Review the assignment due date

NIBLETS FINAL PROJECTION

- Team Number: 12
- Team Name: Niblets
- Team Members: Claren Ogira, Patrick Zhao, Praise Ndlovu

- GitHub Repository URL: https://github.com/upenn-embedded/final-project-s25-niblets
- GitHub Pages Website URL: [for final submission]

Final Project Proposal

1. Abstract

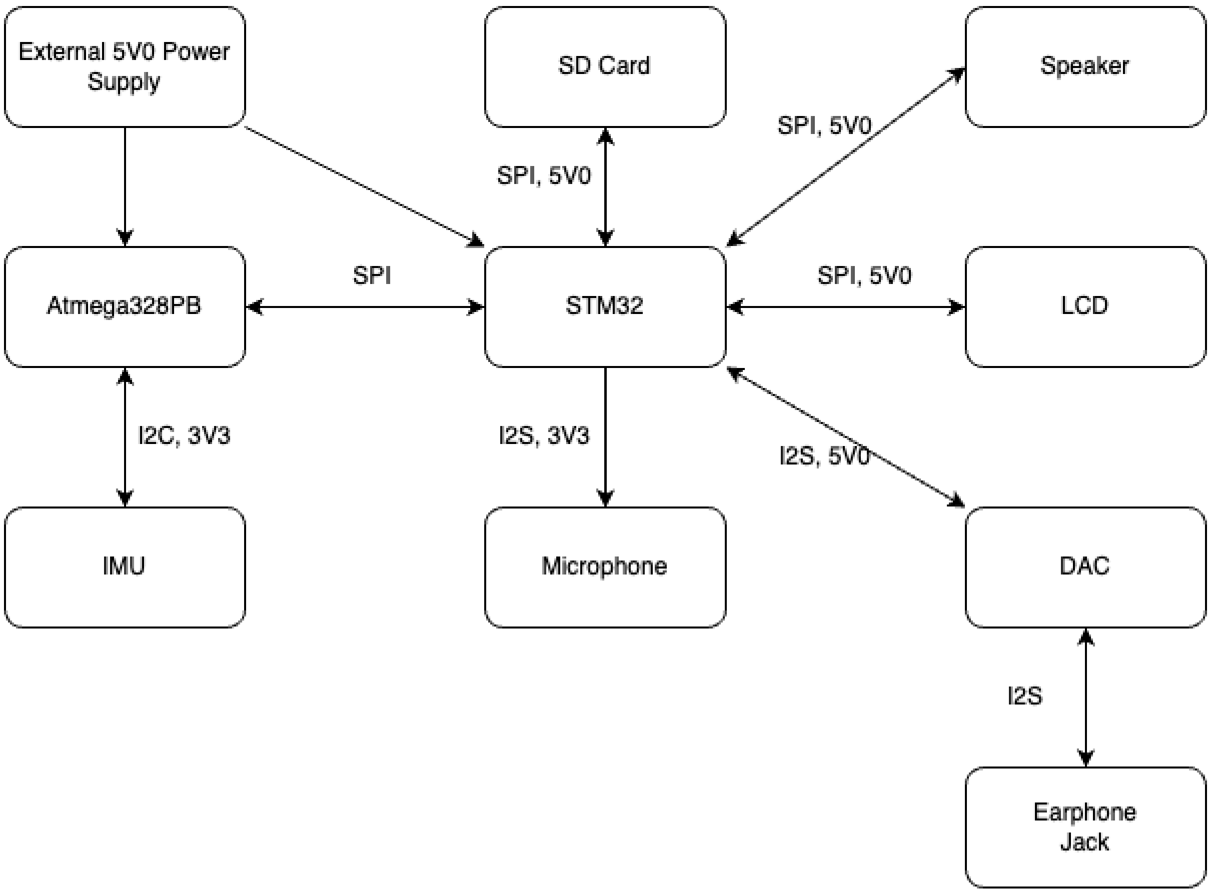
This project develops a digital audio device integrating an LCD, microphone, SD card, speaker, and microcontrollers for seamless audio playback, recording, and user interaction. The system ensures efficient communication, real-time performance, and reliable data storage. Validation includes display responsiveness, audio quality, and peripheral interfacing to optimize functionality and usability.

2. Motivation

This project builds a standalone digital audio device for recording, playback, and user interaction. It’s a hands-on dive into writing graphics, mastering serial protocols, and making hardware talk seamlessly. The goal? A fully functional, ready-to-use device—while leveling up in embedded systems and low-level programming!

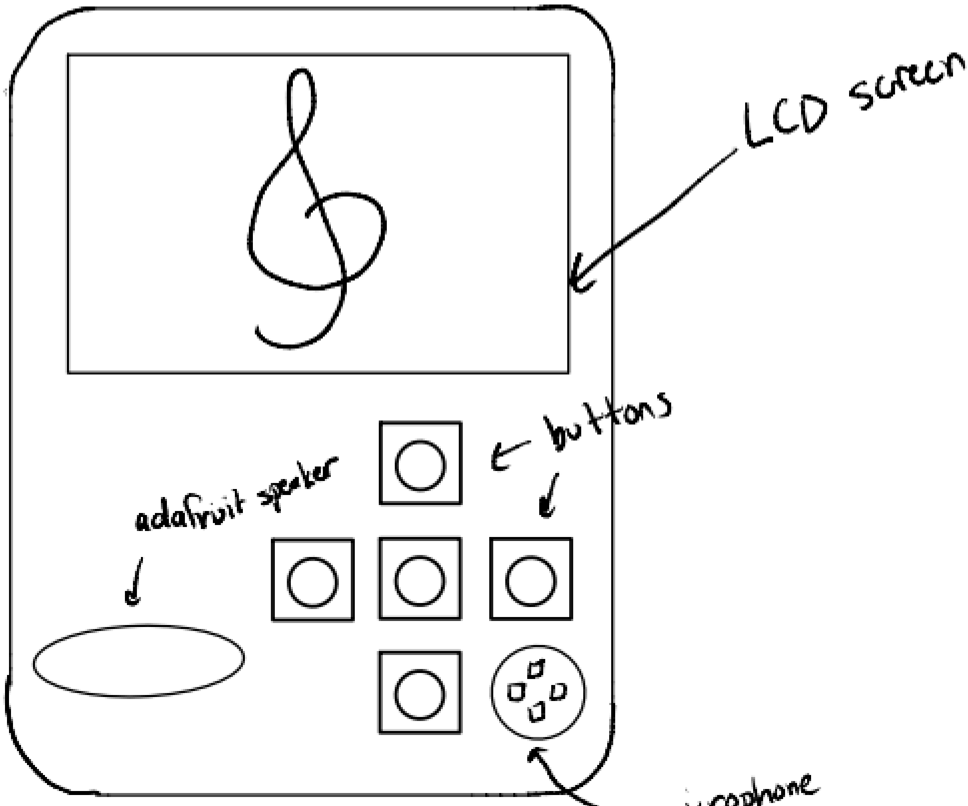
3. System Block Diagram

Show your high level design, as done in WS1 and WS2. What are the critical components in your system? How do they communicate (I2C?, interrupts, ADC, etc.)? What power regulation do you need?

