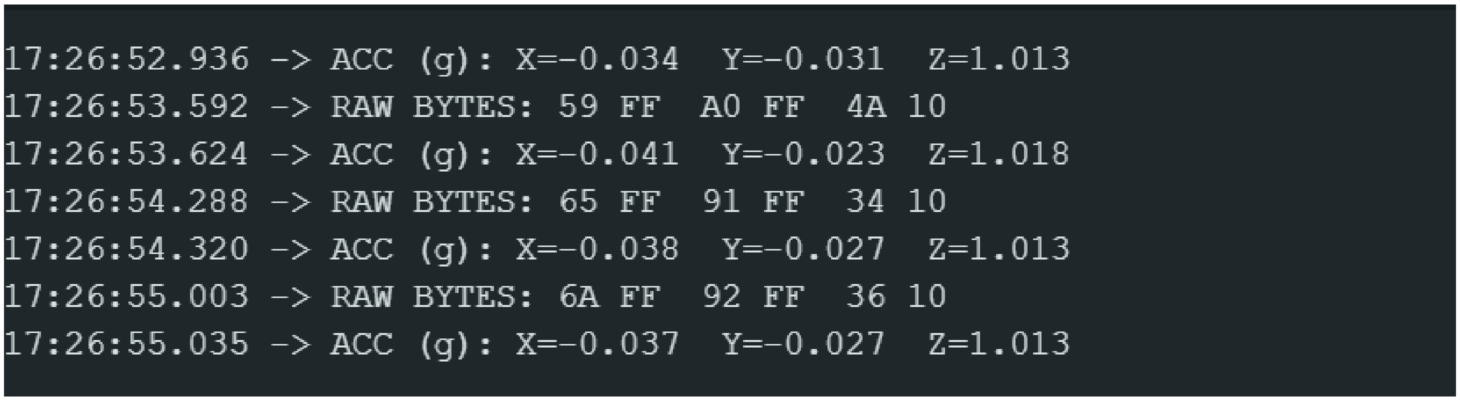
This is the end of the Project Proposal section. The remaining sections will be filled out based on the milestone schedule.

Sprint Review #1

Last week's progress

We wrote and tested the driver code for the IMU and I2C on the ATmega328PB sensor node. The firmware now initializes the LSM6DS0 over I2C, configures the appropriate control registers, and continuously reads out 3-axis acceleration data. We verified that the x, y, and z values update correctly when the node is moved.

We also started building the hit-detection logic on top of this data stream. Using the raw acceleration values, we experimented with simple thresholds around 2g and basic filtering to distinguish intentional downward “strike” motions from small jitters or upward movement.


```
17:26:52.936 -> ACC (g): X=-0.034  Y=-0.031  Z=1.013
17:26:53.592 -> RAW BYTES: 59 FF  A0 FF  4A 10
17:26:53.624 -> ACC (g): X=-0.041  Y=-0.023  Z=1.018
17:26:54.288 -> RAW BYTES: 65 FF  91 FF  34 10
17:26:54.320 -> ACC (g): X=-0.038  Y=-0.027  Z=1.013
17:26:55.003 -> RAW BYTES: 6A FF  92 FF  36 10
17:26:55.035 -> ACC (g): X=-0.037  Y=-0.027  Z=1.013
```

Current state of project

Right now, the motion detection pipeline on the node side is ready: the IMU is configured, the I2C driver is stable, and we can reliably observe clean acceleration traces that will be used for hit detection. We have a basic prototype node running on the bench that we can move and "swing" to see corresponding changes in acceleration.

On the hardware side, most of the remaining components (Bluetooth modules, amplifier, speaker, extra IMUs, etc.) have been selected and ordered according to our BOM, but several of them have not yet arrived. Because of this, we have not fully built out multiple nodes or the audio playback chain on the ESP32 yet. Overall, the project is on track, but progress on wireless communication and audio is currently gated by parts availability.

