ECE 1896

Senior Design

SheetSavvy - Monophonic Music Transcription Device Final Design Document

Version 2.0

Cohort 1 Team 3

Prepared By: Emmy Ploskina
Evelyn Pitts
Justin Pacella

Table of Contents

Table of Figures

Table of Tables

0 Document Overview

0.1 Purpose

The purpose of this design document is to document the initial and final design of *SheetSavvy*, a real-time monophonic music transcription device, outlining requirements, design constraints, and high-level technical specifications. This document serves as a critical reference for all technical and non-technical stakeholders involved in this project.

0.2 Document Version Table

Version	**Date**	**Reason for change**
0.0	*9/9/2024*	*Document creation*
0.1	*9/9/2024*	*Added section zero, minor reformatting, and document purpose*
0.2	*9/16/2024*	*Added to Introduction, Background, System Requirements, Schedule, and Minimum Standard. Completed project timeline*
0.3	*9/18/2024*	*Added to Background, Conceptual Design, Design Constraints, References. Completed System Requirements*
0.4	*9/19/2024*	*Added to Design Constraints, Introduction, Team*
0.5	*9/21/2024*	*Finished Introduction*
0.6	*9/23/2024*	*Added to Background, Design constraints, Conceptual Design, Team, Schedule and Budget Plan*
1.0	*9/26/2024*	*Finished Conceptual Design Document*
1.1	*11/18/2024*	*Began editing document for Final Documentation*
2.0	*12/14/2024*	*Review and submission*

Table 1. Document Version Table

1 Introduction

Most sheet music generators on the market are desktop apps that require the user to supply pre-recorded audio files and offer editing abilities due to their low accuracy. Immediately, we address two things that these apps do not: real-time audio processing and confidence in our tested accuracy. Our product, *SheetSavvy*, is a portable, handheld device that can be plugged into any laptop and used to generate sheet music from an audio signal. Users can pull their laptop to one of the approved instruments, choose the desired time signature, and begin playing. SheetSavvy will listen to the music, detect the specific note frequencies, and generate ready-to-print sheet music. The biggest constraints on this project are human error, the sampling rate based on the available memory, and having the time to accomplish the maximum amount of notes.

The three main portions of this design are the hardware, embedded software, and desktop software. The hardware portion involves providing power to the circuit and designing the PCB which takes in an audio signal, filters the signal, and leads to the microcontroller for note classification. The embedded software portion of the design entails designing the pitch and timing detection modules. The pitch detection module involves Fourier transforms and spectral analysis, while the timing detection will focus on time domain analysis. Both of these modules will operate in real-time. The desktop software module is mainly for parsing data from the device and generating the sheet music.

The tests used to assess the functionality of each module started with individual oscilloscope measurements, being able to detect one frequency sent from a waveform generator, and being able to detect timings of notes from a pre recorded audio file. Through integration, we moved to tests that combined modules such as using the breadboard for input to the microcontroller pitch detection and sending data over serial to analyze the desktop software. Finally, we were able to do full system tests such as playing major scales, well-known songs, and repeated notes to grade the pitch and timing accuracy. Overall the testing of our system surpassed our minimum standard but did not reach full functionality. Over the 19 sequential pitches we were able to detect consistently, our data showed an average of approximately 97% for our pitch detection and 92% for our timing detection. We consistently saw that pitch detection is extremely accurate within range and timing detection is very consistent when pitches differ between notes, but it does struggle when harmonics are detected or pitches are consistently the same.

The requirements we set for ourselves were high but doable and we did achieve the majority of them. Our device had all of the user controls we set out to provide and nearly met our pitch and timing requirements of consistent 95% accuracy. The final product was able to perform well during our live demo and produce sheet music. Given more time, I believe we would want to narrow down timing detection as it is the main inconsistency still and also widen our range on the keyboard.

2 Background

2.1 Pitch Detection

SheetSavvy will be able to detect the pitches of notes being played from the instrument through a signal processing frequency-domain algorithm known as Fast Fourier Transform. This will take into account the real and imaginary parts of an input analog signal. After using the Digital Signal Processing library from the chosen microcontroller, the device will save the notes into an array that gets sent over USB to the desktop.

Fast Fourier Transform (FFT) will allow us to extract the fundamental frequency from sinusoid by taking windows of audio at a time and computing them. Similar computations were done by the International Information and Engineering Technology Association (Minor & Kartowisastro, 2022) where they used FFT to transcribe monophonic and polyphonic versions of a few well known songs like Happy Birthday. Specifically, their monophonic transcriptions were extremely successful using this algorithm with very small error rates for pitches and timings under 5%.

2.2 Timing Detection

Detecting note timings is another critical aspect of our product. From researching a similar project from Harvard University's Engineering & Applied Sciences Division (Benetos & Dixon, 2011), it was found that timing detection is divided into two parts: onset/attack detection and release detection. See Figure 1 below to understand the anatomy of a note in a music signal:

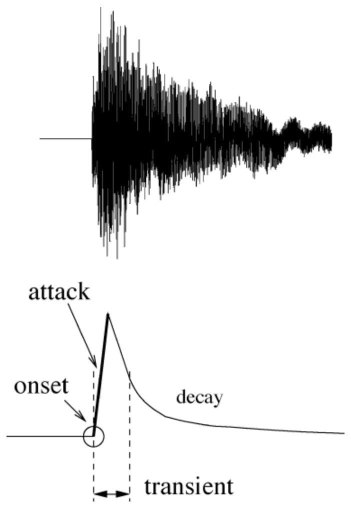


Figure 1. Anatomy of an Ideal Note (Bello & Daudet, 2005)

In the figure, the top graph is the audio signal and the bottom graph is the *reduced* audio signal. The **transient** is a loosely defined duration during which the audio signal changes drastically. A more formal definition may define it as the period of time during which the reduced signal's time derivative exceeds a certain threshold.

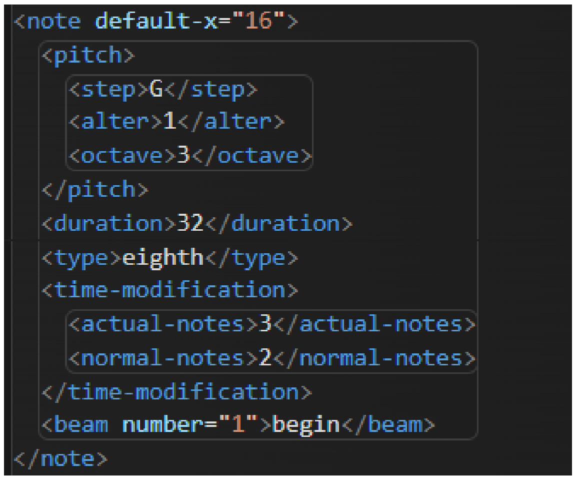
$$\frac{d}{dt}[reductionFunction(signal)](t) > threshold$$

The beginning of the transient is called the **onset** of the note and we defined the **release** as being at the end of the transient. The **decay** is the duration of time after the transient ends. Finally, the **attack** of the note occurs when the reduced signal sharply ramps up. For our device to determine the timing for each note, it should be sufficient to detect the attack and release of the note. On the piano samples we used for testing, we found the duration between the onset and peak to be about 0.015 seconds. For reference, a sixteenth note at 200 bpm lasts about 0.075 seconds. (Bello & Daudet, 2005)

2.3 Sheet Music Generation

Another challenge the team expects to face lies in that sheet music writing is not an exact science. Many aspects of it such as horizontal note spacing and measure sizes may vary depending on a variety of factors. To help solve this and similar issues, our team sought an existing tool to aid in sheet music generation.

One software standard for sheet music notation is the XML-based MusicXML. Michael D. Good published version 1.0 in January 2004 and it has been widely used by online sheet music vendors in years since (Library of Congress, 2024). Notes in MusicXML are denoted using the note tag and are composed of many child tags pertaining to pitch, timing, staff placement, and more. Figure 2. contains a simple example of a note as notated in MusicXML. Specifically, the note represented is a third octave *G#* and is the first note of a half note triplet (MakeMusic, n.d.).



```
<note default-x="16">
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>3</octave>
  </pitch>
  <duration>32</duration>
  <type>eighth</type>
  <time-modification>
    <actual-notes>3</actual-notes>
    <normal-notes>2</normal-notes>
  </time-modification>
  <beam number="1">begin</beam>
</note>
```

Figure 2. MusicXML Note Notation Example

- *Default-X Attribute*: Defines the *x* position relative to the beginning of the measure where the compiler should try to place the note
- Pitch: How high or low the note sounds
  - Step: The letter step of the pitch (*G* in this case)
  - Alter: Half-step modifier; plus one for sharp, minus one for flat
  - Octave: Octave of note starting at *C* with octave one's *C* being at 32.70 Hz
- Rest: Indicates that no pitch is played for the duration of this note. It is written as a self-closing tag and is to be used instead of the pitch tag
- Duration: Numeric duration of the note in defined time divisions. Used to implement notes with different timings than what is written (e.g. swing eighth notes)
- Type: Which note type will be drawn on the page (e.g. whole, quarter)
- Time-Modification: Ratio indicating that the number of notes in a certain region is different from the note type. In this case, the subtags are read as "there are *3* notes played evenly where there would normally be *2*." This indicates a half-note triplet
  - Actual-Notes: Number of notes actually played in the duration
  - Normal-Notes: Number of notes that would normally be played. Also used to define the duration
- Beam: Indicates whether a connector beam should begin, continue, or end. Connector beams connect the stems of eighth notes and smaller notes
  - *Number Attribute*: Describes which beam the tag is describing in the event of multiple beams on the same note.