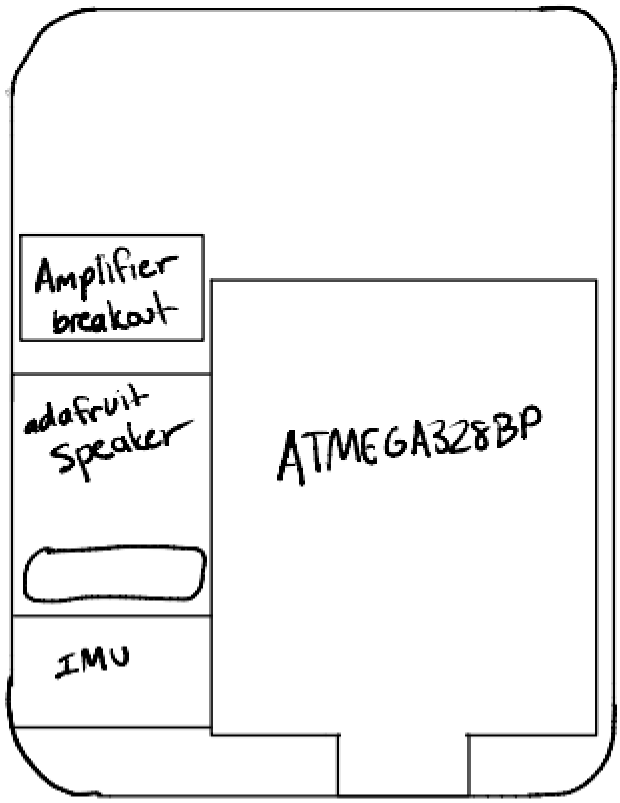


4. Design Sketches

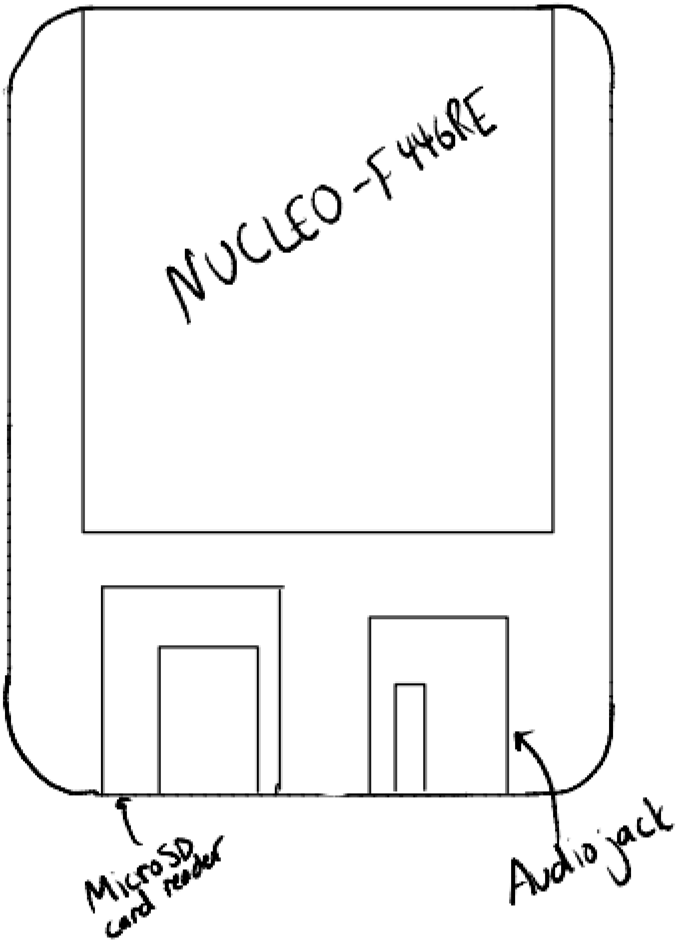
What will your project look like? Do you have any critical design features? Will you need any special manufacturing techniques to achieve your vision, like power tools, laser cutting, or 3D printing?

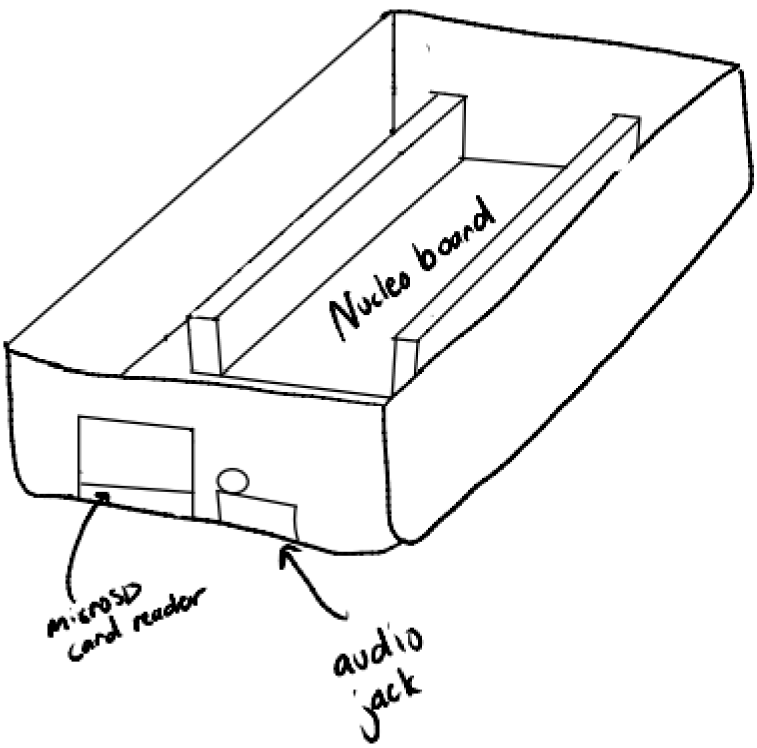


Layer 1



Layer 2





We wll need 3D printing to print out a case for fitting all of our components in. We may also require screws to place our components in.

5. Software Requirements Specification (SRS)

Formulate key software requirements here. Think deeply on the design: What must your device do? How will you measure this during validation testing? Create 4 to 8 critical system requirements.

These must be testable! See the Final Project Manual Appendix for details. Refer to the table below; replace these examples with your own.

5.1 Definitions, Abbreviations

Here, you will define any special terms, acronyms, or abbreviations you plan to use for hardware

5.2 Functionality

ID	Description
SRS-01 - LCD Display Functionality	The LCD must accurately render graphics and display the current state of the iPod, including menus, playback status, and user interactions.**Validation**: Verify display clarity, responsiveness, and update rate during different operations.
SRS-02 Microphone Recording Capability	The microphone must capture audio input with sufficient clarity and store it in a digital format when enabled.**Validation**: Record test samples and analyze audio quality, noise levels, and latency.
SRS-03 SD Card Storage & Read/Write Operations	The SD card module must support reading and writing data efficiently, ensuring reliable storage of music, recorded audio, and system logs.**Validation** Conduct read/write speed tests, check file integrity, and test compatibility with different SD card sizes.
SRS-04 Speaker Audio Output	The speaker must play sound accurately and at a sufficient volume without distortion when provided with an audio signal.**Validation** Measure frequency response, output power, and signal clarity under different playback conditions.
SRS-05 Microcontroller (MCU) Interfacing	The microcontrollers must communicate seamlessly with each other and with peripherals such as the LCD, microphone, SD card, and speaker.**Validation** Perform communication protocol tests (e.g., I2C, SPI, UART),

ID	Description
	measure data transfer latency, and ensure stability under various loads.
SRS-06 User Interface Responsiveness	The system shall provide an intuitive and responsive user interface for navigation and control.**Validation** Test button responsiveness, menu transitions, and overall usability during different operations.

