Review the assignment due date

Music to my Ears

- Team Number: 17
- Team Name: The Redeem Team
- Team Members: Garrett Kirsch, Nikolaos Rapanis, Melina Daniilidis
- GitHub Repository URL: https://github.com/upenn-embedded/final-project-s25-the-redeem-team.git
- GitHub Pages Website URL:

Final Project Proposal

1. Abstract

This project is a music transposer: it takes in a melody played on an electric piano, converts it to a key and octave of your choosing, then plays it back to the user for quick testing.

2. Motivation

Many musicians (*cough* singers) don't know how to read sheet music and haven't learned anything about music theory. This makes it difficult for them to adapt their music to different keys as they don't have the necessary theory background.

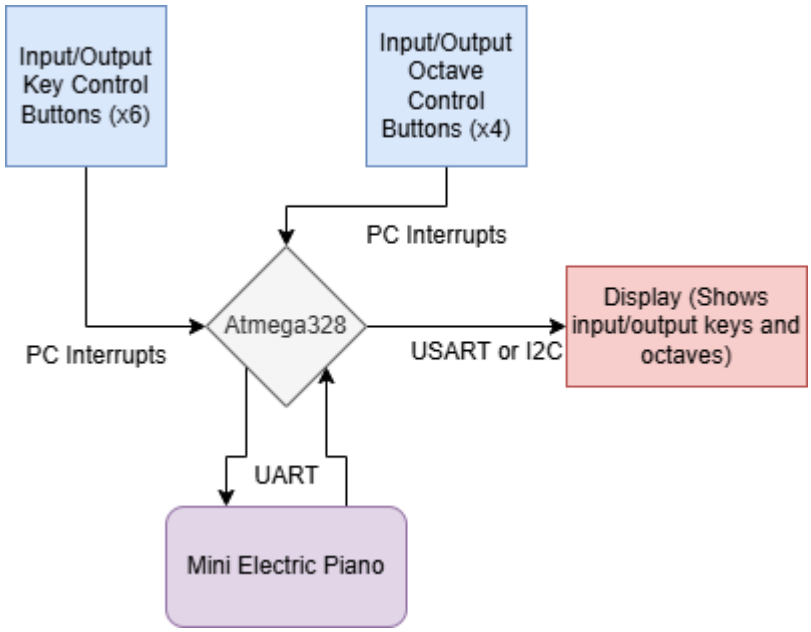
This product will allow them, and learned musicians that couldn't be bothered, to quickly alter their melodies to fit a different key or octave, either for stylistic purposes or because the original music was not in their range.

Key benefits: quick testing of new ideas, easy accessibility to people without music theory backgrounds.