Next week's plan

Next week, we plan to move from sensing-only to end-to-end event transmission and audio output:

- Finish bringing up Bluetooth communication between one ATmega328PB node (via HC-05) and the ESP32 hub.
- Define and implement a compact "hit event" message format so that detected strikes on the node can be sent reliably to the central hub.
- Start implementing the audio driver on the ESP32 side by configuring the I2S peripheral and playing simple test tones or sample clips through the amplifier and speaker once the parts arrive.
- Integrate the existing motion detection code with the Bluetooth path so that a physical swing on the node can trigger a corresponding event at the hub, preparing us for full drum sound playback in the following sprint.

Sprint Review #2

Last week's progress

1. Built the Audio Playback System Using I2S

This week, we worked on getting audio output running through the ESP32's I2S interface. We configured the ESP32 as an I2S master in transmit mode and set it to 44.1 kHz, 16-bit stereo. We also mapped the bit clock, left/right clock, and data pins and verified the electrical connections with our amplifier.

2. Added a Simple Audio Mixer

We implemented a small mixer that allows multiple sound effects to play at the same time. Each sound keeps track of its own playback index, size, and active status.

In every cycle, the mixer reads samples from all active sounds, combines them with slight scaling to prevent clipping, and sends the mixed frame to the I2S driver.

3. Integrated WAV File Handling

We added logic to parse basic WAV headers so we can automatically skip the 44-byte header and extract the audio data size. Playback is non-blocking: each sound advances through its buffer inside the mixer instead of blocking the main loop.

4. Added Button-Triggered Sound Playback

We connected three buttons using internal pull-up resistors. Pressing a button activates its corresponding sound effect. Because the mixer is always running, multiple sounds can overlap without stopping each other.

5. Completed Initial Testing

We tested playback at 44.1 kHz and confirmed that the audio system runs reliably and that overlapping sounds mix correctly. The I2S driver and mixer structure are stable and performing as expected.

Current state of project

The current state of the project is:

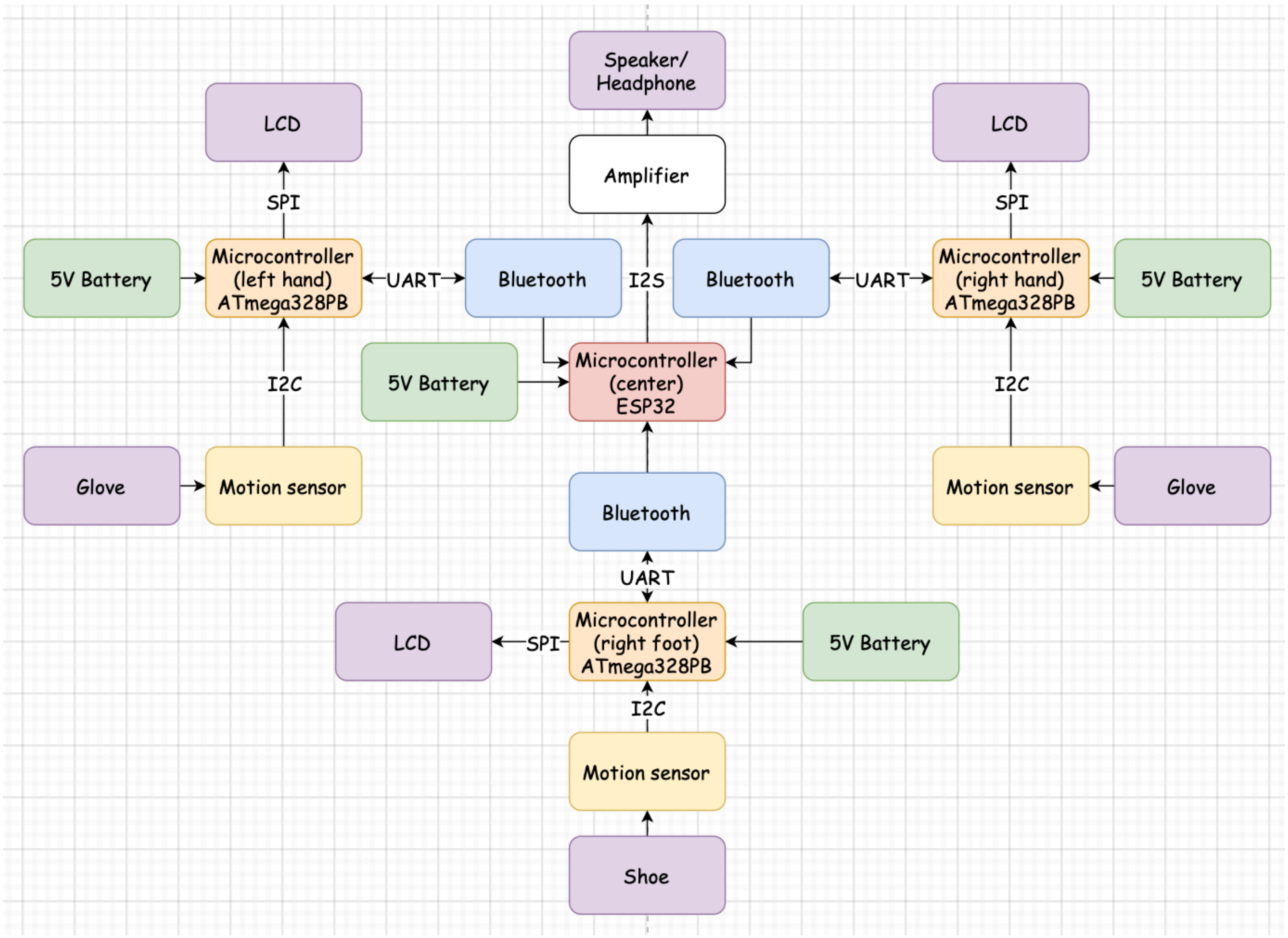
1. The drum strike detection using the motion sensor is ready.
2. The ESP32 is able to play audio files using the I2S protocol while retaining audio quality.
3. We were able to make one node communicate wirelessly using Bluetooth with the ESP32.

Next week's plan

Next week, we plan to integrate all the parts together and demo the final project.

MVP Demo

1. Show a system block diagram & explain the hardware implementation.



2. Explain your firmware implementation, including application logic and critical drivers you've written.

Sensor Node Firmware (ATmega328PB):

Each node initializes I2C, configures the LSM6DS0 IMU, and sets up UART to the HC-05 Bluetooth module.
It periodically reads acceleration data over I2C using a simple IMU driver (read/write registers, convert raw values).
A hit-detection state machine looks for downward acceleration above 2g and ignores upward motion or small jitters.

Debounce logic and a cooldown ensure that each physical swing only generates a single hit event.
On a valid hit, the node sends a compact “hit” message (with node/pad info) over UART to the Bluetooth module.

Central Hub Firmware (ESP32):

The ESP32 configures its I2S peripheral as a 44.1 kHz, 16-bit stereo master and loads drum samples from flash.
A lightweight WAV parser strips the 44-byte header and tracks raw PCM buffers plus playback indices.
A software mixer runs continuously, summing samples from all active sounds each audio frame and preventing clipping.
Hit events from nodes (received over Bluetooth/serial) are mapped to specific drum sounds and activate new playback instances.
Because playback is non-blocking and mixer-driven, multiple hits from multiple nodes overlap smoothly with low latency.

3. Demo your device.

https://drive.google.com/file/d/12pxA8k8FplrHLgwmTBjz-FNin05Deoml/view?usp=share_link

4. Have you achieved some or all of your Software Requirements Specification (SRS)?

We achieved all the SRS we previously set up except for switching from Wi-Fi to Bluetooth, and we are planning to add an LCD display as a status monitor for our system.