6. Hardware Requirements Specification (HRS)

ID	Description
HRS-01 - LCD Display	The LCD display must have a minimum resolution of 480x320 pixels and support clear, legible display of menus, playback status, and user interactions**Validation**: Verify display clarity, resolution, and update rate during different operations.
HRS-02 - Microphone	The microphone must capture audio with high fidelity and low noise, providing clear input for recording and voice commands.**Validation**: Record test samples, analyze audio quality, noise levels, and latency.
HRS-03 - SD Card Module	The SD card module must support reading and writing at fast speeds to efficiently handle large files such as audio and system logs.**Validation**: Conduct read/write speed tests, check file integrity, and test compatibility with different SD card sizes.
HRS-04 - Speaker	The speaker must provide clear and undistorted audio output, with sufficient volume for playback in typical environments.**Validation**: Measure frequency response, output power, and signal clarity under different playback conditions.
HRS-05 - Microcontroller (MCU)	The microcontroller must manage the system’s components, handle user input, audio processing, and communication between peripherals.**Validation**: Perform communication protocol tests (e.g., I2C, SPI, UART), measure data transfer latency, and ensure stability under various loads.

ID	Description
HRS-06 - Power Supply	The power supply must support efficient power management to ensure optimal device operation and battery longevity.**Validation**: Measure power consumption under different workloads and test battery performance over extended usage.
HRS-07 - Buttons & User Interface Controls	Physical buttons or touch interface must allow the user to navigate the system and control media playback and settings.**Validation**: Test button responsiveness, UI transitions, and overall usability during different operations.

6.1 Definitions, Abbreviations

Here, you will define any special terms, acronyms, or abbreviations you plan to use for hardware

6.2 Functionality

ID	Description
HRS-01	The audio player shall support playback of audio files from the SD card, with support for common formats such as MP3 and WAV.
HRS-02	The device shall include a play/pause button that toggles audio playback without delay.
HRS-03	The volume control shall allow the user to adjust audio output levels from silent to a maximum of 85 dB.
HRS-04	The audio player shall support track navigation, allowing the user to skip forward or backward between audio tracks.
HRS-05	The device shall display the current track information, including title and duration, on the LCD screen.
HRS-06	The speaker shall output clear and undistorted audio at all playback levels.
HRS-07	The audio player shall automatically resume playback from the last position upon power-on, if no other track is selected.

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.

Some components that we will need are the ATMEGA328PB which we intend to utilize with the IMU since we already know how the two devices interface given our worksheet experience. We want to use the NUCLEO-F446RE as our main process since it is able to process mode data, which will be required when we are storing and playing music data, and since it has more pins that will allow us to interface with all of our components.

Some output components that we are using are our adafruit speaker, which we find to be standard, an audiojack to allow earbud use and an LCD display touch panel. We chose the LCD display with the touch functionality because of both its touch functionality and because of the bigger display and its wider range of colors.

Our choice of the omnidirectional microphone both reflected on the device's small size and the device's ability to internally process the data that it receives with its internal DAC. We realized that an internal DAC within the microphone would save us tremendous time when building out our device. The same internal DAC reasoning was used in our selection of the amplifier used.

Finally, the IMU we chose reflected an IMU that we are already familiar with and we chose larger buttons for a more consumer-oriented feel.

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/1H9uE5A8rThZXxHaYCtSz2vRuI-nbI0XPYlmzn1kPBnA/edit?gid=2071228825#gid=2071228825

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.

We will demonstrate our project in class. The end product will be small enough to fit on a table. The device may be connected to an external power source to supply power to the system. A quiet setup may be required to hear the audio produced by the device, and to record audio using the microphone.

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	MicroSD Card Reading. LCD turning on with basic graphic functionality. Microphone recording voice. IMU sending understandable data to device. Speaker should be playing some audio that is controllable by both processors. Serial protocols all should be completed as well	Claren - Microphone voice recording, Praise - LCD turning on with basic graphics, Patrick - Speaker playing audio and SD card reading
Sprint #2	Start integration of components. Get audio jack working with amplifier. Start playing music through SD card read through speaker and be able to store recorded audio. Buttons should start controlling board features. LCD screen should display a more understandable graphic. CAD of case should be designed.	Patrick - Design CAD of case. Praise - Playing music through SD card and selection through LCD screen. Claren - Connect audio jack with amplifier and play music through earbuds.
MVP Demo	All components connected and fit inside case. Working minimal demo of song selection and music playing through device.	Patrick - Assembling case, soldering needs. Praise - Connect all of the software components together to play audio. Claren - Work on graphics of LCD screen to allow song selection and better experience.

Milestone	Functionality Achieved	Distribution of Work
Final Demo	Hopefully little to no work left here. Finish any features that were left unfinished from MVP. Program better graphics in the processor for the LCD	Patrick, Praise, Claren - Work on assorted problems to debug at finish ine.

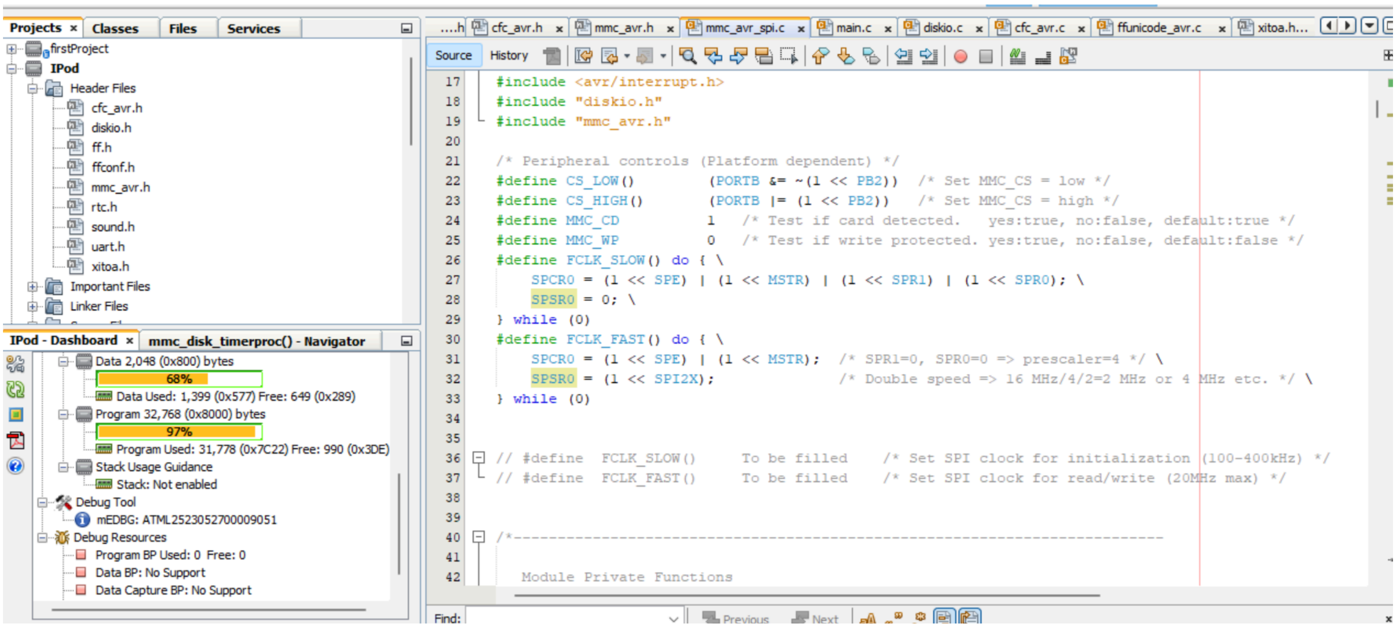
This is the end of the Project Proposal section. The remaining sections will be filled out based on the milestone schedule.