2.4 Audio Input Denoising

A crucial component of pitch detection for music transcription is providing a clean analog signal for the microcontroller before A/D conversion takes place. The first step to denoising is choosing the proper cable to carry the analog signal from the keyboard to the PCB. XLR cables are one of the most popular cable choices for audio applications due to its three conductors, V+ (hot), V- (cold), and ground (common). The hot and cold conductors carry the same signal and are perfectly out of phase with each other, so any noise that is picked up by the cable will effectively be ‘canceled out’ and leave a cleaner input signal (service bc, 2024).

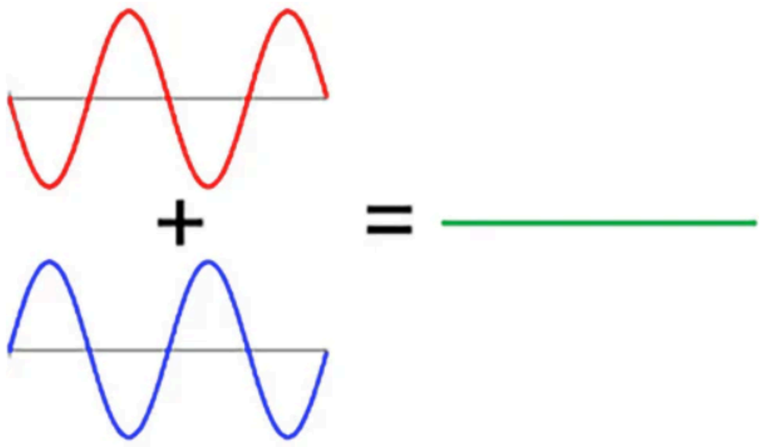


Figure 3. V+ and V- conductors out of phase will cancel resulting noise (service bc, 2024)

Another important component in denoising the input signal is implementing filter circuits that attenuate unwanted frequencies. There are many considerations to make in terms of what filter will best suit the SheetSavvy application, namely whether to use active or passive components. Active filters involve active components, like op-amps and transistors, and they produce a higher quality, ‘cleaner’ analog signal at the cost of more expensive components and the need for external power sources that potentially introduce new noise sources. Active filters also tend to be more complicated, and are best at attenuating low frequencies with low output impedance. (Hotta, 2021)

Passive filters are composed of RLC components and are best at attenuating high frequencies, specifically in the 20k-300GHz range. They offer no amplification and therefore output low gain. Passive filters typically require cheaper components, need no external power source, and tend to have high input and output impedance. They are simpler circuits best at handling high frequencies. For SheetSavvy applications, where most interference would be a result of electrical noise (~7kHz-50MHz), a band pass active filter could be the most effective solution for audio input to the microcontroller. (Marwana, 2024)

3 System Requirements

3.1 Functional Requirements

The requirements detailed in this subsection are specified by the customers/non-technical stakeholders. Functional requirements comprehensively describe all behaviors and functions of the system. We expect that functional requirements will be added, removed, and modified for the duration of the design process.

1. The device must listen to audio in real time using an audio input device.
    a. The audio input device is an audio input jack to plug in an instrument directly.

2. The product is required to detect any **individual** pitch played between F4 (349.228 Hz) and B5 (987.767 Hz).
    a. Desired frequency range from C3 (130.813) to B5 (987.767 Hz)
3. The product must transcribe music played on an electric piano.
4. The product must transcribe music played by any musician.
5. The device must have controls to start/stop recording.
6. The device must implement a metronome with the following requirements:
    a. The device must output sound to indicate tempo.
    b. The device must have controls to start/stop the metronome.
        i. Not necessarily separate from the start/stop recording controls
    c. The device must have controls to adjust the tempo between 60 and 120 beats per minute
7. The device must have controls to select a time signature from 2/4, 3/4, or 4/4.
8. The device must display the following information to the user:
    a. Current beat number
        i. This means the device will display "1" during the first beat of a measure, "3" during the third beat, etc.
    b. Selected time signature
        i. "2/4", "3/4", or "4/4"
    c. Selected tempo in beats per minute
        i. Value displayed must be an integer
    d. Recording status
        i. Indicates whether the device is recording
        ii. Could be shown on a screen or as an LED
    e. Power status
        i. Indicates whether the device powered on
9. The product must generate readable sheet music as a PDF or image *or* as a formatted markup file.
    a. For the latter method, the markup file must be able to be converted to a human-readable format using an existing web application.
10. The PDF generator must handle common note timings including whole, half, quarter, and eighth notes and rests.
11. The PDF generator must write a key signature for each transcription.
12. The device must be able to record the player for at least 8 measures per recording.
    a. This is the average length of one verse in well known songs that you would play when learning music like "Mary Had a Little Lamb" and "Hot Cross Buns".

3.2 Non-Functional Requirements

The requirements detailed in this subsection are specified by the project's technical stakeholders. Non-functional requirements do not describe the functions and behaviors of the system; rather, they describe how well the system performs its functions and behaviors.

1. The product must classify notes' pitches such that measures are correctly transcribed 90% of the time.
    a. In other words, a note's pitch may be wrong no more than 10% in 20 measures.
        i. Ex: 20 measures of whole notes: $18\,notes\,/20\,notes\ =\ 0.90$

        ii. Ex: 20 measures of sixteenth notes: $288\, notes / 320\, notes = 0.90$
    b. The basis of this number is
    c. In the context of system testing, this can be measured simply by recording a user playing a sheet music passage and comparing the resulting sheet music to what the user actually played.
2. The product must classify notes' timings such that measures are correctly transcribed 90% of the time.
    a. In other words, a note's timing may be wrong no more than 10% in 20 measures.
        i. Ex: 20 measures of whole notes: $36\, onsets\ or\ releases / 40\, onsets\ or\ releases = 0.90$
        ii. Ex: 20 measures of sixteenth notes: $576\, onsets\ or\ releases / 640\, onsets\ or\ releases = 0.90$
3. The product's metronome must function with 0% error from the displayed bpm.
    a. This could determined quantitatively by measuring the time between each metronome pulse in software (validation)
    b. This could be determined qualitatively by comparing the metronome (testing)

4 Design Constraints: Standards and Impacts

4.1 Time

The largest challenge our technical stakeholders will face is the project's constrained timeline. Beginning the week of September 9th 2024, our team has until December 2nd 2024 to complete a fully functional prototype of SheetSavvy. This gives us twelve full weeks to conceptualize, design, validate, implement, integrate, and test the SheetSavvy prototype. To accommodate the compressed timeline for this project, our team created a tentative schedule and shall employ the Kanban workflow to aid in visualizing project management. We shall also make budget sacrifices to pay for expedited shipping on essential components so that the team may begin testing quickly. In addition, our team is prepared to, within reason, alter the product's requirements to ensure that the December 2nd deadline is met.

4.2 Budget

While time is our largest and highest prioritized constraint, finances are a major factor the team will consider in design decisions moving forward. The $200 spending limitation has required the team to consider choosing cheaper parts in cases where we can afford to have less strict tolerances or where we can use 'bare bones' devices. For example, we chose to use the dsPIC microcontroller versus a more expensive, 'smarter' Teensy microcontroller.

4.3 Manpower

We are constrained to a team of only three people - one EE and two CoEs. There are many features we considered implementing (for example, the ability to take audio in from multiple instruments or using more efficient/convenient power sources), but manpower constraints required our team to narrow down the system requirements to create three modules that each team member can successfully design, test, and integrate into one solution in the given amount of time.

4.4 Human Error

Human error is a constraint that the team faces because we are relying on users to be consistent with their note timing and pitch accuracy. To minimize note timing errors, a metronome output through headphones (with adjustable bpm for users to choose) was added to the system requirements for the device, making it easier for users to play in time. Another means of minimizing human error is limiting our design to only take audio input from a piano. This will eliminate concerns regarding pitch accuracy from users (for example, if users could use a microphone and sing, there is potential to go sharp or flat and the resulting note on the sheet music would not be accurate or detected at all). However, this design choice limits our market by only attracting those sufficient at playing piano.

4.5 Sampling Rate

One primary focus of this project is estimating the frequencies and timings of notes played during a measure of music. To create a robust system, we need to make sure we are choosing components that have enough capacity to cover a vast range of note frequencies. This is what led us to deciding on using one of the Microchip Digital Signal series microcontrollers. We made sure that it could sample at least 10.0 ksps, and the DSPIC33EV256GM002-I/SP is able to sample up to 1.0 Msps.

4.6 RoHS Standards

While this product is currently being prototyped for academic purposes, our team has decided to design under the assumption that our product will go on the market. Because of this, we are using and ordering components that are Restriction of Hazardous Substances (RoHS) compliant. This will limit the number of options we have when choosing parts, but limited component options is a constraint we are accepting in the name of creating a safe device.

4.7 Impacts in Non-Technical Contexts

1. **Environmental** - To manufacture any electronics, pollution, resource depletion, and energy consumption are key environmental impacts that must be considered. While the components of this device use a relatively small amount of power, there is still energy and resources to be consumed to manufacture the device.
2. **Cultural and Societal** - This device provides an opportunity to enhance music education by providing an easy way for inexperienced musicians to see their pieces transcribed in real time.
3. **Diversity, Equity, and Inclusion** - This device has the potential to promote access to music from people of all cultures and backgrounds, and could be especially beneficial in preserving oral traditions/folklore songs from cultures all around the world.
4. **Economic** - A device like this could be on the market for $20, a one-time purchase that could service hundreds of musicians. Premium music transcription services (with flaws in accuracy) are currently on the market for $50-$250/year. This device will provide an affordable transcription tool for music educators and students.

5 Summary of Design

This section thoroughly details our team's tentative design plans for the full system, each subsystem, and the alternative design concepts for each subsystem. The content presented in this section is subject to significant revisions and updates as the team progresses with the project.