3. System Block Diagram

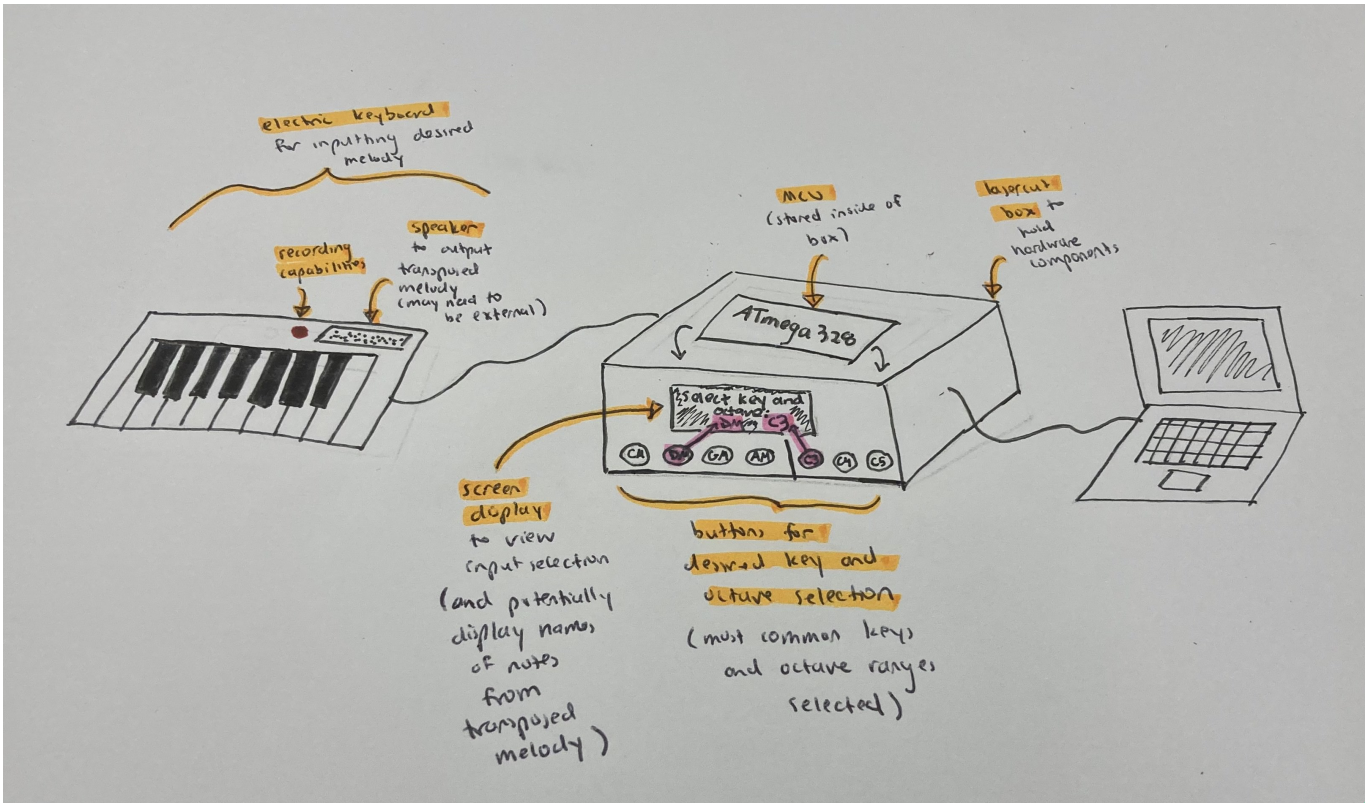
Critical Components: (likely will not need power regulation circuits)

- Electric piano
- Atmega328PB
- Video Display
- Control Buttons (might replace with a WiFi chip and an app if time allows)



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?



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	The user's manual input of the current key (via button press) should be consistent with the notes of the inputted melody, i.e. we check that the notes of the inputted melody match the key signature of the manually inputted initial key. If they don't match, an error message shall be displayed to the user, and perhaps a few compatible keys can be suggested by the system (this would be an advanced feature).
SRS-02	The notes from the input recording shall be cleaned and parsed accurately based on their frequencies. For initial testing, we will assume that one note is placed at a time (no chords) and the melody is not more than seven notes. Once it can parse this simple input successfully, we will incrementally test chords and more complex melodies.
SRS-03	The screen display shall correctly depict the selected key and octave from the button presses.
SRS-04	The user shall not be able to select a desired key transposition that is not compatible with the key of the current melody (e.g. the user cannot select to transposition to a minor key if the inputted melody is in a major key).
SRS-05	The transposition algorithm shall correctly transposition the melody into the user's desired key and octave. The validation for this will involve several stages of testing. First, we will test with a limited selection of keys (namely, the most common ones: C major, D major, G major, A major) and with one standard octave (C4). After we confirm that the algorithm can accurately transposition a few simple melodies with these common keys in C4, then we can gradually add one or two octaves (as many as the keyboard allows) and test. Finally, we can begin adding and testing more complex keys or test more complex input melodies with the common keys.
SRS-06	The speaker shall correctly output the transposition melody. We will test on the same types of inputs as the transposition algorithm and in similar order.
SRS-07	The screen should correctly display the names of the notes from the transposed melody in a readable format, so the user can try to play the transposed melody on the keyboard if they so choose.
SRS-08	The screen should display the transposed notes and the speaker will play the frequencies of the transposed notes simultaneously.