1. Show how you collected data and the outcomes.
   - Latency is negligible, so it is definitely shorter than 50 ms.
   - Detection threshold is set to a moderate acceleration so that we have a balance between sensitivity and mis-triggers.
   - Once turned on, all components are connected seamlessly and no abrupt disconnection has been observed.
   - Multiple sounds can be played at the same time, achieved by the mixer.

5. Have you achieved some or all of your Hardware Requirements Specification (HRS)?

We achieve most of them except for the structural integrity.

1. Show how you collected data and the outcomes.
   Sometimes a wire falls out of the slot and we need to rewire it. Since it is still in development, we don't want to solder those yet. Once we add the LCD module and reach the bare-metal complexity requirement, we will reinforce that for sure.

6. Show off the remaining elements that will make your project whole: mechanical casework, supporting graphical user interface (GUI), web portal, etc.

- A chassis that can hold the speaker in a proper position for sound quality.
- An LCD display implementation of system status monitoring for adding complexity on the bare-metal side.

7. What is the riskiest part remaining of your project?

Structural integrity is a concern for the system since it basically relies on glue and pin/wire friction right now.

1. How do you plan to de-risk this?
   We will have a chassis and wearable platform in the following week to address this.

8. What questions or help do you need from the teaching team?

System-level complexity check (should be fixed by adding the LCD).

Final Project Report

Don't forget to make the GitHub Pages public website!
If you've never made a GitHub Pages website before, you can follow this webpage (though substitute your final project repository for the GitHub username one in the quickstart guide):
https://docs.github.com/en/pages/quickstart

1. Video

Virtual drum Kit

- The video must demonstrate your key functionality.
- The video must be 5 minutes or less.

- Ensure your video link is accessible to the teaching team. Unlisted YouTube videos or Google Drive uploads with SEAS account access work well.
- Points will be removed if the audio quality is poor — say, if you filmed your video in a noisy electrical engineering lab.

2. Images

[Insert final project images here]

Include photos of your device from a few angles. If you have a casework, show both the exterior and interior (where the good EE bits are!).

3. Results

What were your results? Namely, what was the final solution/design to your problem?

3.1 Software Requirements Specification (SRS) Results

Based on your quantified system performance, comment on how you achieved or fell short of your expected requirements.

Did your requirements change? If so, why? Failing to meet a requirement is acceptable; understanding the reason why is critical!

Validate at least two requirements, showing how you tested and your proof of work (videos, images, logic analyzer/oscilloscope captures, etc.).

ID	Description	Validation Outcome
SRS-01	The IMU 3-axis acceleration will be measured with 16-bit depth every 100 milliseconds +/- 10 milliseconds.	Confirmed, logged output from the MCU is saved to the "validation" folder in the GitHub repository.

3.2 Hardware Requirements Specification (HRS) Results

Based on your quantified system performance, comment on how you achieved or fell short of your expected requirements.

Did your requirements change? If so, why? Failing to meet a requirement is acceptable; understanding the reason why is critical!

Validate at least two requirements, showing how you tested and your proof of work (videos, images, logic analyzer/oscilloscope captures, etc.).

ID	Description	Validation Outcome
HRS-01	A distance sensor shall be used for obstacle detection. The sensor shall detect obstacles at a distance of at least 10 cm.	Confirmed, sensed obstacles up to 15 cm. Video in "validation" folder shows tape measure and logged output to terminal.

4. Conclusion

Reflect on your project. Some questions to address:

- What did you learn from it?
- What went well?
- What accomplishments are you proud of?
- What did you learn/gain from this experience?
- Did you have to change your approach?
- What could have been done differently?
- Did you encounter obstacles that you didn’t anticipate?
- What could be a next step for this project?

References

Fill in your references here as you work on your final project. Describe any libraries used here.