5.1 System Design

5.1.1 System Architecture

Our team chose to divide the system into distinct submodules to improve project organization and readability, making it easier for team members to understand and modify specific parts of the system. Each submodule may include software, hardware, or both. The responsibilities of each submodule shown in the diagram are listed below.

1. Audio Preprocessor: Responsible for audio adapter interfacing, frequency filtration, analog-to-digital conversion, and stereo-to-mono conversion if necessary. This module will be implemented partially in software and hardware.
2. Metronome Signal Generator: Generates a metronome signal based on user input. It must output the metronome signal as an audio output and as a digital signal. This must also display the selected tempo and time signature to the user.
3. Timing Detection Module (TDM): Receives a filtered, digital audio input and will be responsible for determining when each note begins and ends. When it detects the onset of a note, the TDM must also signal the pitch detection module to classify the note at the correct time.
4. Pitch Detection Module (PDM): Receives the same audio input as the TDM and must transform it to the frequency domain and perform spectral analysis to determine the dominant frequency. It must then use this frequency to estimate which pitch is being played.
5. Serial Interface: Collects the pitches and timings from their respective detection modules until the stop recording button is pressed. At this point, the serial interface will systematically send pitches, timings, and other attributes through the serial port.
6. Post Processing Module: Responsible for parsing the data sent over serial and compiling it to a format that may be used by the sheet music generator. This module is also in charge of opening and maintaining the serial connection for the duration of the transfer.
7. Sheet Music Generator: Generates sheet music given, pitches, onset/release timings, tempo, and time signature. The module will store the resulting sheet music as a MusicXML file.

The system architecture diagram is shown in Figure 4. The system I/O dictionary is directly beneath the diagram in Table 4. The bold numbers in the leftmost column of the table correspond to the numbered data routes in the diagram.

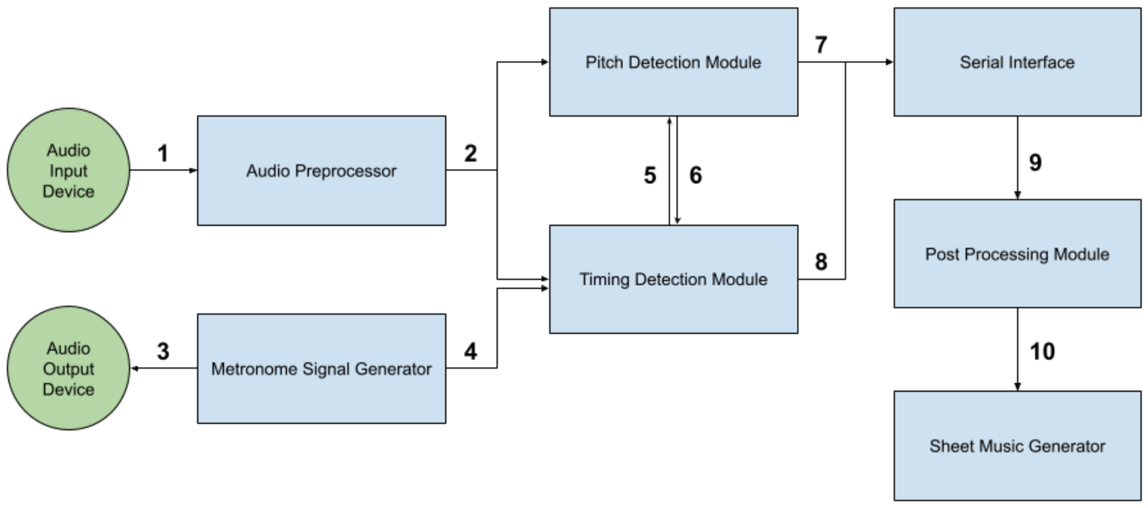


Figure 4. System Architecture Diagram

Signal	**Sender**	**Receiver**	**Description**
1 Raw Stereo Audio	Audio Input Device	Audio Preprocessor	Raw audio input as analog stereo signal
2 Filtered Mono Audio	Audio Preprocessor	Pitch Detection Module	Audio signal after frequency pass filtration and conversion from stereo to mono
		Timing Detection Module	
3 Metronome Audio	Metronome Signal Generator	Audio Output Device	Metronome signal as an audio signal (e.g. “ding” or “click”)
4 Metronome Digital	Metronome Signal Generator	Timing Detection Module	Metronome signal as digital pulses (e.g. square wave)
4 Time Signature	Metronome Signal Generator	Timing Detection Module	Currently selected time signature as enumeration
5 Note Onset Signal	Timing Detection Module	Pitch Detection Module	Sent when TDM detects an onset; tells the PDM to record the current pitch
7 Note Pitches	Pitch Detection Module	Serial Interface	Individual note pitches as unsigned 8-bit integers
8 Onset Timings	Timing Detection Module	Serial Interface	Individual onset timings as 32-bit floating points
8 Release Timings	Timing Detection Module	Serial Interface	Individual release timings as 32-bit floating points

8 Timing Attributes	Timing Detection Module	Serial Interface	Time signature and tempo as encoded bytes
9 Packaged Pitches and Timings	Serial Interface	Post Processing Module	Array of packaged timings and pitches for each note (sent after recording complete
9 Device Status	Serial Interface	Post Processing Module	Either "idle", "recording", "transmitting", or "error"
10 Formatted Pitches	Post Processing Module	Sheet Music Generator	Pitches stored as Step, Alter, and Octave values
10 Formatted Timings	Post Processing Module	Sheet Music Generator	Timings stored as note types (e.g. "eight", "half")
10 Attributes	Post Processing Module	Sheet Music Generator	Other Attributes (i.e. time signature, tempo, etc.)

Table 2. System I/O Dictionary

5.1.2 System Deployment

The system deployment diagram for the SheetSavvy system is shown in Figure 5. The audio preprocessor, pitch and timing detection modules, and metronome will be deployed on our selected microcontroller (DSPIC33). The post processing module and sheet music generator will be deployed in an application on the user's desktop computer.

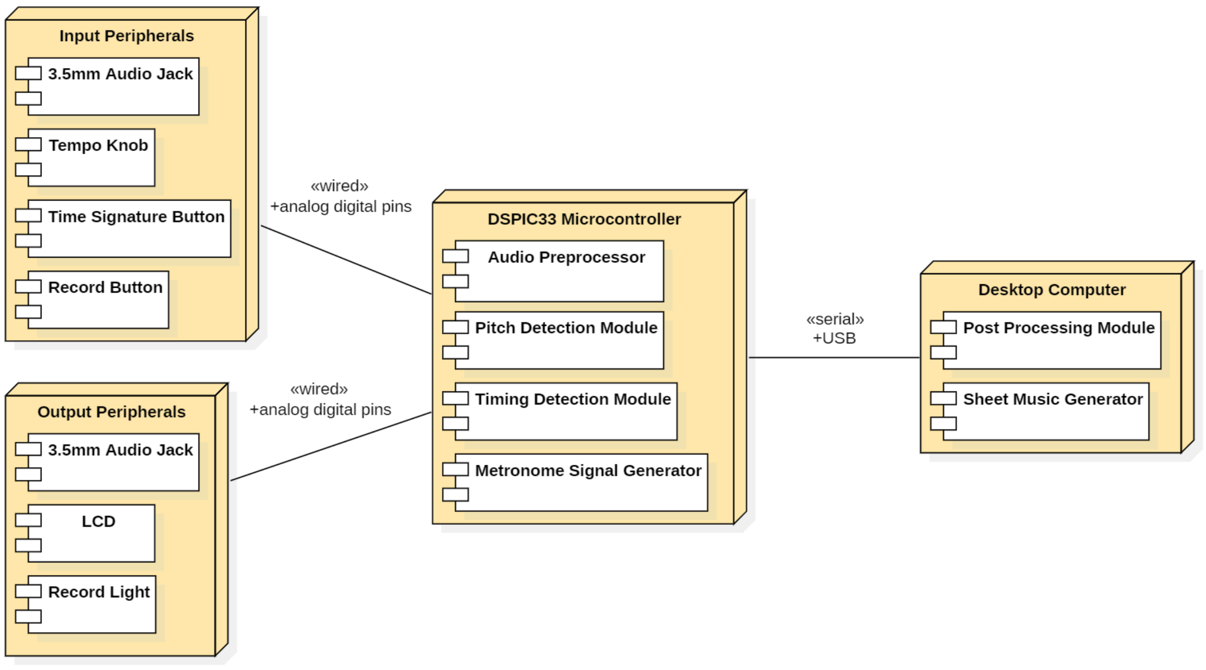


Figure 5. System Deployment Diagram

5.1.3 Users and Use Cases

The users of the SheetSavvy product are pianists who are above the beginner skill level. We assume that our users are capable of keeping tempo, understanding articulation, and interpreting sheet music. The system use case diagram is shown in Figure 6. The subsections below include detailed information about the system's main use cases.