Sprint Review #1

Last week's progress

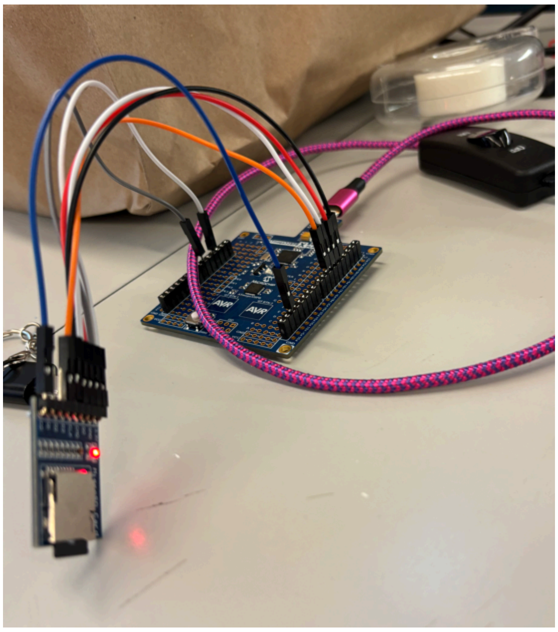
Patrick - I have been working through interfacing our ATMEGA328PB with the SD card reader using SPI which has been difficult. I've conducted some research and found the FATfs library as a way of file management and reading the data through buffers. I was able to change the configurations of an example FATfs project from ATMEGA1284P to our current device, but I think there are still some issues with the current configurations. Below is a picture of us interacting with the FATfs library after we ran it on our ATMEGA where we are able to initalize mounting, but the file reader isn't displaying any values even though the SD card reader board and SD card reader is connected to our microcontroller. The bulk of the work went into reading through the library and understanding how each part of it works mechanically which has taken a long time. `<img width="214" alt="image" src="https://github.com/user-attachments/assets/e303c498-f2db-4be8-b9d2-67991d18a81f" />`

The code is too long to place here, so we've included a screenshot of us interfacing FATfs with the ATMEGA328PB (most of the program memory is consumed by the library!)



Please refer to the FATfs library code that we've uploaded and have been modifying.

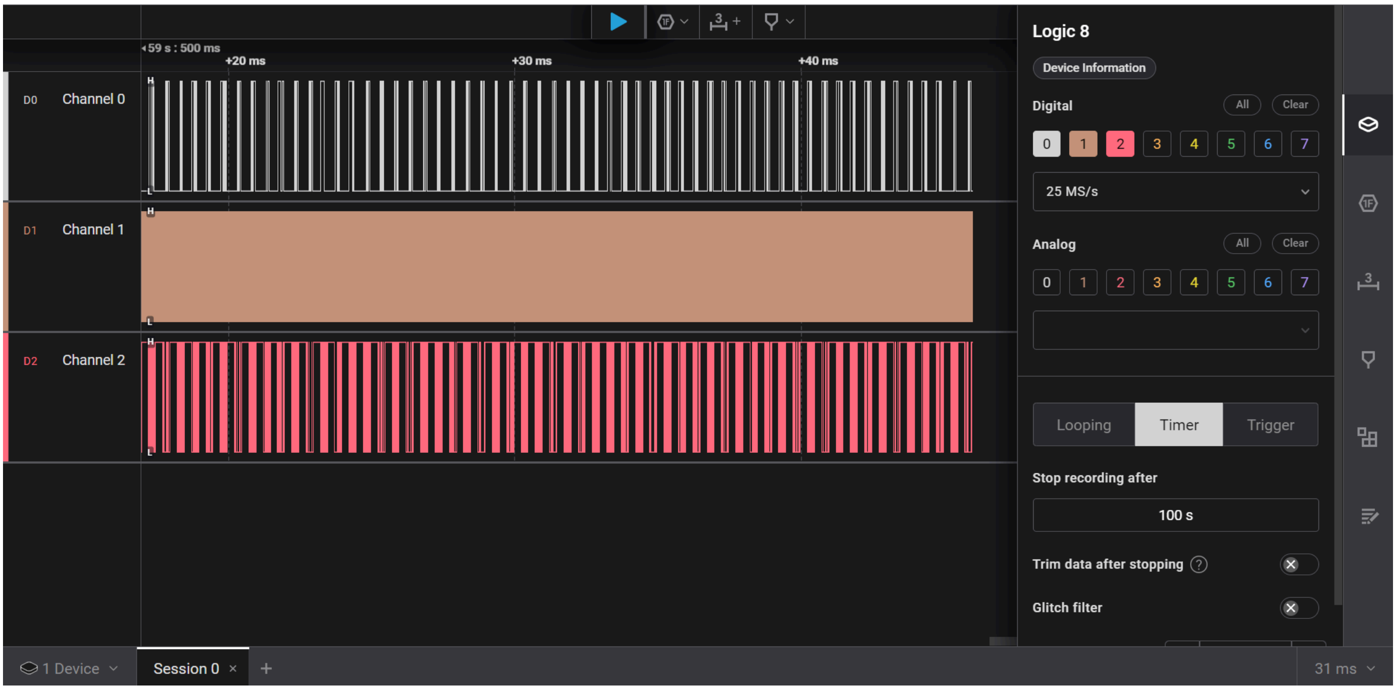
Below is a picture of our wired AVR with our SD Card Reader Board.