6. Hardware Requirements Specification (HRS)

ID	Description
HRS-01	A mini electric piano will be used to provide inputs (i.e. music) that will then be processed by the atmega
HRS-02	The mini-piano will also be used to output music - both the raw inputs and the transcribed outputs

ID	Description
HRS-03	4 buttons will be used to choose the input key of the music.
HRS-04	4 buttons will be used to choose the output key of the music
HRS-05	A screen will be used to display the input key, output key, input octave, and output octave of the music.
HRS-06	2 buttons will be used to cycle through octaves.

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.

Piano: It is a MIDI keyboard chosen for full-size velocity-sensitive keys, real-time note transmission over 5-pin DIN (not USB), and wide availability.

MIDI Cable: Standard 5 pin M-M cable, compatible with SDS-50J.

6N138: Opto-isolator. Required for signal preservation according to the datasheet. Supports 31,250 baud rate we need for UART.

SDS-50J Connector: Matches the pins from the cable to wires.

AST-03008MR-R Buzzer: Externally driven (PWM) magnetic buzzer with wide frequency range (300–8,000 Hz) to play all the notes.

1N4148 Diode: Required for circuit protection by the datasheet.

1.8" Color TFT LCD display with MicroSD Card Breakout - ST7735R: We already have this screen, so we will use this.

Potential resistors/transistors/breadboards/cables/buttons/potentiometres (we will get them through detkin).

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/1vXRE2RFJ4J_eHgv4eHKd_utGaQBJlPumqM2778kMy1A/edit?gid=253149064#gid=253149064

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 put the mini piano and our other components on the workbench. We will play a melody on the piano while in listening mode, the MCU will read the signals from it, and then we will turn a physical switch on the board to choose the transposition of the song. Then we will press a button for a buzzer to play the song in the new key.

Our system does not need any special environment - just a countertop. We will power the system with a laptop/battery/DC supply.

The constraint is just enough space on the workbench.

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	Input circuit works, we have connection between the piano and the MCU; UART works and we get response; Parsing notes and timings	Nikolaos - Circuitry; Melina: UART enable communication and load the notes; Garrett: MIDI keyboard works + mock input test
Sprint #2	Transposition works and we are able to produce correct new frequencies; Timing works and buzzer (+ circuit) is able to produce notes.	Garrett & Melina - Timing logic; Nikolaos: being able to programme the output to the buzzer, programme switch, testing that everything works
MVP Demo	Everything works together maybe with some integration issues, but reliable input/output	All hands: Team works together to integrate the different parts each one made. We reharse and made sure we have a viable MVP
Final Demo	Final polish, potential UX enhancement (LEDs etc), edge cases, make sure everything works flawlessly	All hands work together because 6 eyes/ears are better than 2 in catching issues

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 ordered our BOM and are currently waiting on part arrivals.
- We made some updates to our firmware to configure the LCD screen to the needs of our project. Specifically, we wrote a function to display notes and another to display a measure in a designated portion of the screen. The notes appear in their proper place (as according to a treble clef).
- Here is the demo of some of our graphics features.
- Researched the MIDI keyboard input protocol and brainstormed the flow of our program, in regards to reading in and sounding the notes. We made a detailed pseudocode outline, so that we have a better

idea of how to plan and modularize our code: read_in_pseudocode.txt
- Implemented transposing algorithm: transpose.c
    - It currently stands independently from the rest of the implementation (just to get the theory down), but we will modify our note representations and function inputs to comply with the rest of our system in our next sprint.

Current state of project

We currently have a detailed flow of our system that takes into account the specific inputs and outputs of the parts that we ordered and the parts we intend to use. This puts us in a good position for our hardware implementation and for our next immediate goal, which is to prototype and test our transposition/scaling algorithm with the buttons and buzzers that are provided by Detkin. Additionally, parts have been ordered and are in shipment.

Next week's plan

- Investigate the MIDI keyboard datasheet and other relevant resources to clarify the serial communication protocol. Make the necessary adjustments in code.