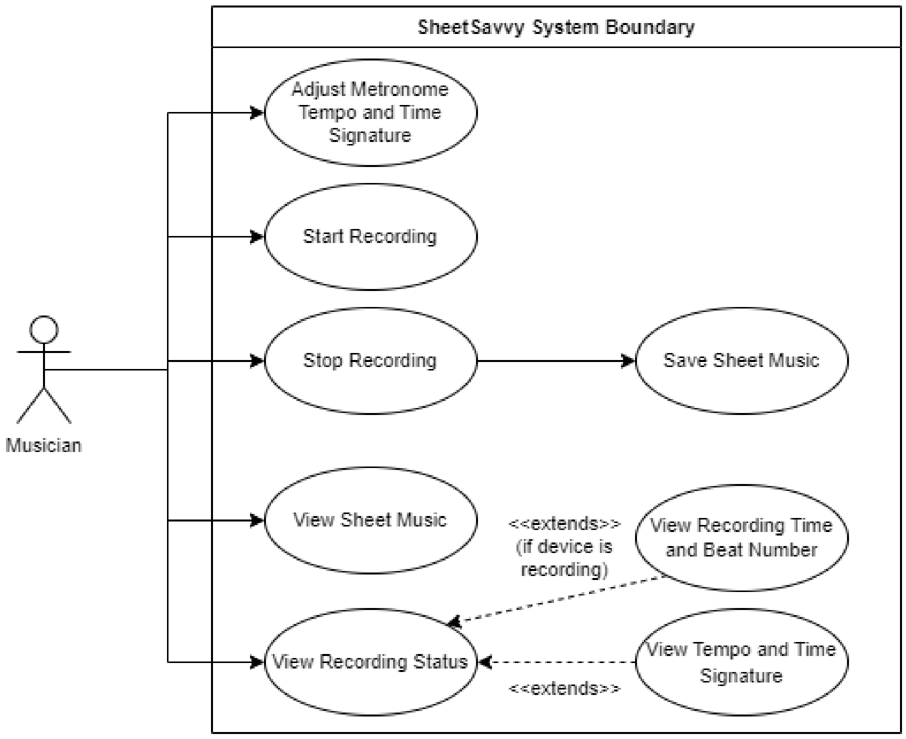

SheetSavvy System Boundary
Adjust Metronome Tempo and Time Signature
Start Recording
Stop Recording
Save Sheet Music
Musician
View Sheet Music
<<extends>> (if device is recording)
View Recording Time and Beat Number
View Recording Status
View Tempo and Time Signature
<<extends>>

Figure 6. System Use Case Diagram

5.1.3.1 System Use Case: User Starts Recording

In this use case, the musician is prepared to begin recording. When they press the record button, the device starts the metronome and gives a one measure countoff after which the device begins recording. The sequence diagram for this use case is in Figure 7.

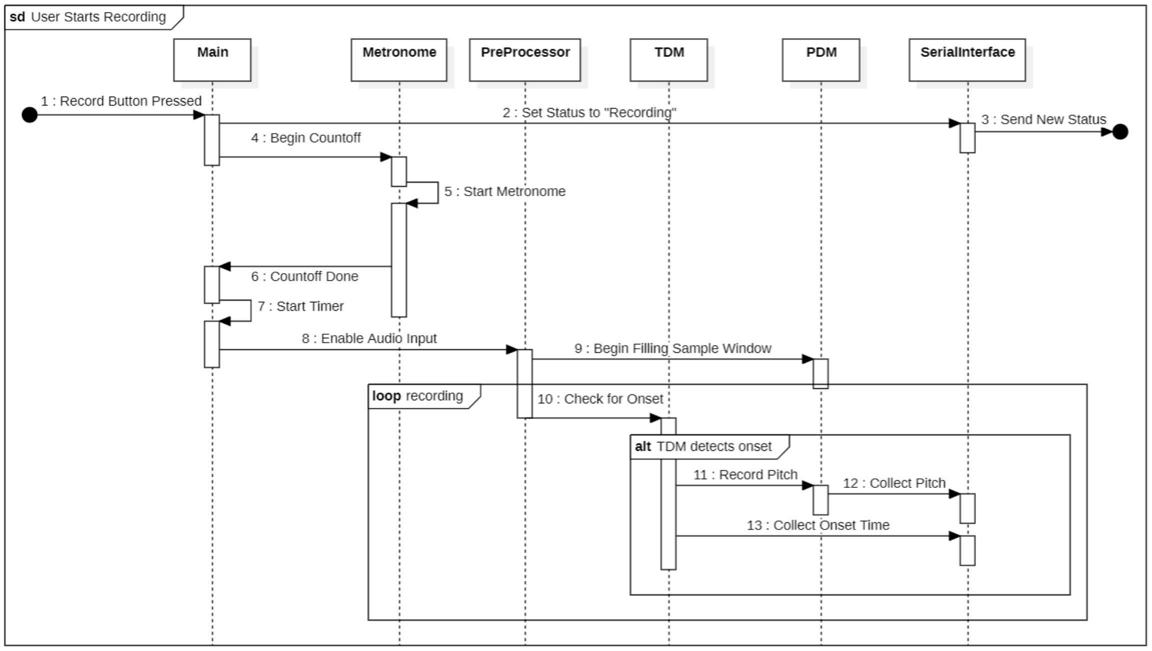


Figure 7. User Starts Recording Sequence Diagram

5.1.3.2 System Use Case: User Ends Recording

In this case, the user has just finished playing and is ready to end the recording. When they press the record button, the metronome will stop and the data transfer will begin. The serial interface will alert the desktop that it is ready to send and wait for the desktop application to respond with its own ready signal. After this, the data transfer will commence.

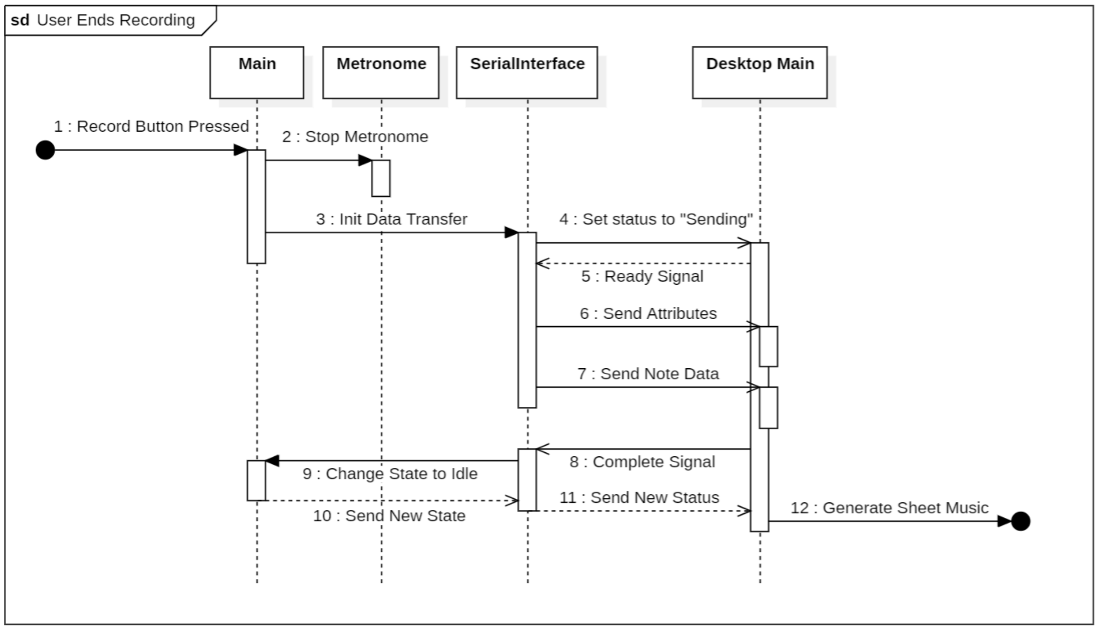


Figure 8. User Ends Recording Sequence Diagram

5.1.4 General Design Choices

The hardware design concept includes six main submodules: power and data transfer, audio input, input signal filtering, microcontroller for A/D conversion and FFT, and user inputs. Hardware will be implemented via breadboard testing with a resulting PCB to house each component. The hardware design concept requires +5VDC and data transfer through USB, a PC application, and a keyboard with aux input to the device. The hardware design also includes a power switch and LED to display to users the device is powered on, a start/stop recording button with an LED to show when recording is in progress and to start/stop the metronome, and three buttons to select the time signature (2/4, 3/4, or 4/4). The diagram depicting the general hardware structure is in Figure 9.

The embedded design includes a microcontroller which implements the pitch detection and timing detecting using digital signal processing libraries and timers. It will use UART serial communication to speak to the desktop module over USB. The desktop module will showcase the pitches and timings it has been sent in sheet music which it will produce. This will occur on a graphical user interface which the musician will be able to interact with.

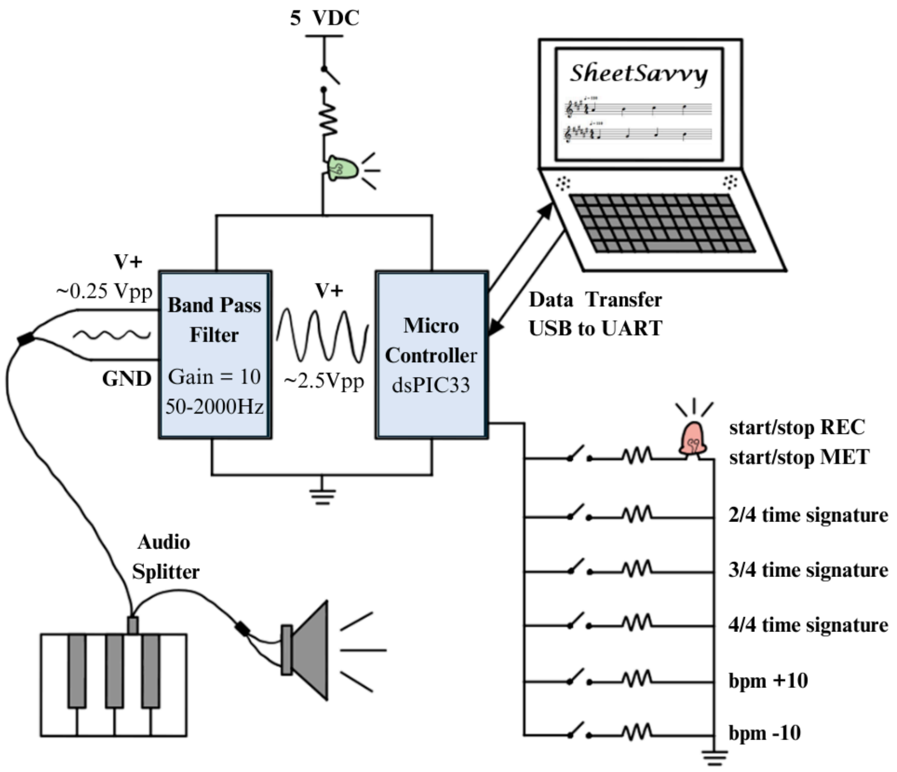


Figure 9. General Hardware Diagram

The device must sample audio at a rate no less than 10 kHz, calculated from the Shannon-Nyquist sampling theorem which says an analog signal needs to be sampled approximately 2.3 times to be represented in digital space. Because the highest input frequency that will be detected is 4186 Hz (C8 on a standard 88 key piano), the microcontroller sampling rate must be greater than 10 kHz.