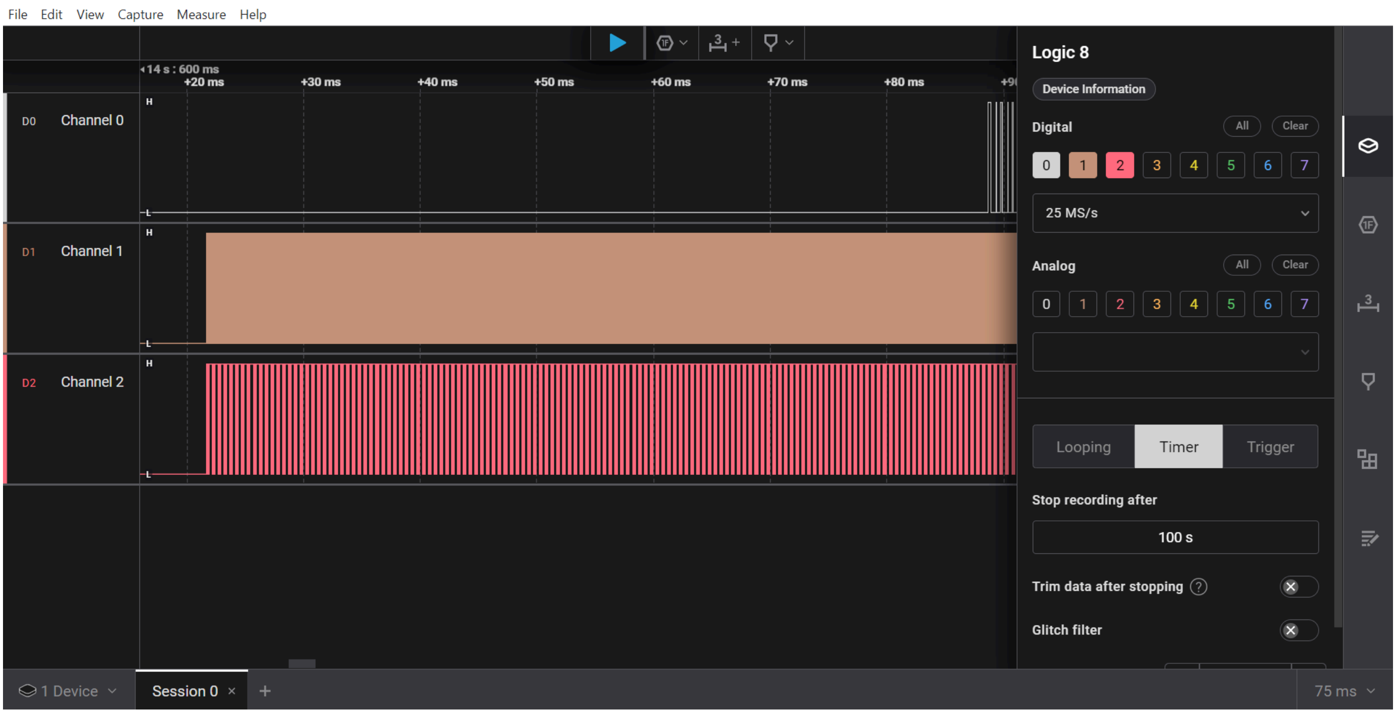
Claren - I worked on setting up the I2S protocol betweeen the Atmega328PB and the INMP 441 microphone. The remaining task regarding this is interfacing with the rest of the project components. Below are the updates from this week's work:

Logic Analyzer Images

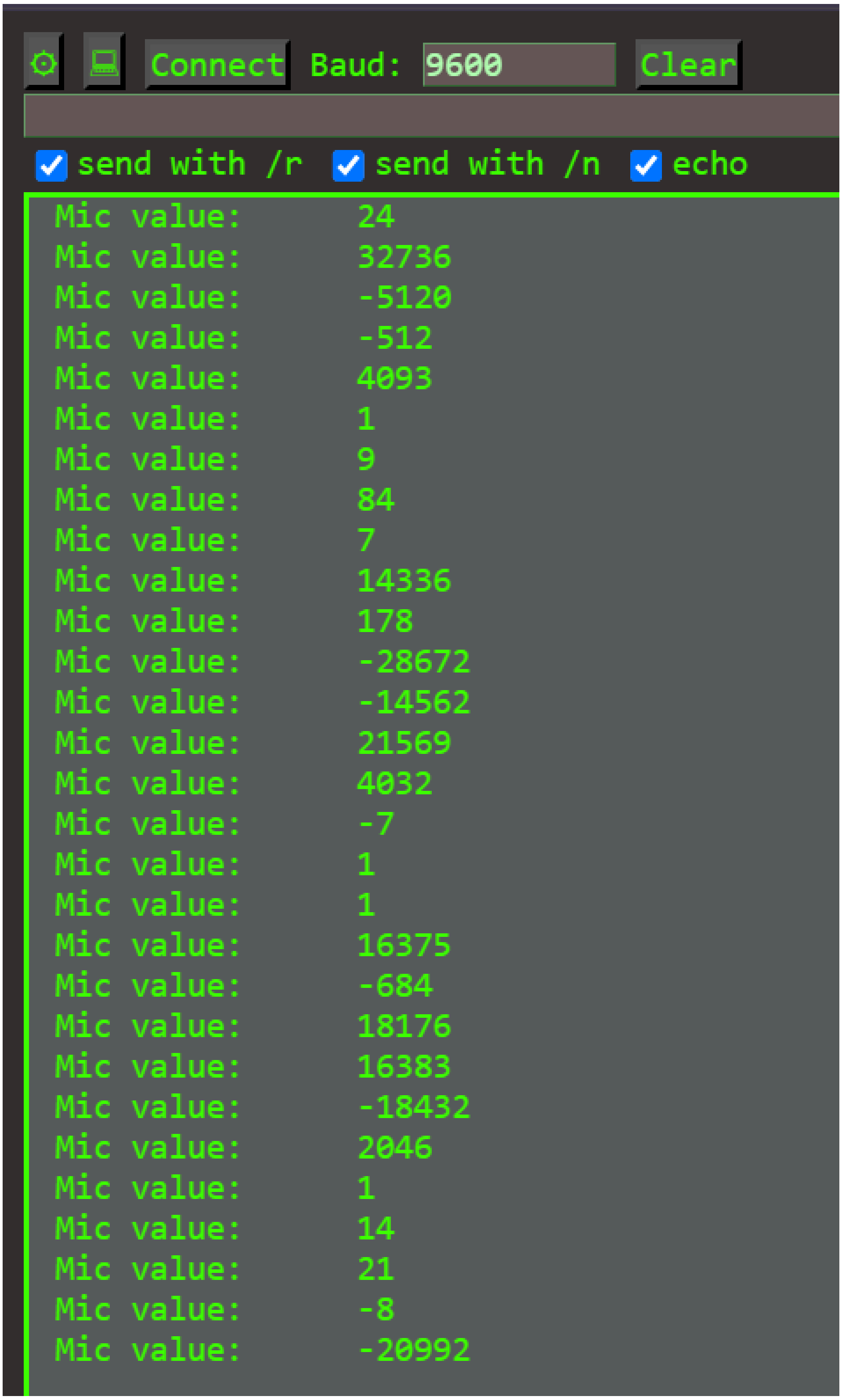
For the logic analyzer images below, Channels 0, 1, and 2 refer to SDA, SCL and WS respectively.



According to the datasheet, SDA stays low for 2^18 clock cycles. I found this to be about 70ms which we should account for in our readings. This could be accounted for by a delay or maybe boot other operations as the microphone sets up.



The values printed on the serial terminal:



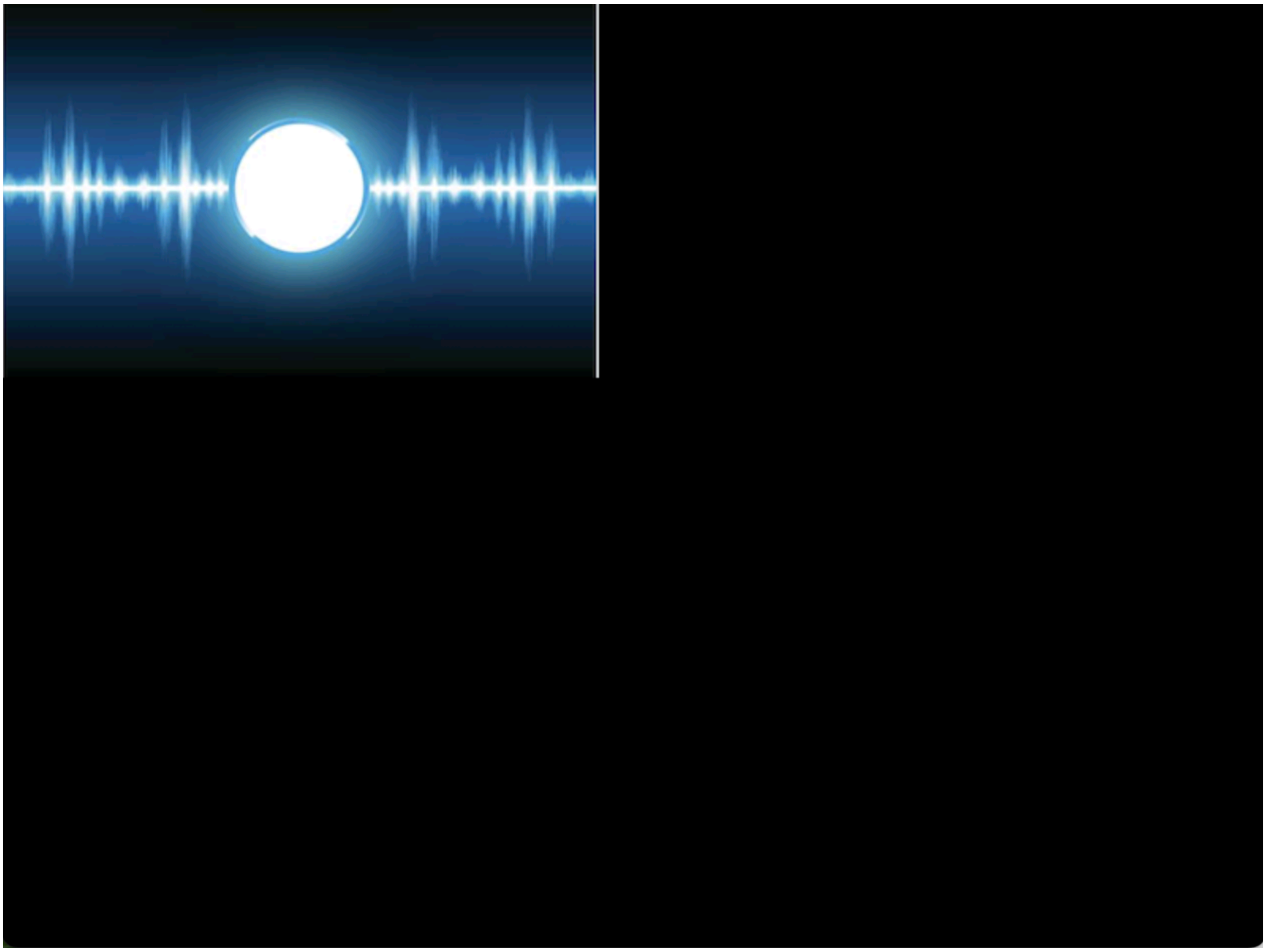
Praise - I have been setting up our STM32 project and writing the initialization code for all the essential peripherals. This includes configuration for GPIOs, I2C, SPI, timers, and syscalls, which form the foundation of our embedded application. The SPI interface is still undergoing testing: I attempted to communicate with the ICM-20948 IMU using SPI, and so far, I have received 0x0F as a response when reading the WHO_AM_I register, which suggests a fault or misconfiguration. I am currently debugging this by checking the SPI settings (mode, prescaler, CS handling) and verifying physical connections.

In parallel, I have also been working on the LCD module. It is partially functional—some data is being displayed, but the refresh rate is too low, and output only appears on half of the screen. I suspect this is due to incomplete initialization, possibly missing one or more commands required by the LCD controller's startup sequence. I am currently cross-referencing the controller datasheet with the initialization routine.

To speed up development and reduce iteration time, I created a simulation workspace on my laptop that uses SDL graphics to simulate the GUI. This allows us to test the visual components of the UI before deploying them on hardware. The SDL environment is functioning properly and mimics our target LCD resolution and layout.

Lastly, I wrote a bitmap-to-pixel conversion utility to convert any image into the pixel array format required by our LCD driver. This will make it much easier to render static images or UI elements such as logos or backgrounds.

I'll upload screenshots of the partial LCD display and simulated GUI once we finalize the layout. The current SPI and LCD code is being refined and will be pushed to GitHub once it stabilizes.



Current state of project

In terms of parts, we have everything we need and don't expect to need anything in the near future. One of us also has a 3D printer that we have ready access to which we will use for making the case so there shouldn't be any problems.

We haven't been able to read data from an SD card yet which we expected to be the hardest part of this project. We intend to work on this throughout this weekend to fix this problem and get it reading txt files first, and then read WAV files from the SD card. The problem is that the SD card isn't being recognized which we suspect is an issue with our setup.

We have been able to set up the microphone which is essential for saving data that we will later play back to the user if they choose to record themselves. Since this part is mostly complete, we will just look to integration in the future.

We have been working on building out the renders that will be made on the LCD screen which is almost ready. We have also been setting up the LCD screen where we are still looking for a few more addresses to write data to. This is an important part of the project since we need to both display our data and use the LCD screen as a touch screen device.

The bulk of the work is to configure the hardware we have to our microcontrollers given that we are leveraging a few complex libraries to complete some of the harder components such as file management when utilizing the SD card. The backup plan is to store the data on the cloud and send it to the STM32 using an ESP32 if we are unable to utilize the SD card reader board.

Next week's plan

1. Reading data from the SD card `<br>` a. We expect this to take another 5 hours. Debugging the code is taking a long time and requires use to better understand the FATfs library for file management. `<br>` b. Patrick and Praise `<br>` c. When we run the example code provided by the FATfs library for AVR, we want that when we write "fl /" to the serial terminal to see a return of FR_OKAY and of the folders in the SD card. `<br>` d. Since we are listening to music from SD card readings, we need our Atmega328pb board to be able to extract music data from an SD card and play it later. To accomplish this, we need to utilize a file

management system that lets us select specific files and which we will then read and use to play music. We were able to find code online that we can use as a file management system, but reading and understanding the code has taken a long time. We are also configuring the code that we found online to fit our device as well as our needs. We need to make sure that this completely works before we can start playing data and doing other things. If reading from the SD card doesn't work by Monday, we are planning to use data from the cloud instead by using an ESP32 and sending data from there into the STM32. `<br>` `<br>`

2. Integrate data reading from SD card onto STM `<br>` a. We expect this to take 3 hours. Once we have data reading from SD card onto the AVR, we can easily shift it onto the STM32 given we change some of the input / output ports. `<br>` b. Praise `<br>` c. We want the STM to be able to read data from our input on an SD card and then to print the names of the files inside of the SD card. `<br>` d. We need to make sure that the STM is able to read data in the same way as we program the AVR. `<br>` `<br>`

3. Playing music from data read on STM `<br>` a. We expect this to take 5 hours. Once we are able to read data from the SD card onto the STM, we need to be able to interpret that data and play it on our mono speaker `<br>` b. Claren and Patrick `<br>` c. We ultimately want to play a simple song (like twinkle twinkle) and see if we are able to play it through our mono speaker. We want the sound quality to be recognizable. `<br>` d. This is one of the more essential parts of our project, we need to be able to play music that we read from the STM. This will involve getting the data from the WAV files that we load onto our STM and being able to convert them to frequencies to play on our speaker. `<br>` `<br>`

4. CADing a case to contain everything `<br>` a. We expect this to take 8 hours. We need to CAD a case both large enough to contain all of our components while also being small enough that can be handheld. `<br>` b. Patrick `<br>` c. For this task to be finished, I want to have created a CAD that contains all of our components within.`<br>` d. This is a very important step in terms of creating housing for our device. We need to start early in terms of building our case because we will probably want to iterate in the future and printing will take a lot of time. `<br>`

5. Fix SPI communication with IMU (ICM-20948) `<br>`

   a. Estimated time: 4 hours`<br>`

b. Assigned to: Praise

c. Definition of done: Reading the correct WHO_AM_I value (0xEA) from the IMU via SPI consistently.

d. Detailed: This involves confirming SPI timing, polarity, and chip-select handling. If needed, I’ll use a logic analyzer to verify the signals.
``

6. Debug and finalize LCD initialization

   a. Estimated time: 5 hours

   b. Assigned to: Praise

   c. Definition of done: Full-screen content is displayed correctly with stable refresh rates and no visible flickering.

   d. Detailed: I’ll review command sequences from the datasheet and compare them with known-good examples online. If the issue persists, I will test alternate initialization sequences.