    - Estimated time: 1 week
    - Person(s): Garrett
    - Definition of "done": Working demo of serial communication from the MIDI keyboard being converted into usable data. Piano to note in firmware would be ideal.

- Test inputting the current key and desired key on a set of buttons (Detkin parts). Test the transposition and outputting the melody on Detkin buzzers. The inputted melodies will be hardcoded for now.

    - Estimated time: 1 week
    - Person(s): Nikolas and Melina
    - Definition of "done": Successful and accurate output of a transposition melody on a buzzer. Alternatively, if we run into a lot of debugging, document this, with clear recognition of the problem and the next steps that need to be taken to mitigate the problem.

Sprint Review #2

Last week's progress

- We've received most of our parts and will be able to test the piano communication next Monday.
- We finished the graphics libary for our visual outputs (Notes, measures, sharp/flat) and it is very fast!
- In newmain.c we have the working code to play a hardcoded song in the buzzer.
- In transpose.c we have the transposition code.
- Here is the video of our waveforms created by the code playing ode to joy.
- Here is a video of our updated graphics.

Current state of project

Unfortunately, we have not been able to test the core input to our system - the piano. So far all we've been able to do is work on the peripherals, but we need all our parts to make real progress on the project. We are

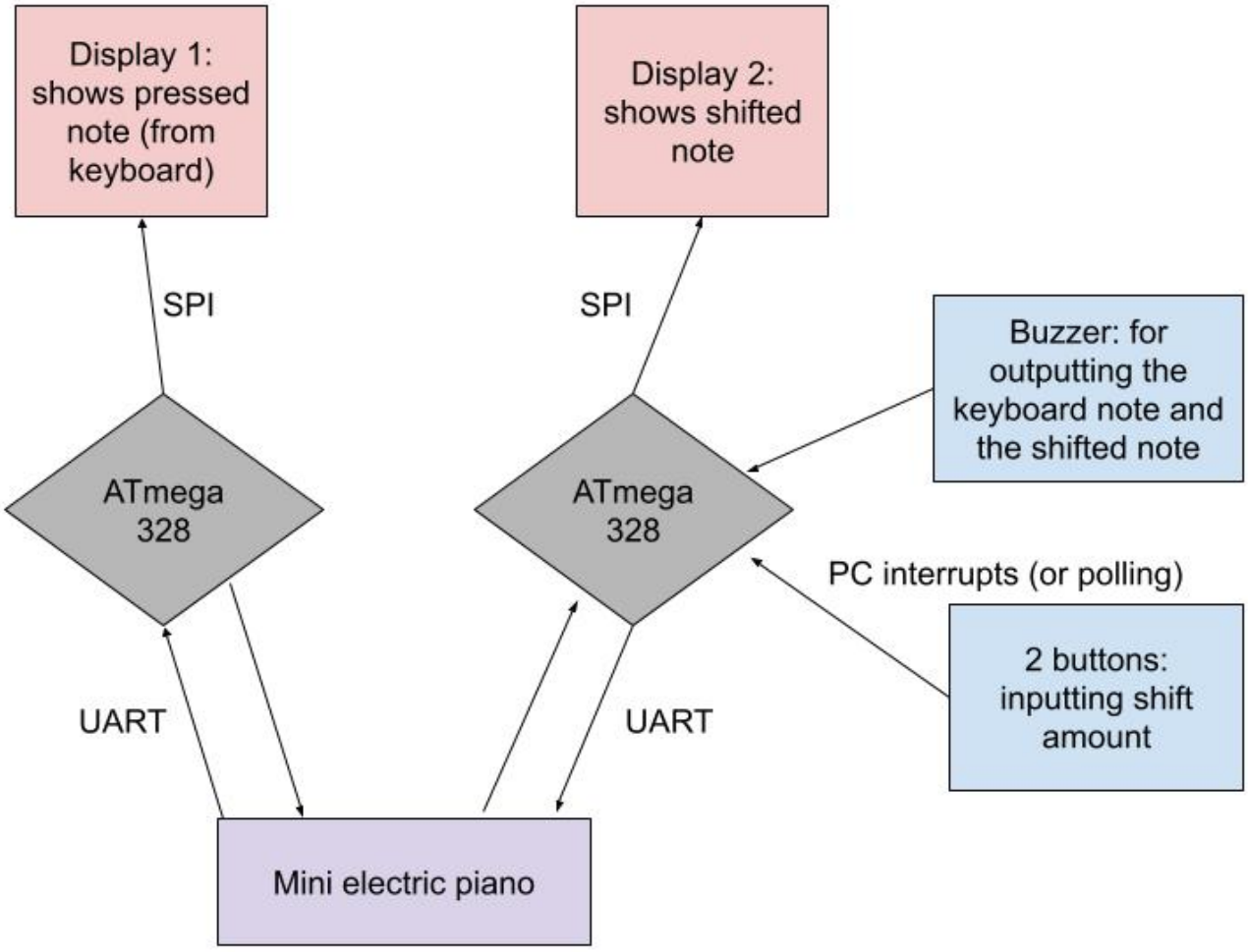
confident, though, that when we get everything, all the work we've been doing will fall in place and we'll have a better picture of what needs to be done.