5.2 Subsystem Design Concepts

5.2.1 Hardware Module Concept

From the keyboard, a 3.5mm aux cable feeds the audio signal to the hardware filtering submodule, pictured in Figure 10. The audio input will go through an active band pass filter circuit to eliminate electrical noise. Due to the peak-to-peak voltage from the keyboard reaching a maximum of ~0.25Vpp, the active filter was chosen to manipulate the signal's gain, which is 10 in this configuration, and consistently outputs notes at ~2.5Vpp. The filter was designed to attenuate frequencies outside the 50-2000 Hz range. This range was chosen because the desired range for pitch detection was between notes C2 and C6 on the piano, whose frequencies range from 65-1024 Hz. There is also hardware in this filter to implement a DC offset of ~2V to shift the signal above the 0V axis, a necessary step that allows the microcontroller to receive the full signal that is sent, as the microcontroller cannot read negative voltages.

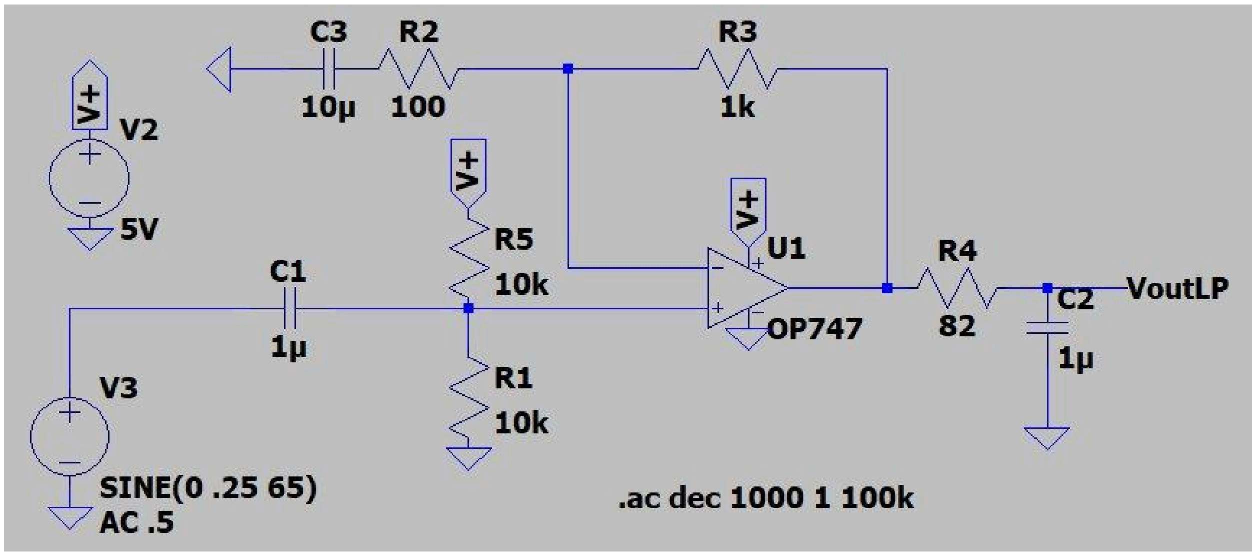


Figure 10. Active Band Pass Filter

From the filter, the audio signal inputs to the microcontroller to perform analog to digital conversions and FFT/time domain algorithms to classify pitches and note timings. The microcontroller outputs an LED associated with a button to start and stop recording and start and stop the metronome. The metronome is implemented in embedded and outputs to the user's PC, and has associated buttons to increase and decrease the bpm of the metronome by 10 bpm. Users are also able to choose a time signature (2/4, 3/4, 4/4) with three buttons associated with the three options.

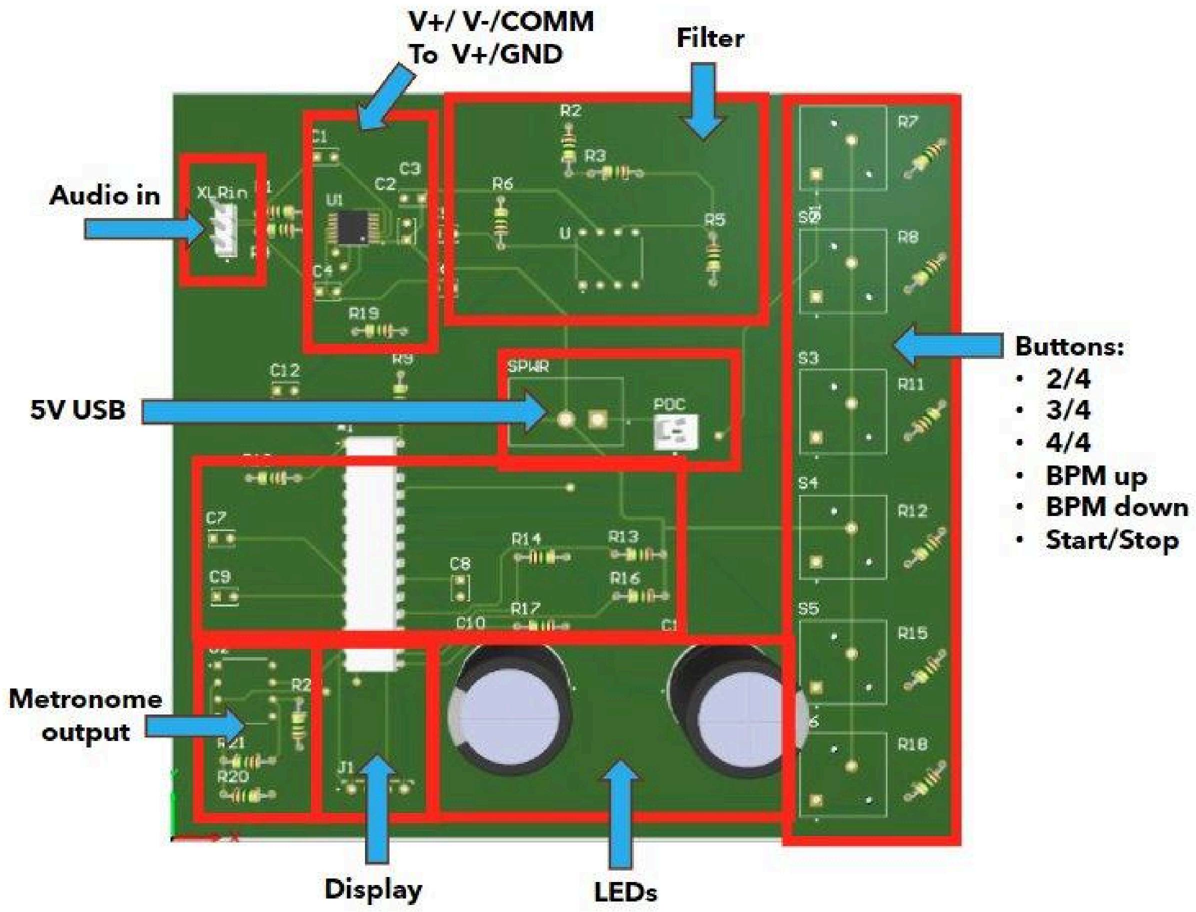


Figure 11. Assembled PCB

5.2.3 Embedded Software Module

The module for the embedded software includes using a DSPic, from Microchip, as the microcontroller, and the Digital Signal Processing (DSP) library included with the DSPic compiler. The microcontroller will take an analog input to which it will perform the Fast Fourier Transform and replace the input with its computed output. Then, it will be able to use vector functions to obtain the dominant frequency, match that to its corresponding note, and report the notes over USB data lines. (Fergus O'Kane Microchip Technology Inc, 2015) The general process is shown in the diagrams below.