7. Push finalized SPI and LCD code to GitHub

   a. Estimated time: 1 hour
 b. Assigned to: Praise
 c. Definition of done: SPI and LCD drivers are committed with documentation and usage examples in the repository.
 d. Detailed: I will add comments and organize the code into modular drivers with initialization and usage APIs.

8. Integrate LCD GUI elements using SDL simulation output

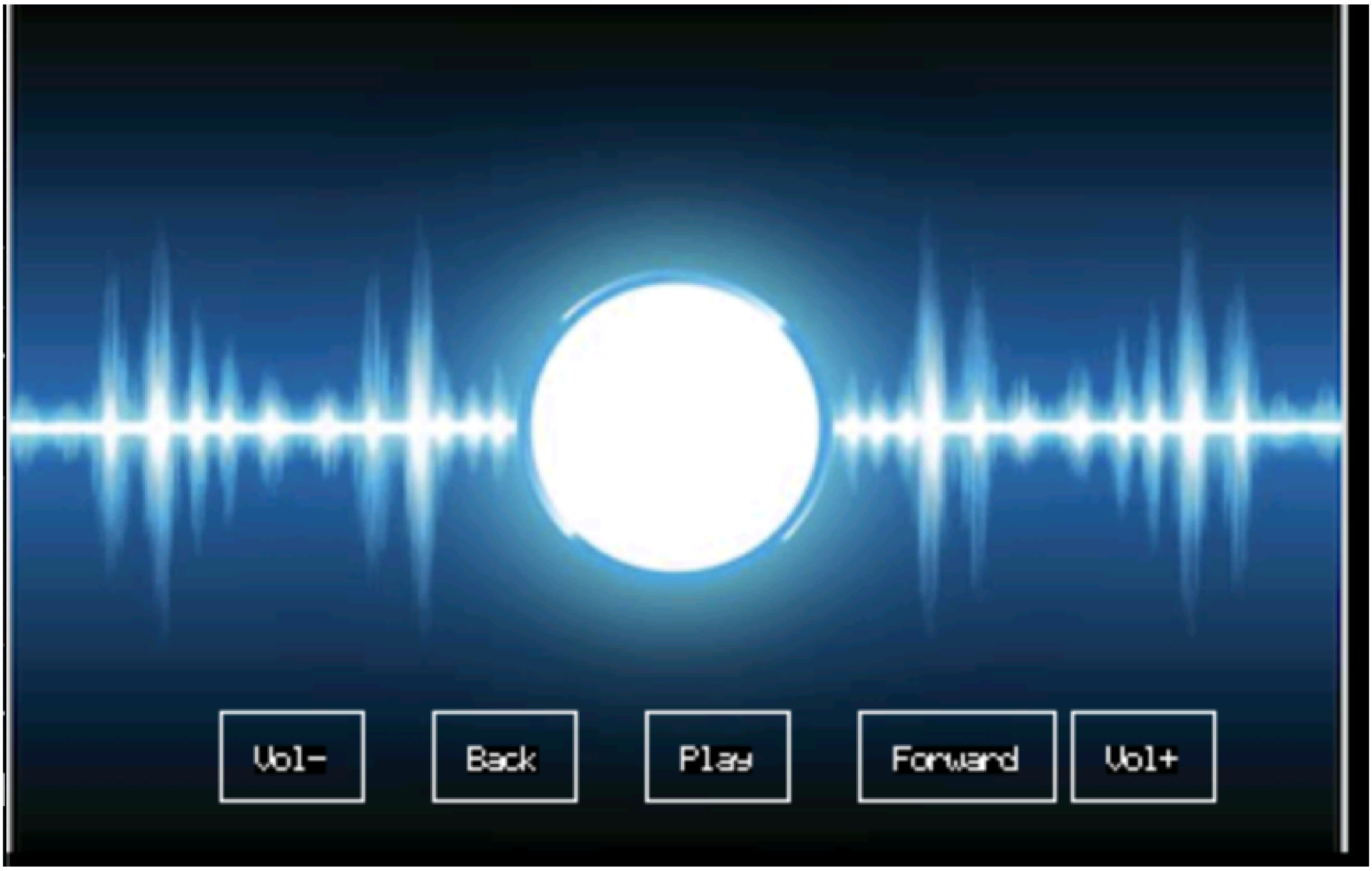
   a. Estimated time: 3 hours
 b. Assigned to: Praise
 c. Definition of done: At least two GUI screens (e.g., boot screen and music selection screen) working in the simulation and partially on the real LCD.
 d. Detailed: I will transfer the screen logic from SDL to the hardware LCD once it displays correctly in the simulation.

Sprint Review #2

Last week's progress

Claren - For this week I focused on designing the user interface and designing the functionality. This involved adding more versatile functions to the SDL_GFX.h library to improve our flexibility in laying out our button design. We then tested the design on a simulation that we developed on our laptops to reduce the amount of time to test and simplify the process. Afterwards, the GUI can be uploaded into the MCU and ran on the LCD. The images of the GUI are shown below:

Initial design:



Second design:

Patrick - This week I worked on making sure that we had working I2C readings from the IMU and the STM32. Specifically, I focused on configuring gyroscopic readings from the IMU and being able to print those values to the serial terminal. I was able to achieve this by using the STM32's built in HAL library to send and receive data through I2C. I was then able to obtain gyroscopic readings with some slight bias that we expect can be corrected.

I also spent some time thinking through how we might want to integrate the IMU as a screen flipper onto our device. I realized that it would be really difficult to constantly track the roll, pitch, and yaw of our device, so I decided it would be easier to track large movements in angular velocity to use as our metrics for screen flipping instead. In this way, the user could initiate a flip by flipping the device quickly in one direction and would not require a lot of recalibration. This would be the next step in our project in adding a rotating screen.

```
Gyro X: -20 dps, Gyro Y: 3 dps, Gyro Z: 0 dps
Gyro X: -16 dps, Gyro Y: 3 dps, Gyro Z: 1 dps

Gyro X: -16 dps, Gyro Y: 1 dps, Gyro Z: -1 dps
Gyro X: -19 dps, Gyro Y: 0 dps, Gyro Z: -3 dps
Gyro X: -18 dps, Gyro Y: 4 dps, Gyro Z: -1 dps
Gyro X: -21 dps, Gyro Y: 0 dps, Gyro Z: 0 dps
Gyro X: -20 dps, Gyro Y: 0 dps, Gyro Z: -3 dps
Gyro X: -17 dps, Gyro Y: 0 dps, Gyro Z: -4 dps
Gyro X: -19 dps, Gyro Y: 1 dps, Gyro Z: -6 dps
Gyro X: -21 dps, Gyro Y: 1 dps, Gyro Z: -3 dps
Gyro X: -20 dps, Gyro Y: 1 dps, Gyro Z: -5 dps
Gyro X: -20 dps, Gyro Y: 2 dps, Gyro Z: -2 dps
Gyro X: -18 dps, Gyro Y: 2 dps, Gyro Z: -2 dps
Gyro X: -15 dps, Gyro Y: 0 dps, Gyro Z: -4 dps
Gyro X: -17 dps, Gyro Y: 0 dps, Gyro Z: -2 dps
Gyro X: -20 dps, Gyro Y: 2 dps, Gyro Z: -2 dps
Gyro X: -18 dps, Gyro Y: 0 dps, Gyro Z: -5 dps
Gyro X: -17 dps, Gyro Y: 1 dps, Gyro Z: -5 dps
Gyro X: -22 dps, Gyro Y: 3 dps, Gyro Z: -3 dps
Gyro X: -17 dps, Gyro Y: 0 dps, Gyro Z: -2 dps
```