Next week's plan

- Get the piano input working and convert it into real audio.
- Tranpose and inputed note into another key or shift it in some way that we can show proof of concept.
- Make a small structure to house the product (maybe out of cardboard).
- Output video to two separate screens on two separate ATMegas communicating together with UART.

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.

Our firmware receives MIDI messages over UART and plays notes using PWM on a buzzer. When a note-on message is received, we calculate the frequency, set the PWM registers, and draw the note on the LCD. We also store each note's data in a melody array for future use. The system supports shifting notes up or down using a constant. We wrote the main application logic and critical drivers for the LCD screen and re-used the UART driver from a previous lab.

3. Demo your device.

https://drive.google.com/file/d/1alwgwSCpyfBwg8v-MOm49NPUrAQvz7KO/view?usp=sharing

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

Please note that we have made the following important change to our design: Instead of selecting a desired key to transpose to, we will now have two buttons (up and down) that will allow the user to shift the notes of the inputted melody up or down a certain number of semitones, depending on how many times they press the button of the corresponding direction. This, in turn, has changed some of our software and hardware requirements, which we will address below.

SRS-01: Since we are shifting the melody instead of transposing to a different key, it will no longer be necessary for the user to indicate to the system what the key of the inputted melody is. Therefore, we will not need to check if the user's selection of the inputted melody actually matches the melody played.

SRS-02: We have met this requirement. Specifically, we have managed to read in the MIDI input and map the note number to its corresponding frequency. Then, we send this frequency via PWM to the speaker. So far, we have connected only one speaker and have tested with one note playing at a time. If we have time, we will add a second speaker and test with chords.

We have validated the input reading by printing the MIDI note to the serial terminal whenever it is played:

https://drive.google.com/file/d/1kMERMMm2DjY6He5HBS92R28iUckF7n3f/view?usp=sharing

And we validated that the frequency outputted from the buzzer matches the frequency of the pressed piano key both on the oscilloscope and by placing a tuning app by the speaker:

https://drive.google.com/file/d/1KfSLp451ht9jftIamnTkq8h12IsP6i24/view?usp=sharing

SRS-03: We have modified this requirement this requirement. Instead of depicting the selected key, we can have the screen show the number of times the shift button has been pressed, so the user has a visual indication of how many semitones they want to shift by. We have not met this requirement yet, but it is in our next steps towards the final demo.

SRS-04: Again, since we are shifting the melody instead of selecting a desired key to transpose to, this requirement will no longer be necessary. The shifting will be able to occur regardless.

SRS-05: This requirement also needs a bit of modification. The requirement is now that the transposition algorithm will correctly shift each note of the inputted melody by the user's desired amount of semitones. We have demonstrated that this works in the demo video (in a synchronous manner), as one LCD screen shows the note currently being played on the piano and the other shows the note shifted by a hardcoded (for now) number of shifts.

SRS-06: This is a requirement that we have not yet, but is our first priority for out next steps. We have struggled a bit with marking the duration of each inputted note because our ISRs for the timer that is used to count the duration does not interact well with the MIDI's UART system. We will investigate this further.