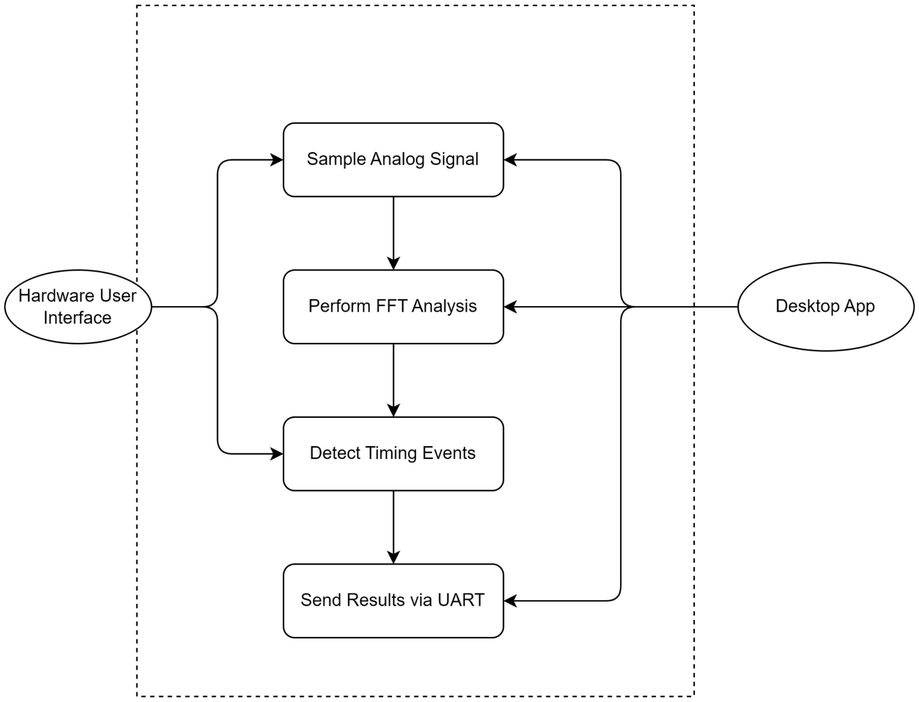


Figure 12. Embedded Module Use Case Diagram

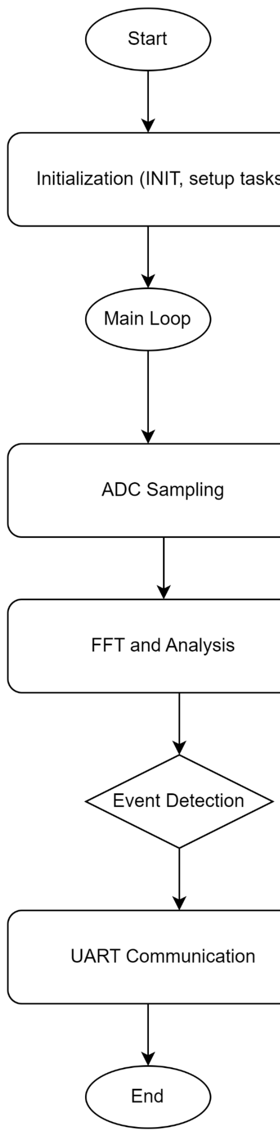


Figure 13. Embedded Procedure Flowchart

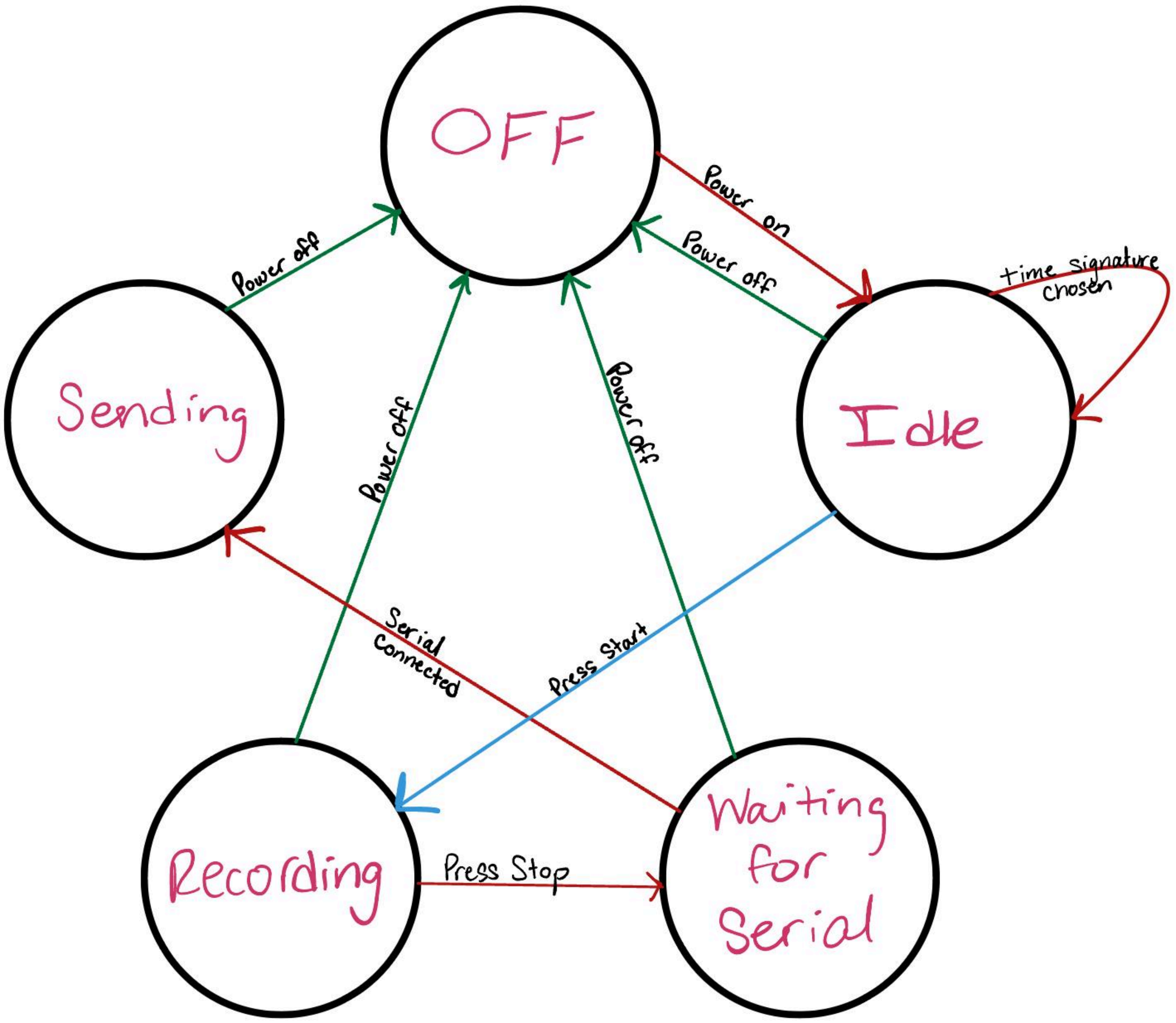


Figure 14. Embedded System State Flow Diagram

5.2.5 Desktop Software Module

As per our overall system design, the desktop software module must communicate with the embedded module and generate the sheet music when the user ends the recording. The preferred design concept for the desktop software module involves generating the sheet music as a formatted markup file such as MusicXML. MusicXML is a subset of XML that specifically stores sheet music in an easily modifiable, unencrypted format (Good, 2009).

The plaintext format of the file is one of the advantages of this design concept; in this method, the sheet music could be easily generated and modified using builtin libraries (XML parsers) for most modern software languages. Generally, using these libraries instead of external libraries enhances reliability due to thorough testing and community support. A disadvantage this concept introduces is reduced portability. While a MusicXML file can be easily converted to a PDF using an online converter such as SoundSlice (SoundSlice, n.d.), this method requires that the user has an internet connection which may not always be present. One potential workaround for this issue may be to download and localize a MusicXML-to-PDF compiler so that it may be used offline. The MuseScore Studio desktop application can be used for this purpose.

At this time, the desktop software module has only a few use cases; the main one being to generate sheet music from the note data sent from the microcontroller. The desktop software module use case diagram is shown in Figure 15.

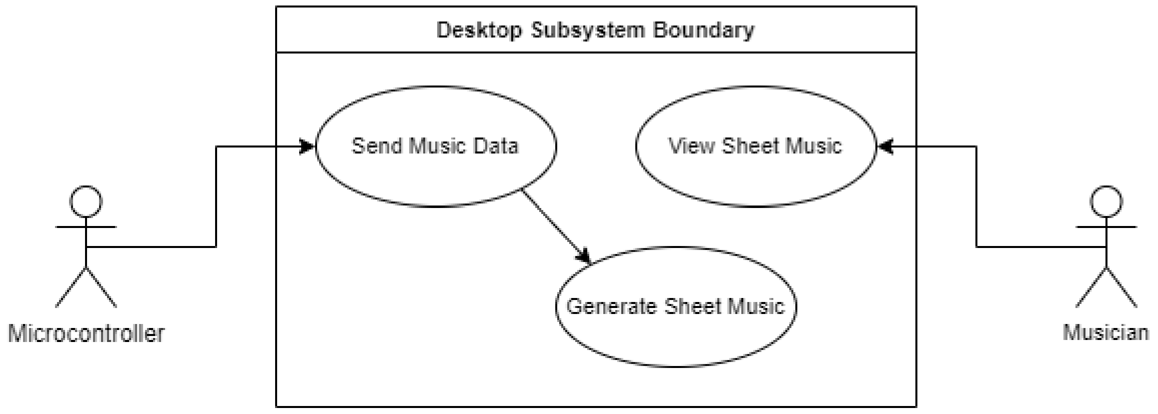


Figure 15. Desktop Software Use Case Diagram

The class structure of the desktop software is shown in Figure 16. The SerialInterface, PostProcessor, and SheetMusicGenerator classes form the main backend for the desktop software module. The Score, Measure, and Note objects are a tree of XML-inherited classes included to make the sheet music generation process more readable and analyzable.