A video of our implementation of the IMU is shown below.
https://drive.google.com/file/d/1kdk18VxoSriLACKgJFGk2Km8MMnqj_ot/view?usp=sharing

We have also been working on getting the STM to work with our LCD screen. The issue with the integration so far has been that the original LCD screen we wanted to use had some technical errors, so we are reverting to the screens that we worked with in class. We are working through this by using the libraries we used in class and going from there.

I have also been compiling the CAD parts that we would be using with our project to prepare for CAD-ing for our case. I've found most of the parts as STEP files which would be good for using with Fusion.

My Drive > ESE 3500 CAD PARTS

Type | People | Modified | Source

Name	Owner	Last modified	File size
INMP441 I2S Microphone Module v15.step	me	Jan 21, 2024 me	1.8 MB
LGA-14L_2P5X3X0P86_STM.step	me	Sep 17, 2021 me	86 KB
LSM6DSO.SLDASM	me	Sep 17, 2021 me	346 KB
LSM6DSO.SLDPRT	me	Sep 17, 2021 me	368 KB
MAX98357A_3W_I2S_AMP v27.step	me	Dec 29, 2023 me	5.9 MB
NUCLEO-F446RE.STEP	me	Apr 10, 2025 me	20.6 MB
NUCLEO-F446RE.STEP	me	Apr 10, 2025 me	20.6 MB
TFT LCD 2.4 inch ILI9341.step	me	Mar 27, 2024 me	419 KB
TRRS20153.step	me	Dec 14, 2020 me	49 KB
TRRSBreakout.step	me	Dec 14, 2020 me	64 KB

Praise I have been working with being able to play audio through the STM32 with stereo speakers and was able to due so this week with our amplifiers. Here is a link to a video of our speakers playing a short snipped of music:

https://drive.google.com/file/d/11OGU_7PfAK1LRACp8a9eyxocnXaY17er/view?usp=sharing. Additionally, I have been working on building out a SPI connection so that we can connect to our LCD screen. I've succesfully tested SPI communication with our IMU so we know that the functions I wrote for SPI communication do work. Patrick will handle the integration of the SPI with LCD from here.

Current state of project

We have all the necessary parts to move forward with our project and it is just a matter of integration.

Patrick - On my end, I think there's been good progress with using I2C and now need to integrate the readings from the gyroscope with the LCD screen display flipping once we have the display working with the STM32. I will then need to work with Claren in order to integrate his GUI work with our current design.

Praise I have successfully played audio through a speaker using the I²S protocol via STM32 HAL. I have also implemented the FATFS file system on HAL and verified correct reading of WAV files and sampling frequency. I am currently transitioning to a bare-metal implementation of the I²S protocol and will soon begin work on microphone integration, with the goal of playing audio directly from an SD card using DMA transfers.

Next week's plan

1. **IMU Detecting Screen Rotations**
   a. Estimated time: 3 hours
   b. Assigned to: Patrick
   c. Definition of done: IMU prints which direction the screen should flip to if enough angular change in one direction is sensed. .
   d. Detailed: We need to make sure that we are using the readings from the IMU to actually flip the screen. That means taking time to understand how the roll, pitch, and yaw could actually work together to interpret which way the LCD screen should be facing (downward, upward, sideways some way). I think this will take 3 hours to figure out and program orientation flips into.

2. **LCD Screen Working with STM32**
   a. Estimated time: 3 hours
   b. Assigned to: Patrick
   c. Definition of done: For this task to be completed, we should have the LCD screen displaying the name of a song we are planning to play.
   d. We need to integrate our LCD screen with our STM32. Since we are using the code from our worksheet 3 lab, I don't expect this task to take a long time, and I think we will be able to configure the LCD screen with letters printed on it in 3 hours.

3. **Integration of GUI with LCD Screen**
   a. Estimated time: 2 hours
   b. Assigned to: Patrick and Claren
   c. Definition of done: Image and buttons that Claren built for GUI displayed on physical LCD screen.
   d. Once we finish both the LCD screen setup and the GUI design, we will look to integrate the two together by actually showing the GUI design on the LCD screen. We expect this to take some time given that there may be difficulties displaying images that we have set up on the GUI, but given that (Claren and Patrick) will have spent a lot of time working with both components, we don't think it will take more than 2 hours to get a working prototype.

4. **Bare-metal I2S Playback (Speaker Audio Output)** Estimated Time: 3–4 hours Assigned to: Praise Definition of Done: Audio file played from STM32 via I2S in bare-metal (non-HAL) implementation. Details: You've successfully tested the speaker output using I2S with HAL and verified the playback frequency. Now, the task is to migrate the I2S speaker playback logic to a fully bare-metal implementation. This will involve configuring the I2S peripheral registers directly and ensuring audio data from memory is sent reliably using DMA or polling.

5. **Bare-metal I2S Input (Microphone Integration)** Estimated Time: 3–4 hours Assigned to: Praise, Claren Definition of Done: Raw data successfully read from the INMP441 microphone using I2S in bare-metal. Details: With I2S output working, the next step is to read input from the INMP441 MEMS microphone. This involves configuring I2S in receive mode, validating microphone connection, and verifying data integrity. The microphone's PDM output must be captured properly and can be visualized (e.g., via UART output) for debugging.

6. **SD Card Playback Integration** Estimated Time: 4 hours Assigned to: Praise, Claren Definition of Done: WAV or raw audio file is read from SD card and played through the speaker. Details: You've already implemented FATFS over SPI and can read/write to the SD card. The task now is to stream audio from the SD card to the speaker using I2S. This will involve managing file reading in chunks and possibly using DMA to maintain continuous playback. You may also need to parse WAV headers if you're using standard audio files.

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.
3. Demo your device.
4. Have you achieved some or all of your Software Requirements Specification (SRS)?
    1. Show how you collected data and the outcomes.
5. Have you achieved some or all of your Hardware Requirements Specification (HRS)?
    1. Show how you collected data and the outcomes.
6. Show off the remaining elements that will make your project whole: mechanical casework, supporting graphical user interface (GUI), web portal, etc.
7. What is the riskiest part remaining of your project?
    1. How do you plan to de-risk this?
8. What questions or help do you need from the teaching team?

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

[Insert final project video here]

- 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 "validation" folder in 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 maximum distance of at least 10 cm.	Confirmed, sensed obstacles up to 15cm. 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.