SRS-07: We have met this requirement, as evident in the demo video and further explained in SRS-05. The key difference between the MVP and the final demo will be that the transposed notes will be shown on the LCD screen retroactively, so that the user can take note of them and try playing them on the keyboard.

SRS-08: We have met half of this requirement. The screen does display the transposed notes, but we have yet to play the frequencies of the transposition through a speaker (this was a dependency from SRS-05). This is our next step.

Below is our **UPDATED** list of software requirements:

ID	Description
SRS-01	The notes from the input recording shall be cleaned and parsed accurately based on their frequencies. For initial testing, we will assume that one note is placed at a time (no chords) and the melody is not more than seven notes. Once it can parse this simple input successfully, we will incrementally test chords and more complex melodies.
SRS-02	The screen will show the number of times the shift button has been pressed, so the user has a visual indication of how many semitones they want to shift by.
SRS-03	The transposition algorithm will correctly shift each note of the inputted melody by the user's desired amount of semitones.
SRS-04	The screen should correctly display the names of the notes from the transposed melody in a readable format, so the user can try to play the transposed melody on the keyboard if they so choose.
SRS-05	The speaker shall correctly output the transposition melody. We will test on the same types of inputs as the transposition algorithm and in similar order.
SRS-06	The screen should display the transposed notes and the speaker will play the frequencies of the transposed notes simultaneously.

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

HRS-01: We have met this requirement, as shown in the demo video. At first, we struggled with reading in the input because of our opto-isolator, but through the professor's advice, we eventually tried removing the opto-isolator and were clearly able to read the MIDI note, on/off status, and the velocity (volume control) of that note.

HRS-02: The keyboard we ended up ordering did not have a built-in speaker (for budget reasons), so we ordered a separate speaker and configured it to sound the inputted keys of the piano (again, you can see this in the demo video). What is left for us to implement is to sound the transcribed outputs.

HRS-03: This requirement is no longer necessary, as we do not need to have the current key inputted in order to perform a note shift operation.

HRS-04: We have changed this requirement due to our decision to do shifting instead. So, there will be two buttons to choose the shift amount (either to a higher or lower note) by a semitone for each button press.

HRS-05: Currently, we have two screens: one shows the inputted note and the other shows the shifted note. For the final demo, we will have the shifted notes appear on the screen in a sequence after the melody is inputted. We will also add a visual to indicate the number of times the button is pressed, so the user can see by how many semitones they want to shift their melody.

HRS-06: Since we are allowing the user to choose the number of shifts, and given the small size of our keyboard, it will no longer be necessary for the user to select an octave preference.

Below is our **UPDATED** list of hardware requirements:

ID	Description
HRS-01	A mini electric piano will be used to provide inputs (i.e. music) that will then be processed by the atmega
HRS-02	A separate speaker is used to sound the inputted keys of the piano, and the shifted notes.
HRS-03	There will be two buttons to choose the shift amount (either to a higher or lower note) by a semitone for each button press.
HRS-04	A screen will be used to display the input key, output key, input octave, and output octave of the music.

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

In our transition from the MVP to the final deliverable we intend on doing the following:

- Solder all the equipment and hide it in a box of cardboard or MDF.
- Enhance the screen, although it is almost complete, adding the shift on it.
- Given time is sufficient, we will add Wi-Fi capabilities.

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

The most challening part remaining of the project will be to figure out how we can use interrupts that are compatible with the MIDI UART serial communicator. We will need timers and interrupts for two reasons: (1) to determine the duration that each note is played, so we can output the shifted melody with the correct rhythm. (2) configure the buttons that will be used to input the amount of shifting.

For the buttons, we plan to de-risk this by potentially doing polling instead of interrupts to mitigate any conflicts with the MIDI input. For the note duration measurement, we will experiment more the use of timers and ISRs to determine where the conflict lies and what causes it. We will also consult students of past projects who have also interacted with MIDI inputs for their mentorship.

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

We would appreciate some advice or alternative solutions to the challenges presented above (question 7).

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

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.