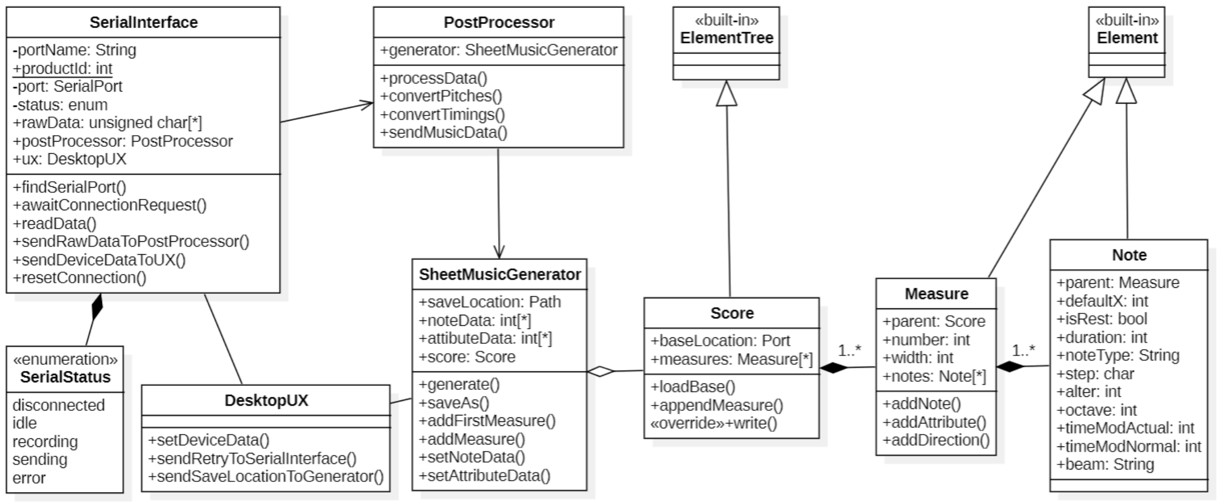


Figure 16. Desktop Software Class Diagram

We decided to keep the user interface for the desktop application as simple as possible due to the very simple set of use cases. The UI shows the user information about the connected device, allows them to retry the connection, and allows them to save the sheet music file when it becomes available. The UI prototype for when a device is connected and disconnected are shown in figures 17 (connected) and 18 (disconnected). The final version of the desktop software is in figure 19.

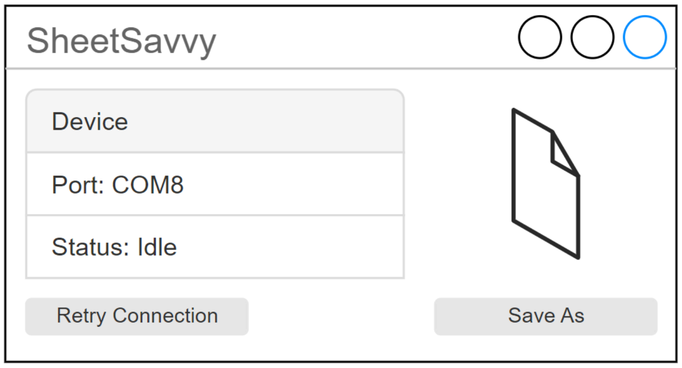


Figure 17. Desktop Software UX Prototype

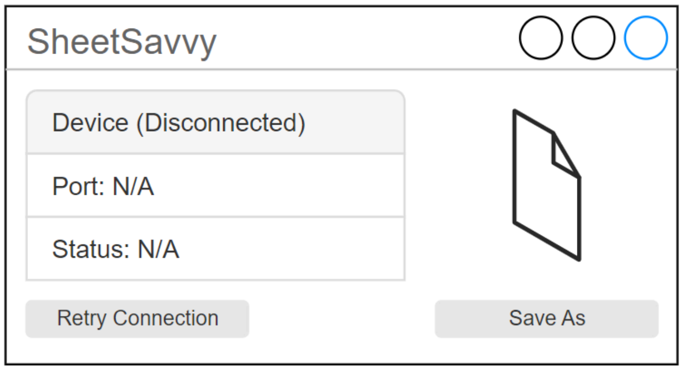
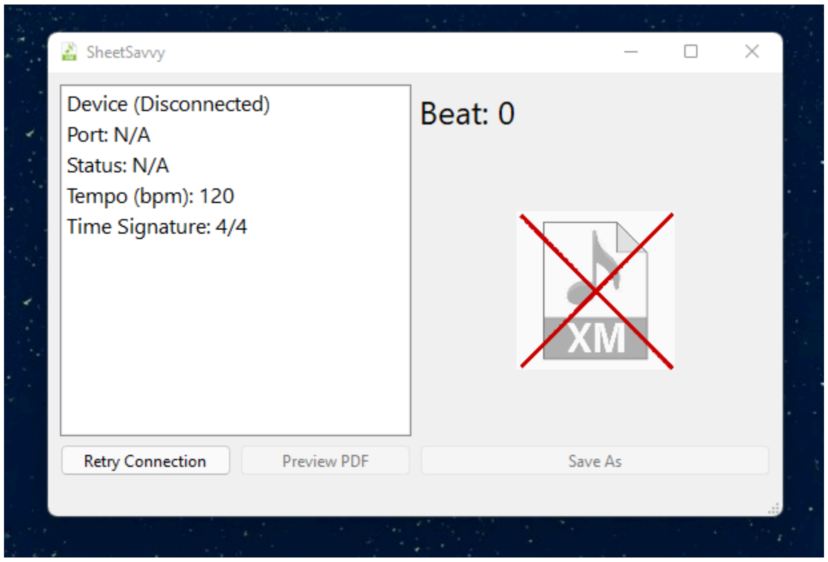


Figure 18. Desktop Software UX Prototype (Disconnected)



Device (Disconnected)
Port: N/A
Status: N/A
Tempo (bpm): 120
Time Signature: 4/4

Beat: 0

Figure 19. Desktop Software Implementation

As mentioned earlier, the desktop software module's main use case is generating sheet music. Specifically, when the user ends the recording, the SheetSavvy device will send the music attributes and note data over serial to the desktop. At this point, the desktop module will begin generating the sheet music. The detailed sequence diagram for this use case can be seen in Figure 20.

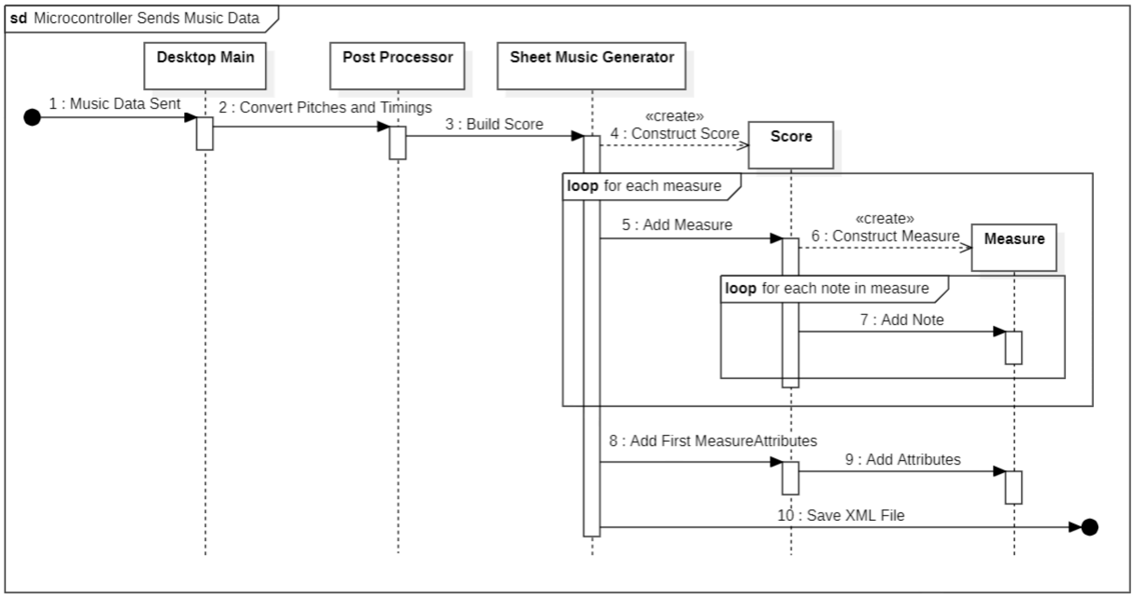


Figure 20. Microcontroller Sends Music Data Sequence Diagram

5.3 Selected Design Concept

The selected design concept will be a combination of Hardware Concept A, Embedded Software Concept A, and Desktop Software Concept A.

Because of budget constraints and the opportunity for an active filter to introduce new noise, hardware concept A with the passive low pass filter circuit seems to be the best option for the SheetSavvy device's operation. In acknowledgement of the compressed timeline for this project, we chose desktop software concept A since it is less time consuming to implement.

Embedded Concept A plays to the strengths of our embedded software engineer and her previous IDE experience. It is also a cost efficient solution with readily available resources for troubleshooting.

6 Team and Timeline

6.1 Emmy Ploskina - Hardware Development Engineer

Emmy is the electrical engineer on the team, whose main functions are PCB design and audio filtering before the input signal reaches the microcontroller for note classification.

6.1.1 Skills learned in ECE coursework

The skills required to design and test the hardware/PCB module were primarily acquired in ECE 1212: Circuit Design Lab and ECE 1895: Junior Design. In ECE 1212, students built various filter circuits using hardware discussed in ECE 0102: Microelectronic Device Theory. In

ECE 1895, students were taught skills that are necessary for the success of our project, including PCB designing in Altium, good soldering practices, time management, and 3D enclosure design for product marketability. All of the skills learned in junior design will be applied in this project, and much of the theory taught in ECE 0102 and their applications taught in ECE 1212 will be applied throughout the duration of the project.

6.1.2 Skills learned outside ECE coursework

Outside of the ECE coursework, music theory plays a role in both formatting sheet music and for functionality requirements such as classifying notes and time signatures. I took voice lessons for ten years and piano lessons for eight, where much of my time was spent reading and analyzing sheet music. Other skills obtained outside of class come from work experience as a technical sales/marketing intern - team communication and collaboration, project management and budgeting, and 'big picture' items like designing for the user and considering marketability for the senior design EXPO. All of these skills will play large roles in the success of this project.

I have experience designing filters, however there is much research to be done in terms of what configuration will best reduce noise from the input signal (especially at the EXPO, where the rooms will be loud and echoey, and noise will be random), and this will be a major portion of my research and testing in the first few weeks of design. Another item that will be independently researched is the implementation of a metronome that outputs to users so that users stay in time with whatever time signature/bpm they choose, while not interfering with the input signal.

6.2 Evelyn Pitts - Embedded Software Engineer

Evelyn is one of the Computer Engineers on the team, and she will be focusing on the embedded programming of the microcontroller pitch detection algorithm and peripherals (buttons, LEDs, etc…). She will act as the main link between Emmy and Justin in the full integration with her main focus on note distinction.

6.2.1 Skills learned in ECE coursework

The main classes that I will be applying towards this project are ones I took recently such as ECE 1212 : Electronic Circuit Design Lab and ECE 1188 : Cyber Physical Systems. In ECE 1212, I was able to expand my knowledge on circuitry like the bandpass filters and analog-to-digital converters that we will be utilizing on this project. This makes interfacing and testing in conjunction with Emmy much easier as we both know how it should be working. In ECE 1188, I had more than enough experience programming a microcontroller and using interrupts, hardware timing systems, and other tools that will be functional for the signal processing aspect of this project. These classes did a great job of letting me improve on the basics that I learned during ECE 102 and ECE 202.

6.2.2 Skills learned outside ECE coursework

A large part of my skill set was learned during my co-op rotations at Powercast. I worked on many different types of filtering circuits focusing on specific frequencies as well as learning how to use an array of equipment to my advantage during testing. This bolstered my ability to come up with creative solutions that will help greatly during the trial phases of this project. I'm also currently working on programming a microcontroller at Powercast and utilizing communication protocols like SPI and UART. Finally, during the semester while we work on this project, I plan to learn more about signal processing, especially Fast Fourier Transform and equivalent signal processing algorithms.

6.3 Justin Pacella - Desktop Software Engineer

Justin is one of the team's Computer Engineers focusing on the software endpoint portion of the design. This entails generating the sheet music from the detected notes and timings outputted by Evelyn's module.

6.3.1 Skills learned in ECE coursework

To complete my portion of the project, I expect to apply a volume of ECE coursework skills. For the design as a whole, I expect to apply skills learned from ECE 1140 (Systems and Project Engineering) and ECE 1895 (Junior Design Fundamentals). In ECE 1140, I gained hands-on skills in full-stack development, architectural/design patterns, and software/hardware integration. ECE 1895 taught me how to efficiently structure an engineering project including research, design, validation, implementation, and testing phases. I expect that I will apply many fundamental skills learned from ECE 0302 (Data Structures and Algorithms) and ECE 1148 (Algorithmic Thinking). Reading, parsing, and generating markup language files for PDF creation will involve the use of several non-trivial algorithms learned from these classes.

6.3.2 Skills learned outside ECE coursework

The part of my skill set I learned outside of coursework mainly consists of programming fundamentals and deep learning research. I have dedicated a lot of time to solving online programming problems and honing skills that I will apply in the context of this project. I don't imagine the technical skills I learned from my deep learning research will be applicable to this project. During that time, however, I improved other skills such as time management, engineering research, and technical writing. For the duration of this project, I intend to research and learn more about software/hardware interfacing schemes, applied digital signal processing, and XML parsing.

6.4 Project Schedule

For our final project schedule we are going to organize the information by the large milestones throughout the class. This will be based on what we were actually able to accomplish.

Checkoff #1	**Emmy**	Low pass filter testing: LtSpice simulations and breadboard tests using waveform generator, oscilloscope, and DC power supply
	Evelyn	UART communication in progress, Peripheral LEDs tested, ADC Sampling started
	Justin	Desktop software minimum functionality minus serial interface implementation (unit tests plus software test stub)

Midterm Presentations	**Emmy**	Low pass filter and balanced to single ended converter testing: LtSpice simulations and breadboard tests using audio from keyboard, oscilloscope, and DC power supply
	Evelyn	ADC Sampling and UART Communication established, FFT computing in progress
	Justin	Timing detection algorithm results & desktop software serial interface (hardware test stub)
Checkoff #2	**Emmy**	Band pass filter testing: New filter design with more narrow passing range, gain of 10, and ~2 VDC offset for the microcontroller. LtSpice simulations and breadboard tests using audio from keyboard, oscilloscope, and 5VDC from USB. PCB Design: designed and ordered, with some known but fixable errors Integration: able to detect 1 whole octave. Audio input and filtering fully integrated with embedded to deliver a clean, consistent signal to the microcontroller.
	Evelyn	Able to detect 1 whole octave, Adding in overlap buffer/FFT windowing to increase pitch and timing detection, integrating with hardware, began integrating peripherals and desktop software
	Justin	Demonstrate full note detection functionality. Final sampling rate and note detection pipeline established
Final Presentations	**Emmy**	Full module functionality with the same hardware design from checkoff 2 and components soldered on the PCB. System fully integrated with embedded and desktop modules, and live demo prepared.
	Evelyn	All sections fully integrated, Pitch detection working above minimum functionality, Live demonstration prepared for class presentation
	Justin	Enhanced desktop software appearance and added beat number, tempo, and time signature to user interface.

Table 3. Project Timeline

7 Testing, Data Analysis, and Results

7.1 Hardware System Testing

The design of the ordered PCB was altered prior to receiving it, so assembling the PCB was taken one step at a time to ensure the hardware module would be fully functional. First the filter was assembled to make sure the signal would pass properly to the microcontroller. The filter's frequency response on the PCB, powered by 5VDC through a power switch, was the first test to be completed. The max peak-to-peak voltage of the signal through the filter was 2.64V, and the cutoff voltage for the signal to be read was half of that, at 1.32V. Figure 21 shows the average output voltage and peak frequency detected over 5 trials per pitch, from 10 Hz to 50 kHz. The desired band pass frequency range is between 50 and 2000 Hz, a range chosen to satisfy the original goal of detecting pitches between 65 and 1046 Hz, notes C2 to C6 on the keyboard. Among the five trials, the pitch detected was extremely consistent, reading the same pitch in each trial down to 2 decimal points. As anticipated, frequencies below 50 Hz and above 2000 Hz were below the 3dB cutoff (1.32V).

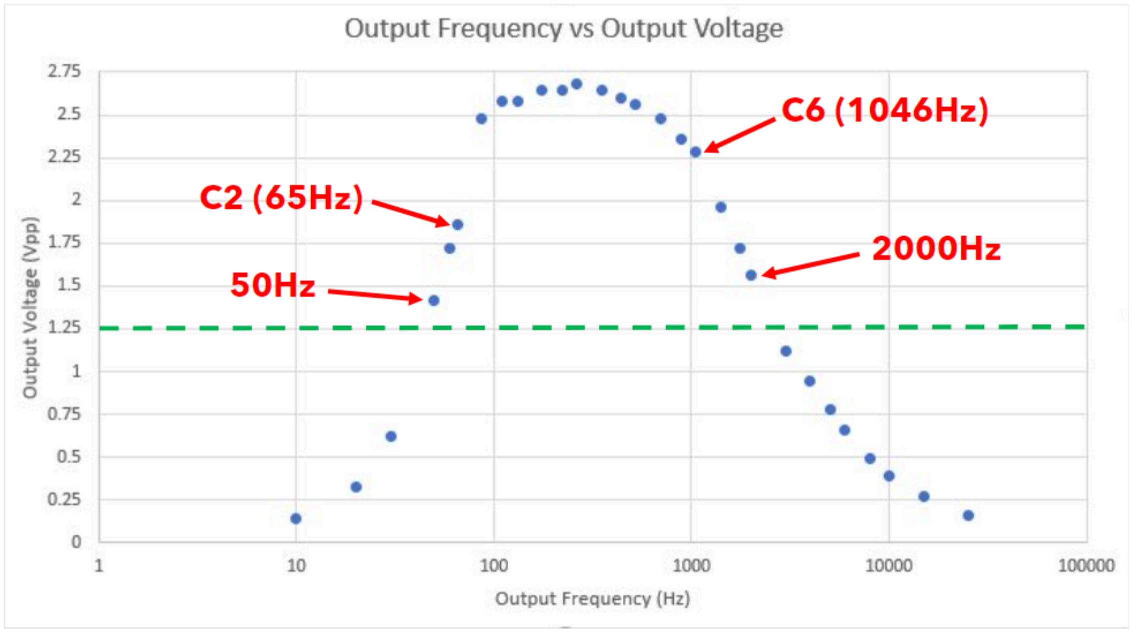


Figure 21. Filter Frequency Response Analysis

Throughout the PCB design and assembly process, IPC industry standards were taken into consideration to ensure the integrity of the circuitry was up to par. IPC-2581 is "A generic standard used when sending information between a PCB designer and a manufacturer or assembly company." (Millennium Circuits Limited, n.d.). When ordering the PCB, all files were checked for correct formatting and it was confirmed they were correctly uploaded to JLCPCB, the manufacturing company from which the PCB was ordered.

Another IPC standard, IPC-2221 addresses design layout, electrical properties, thermal management, and more. In design, the PCB was rule checked for design constraints such as trace width and spacing, component placement, via sizing, and more to ensure signal integrity across the board (Millennium Circuits Limited, n.d.). In testing, once the PCB was in hand, each submodule was assembled and continuity tested to ensure closed loops and proper grounding and avoid shorting once power was applied. The assembled PCB is pictured below.

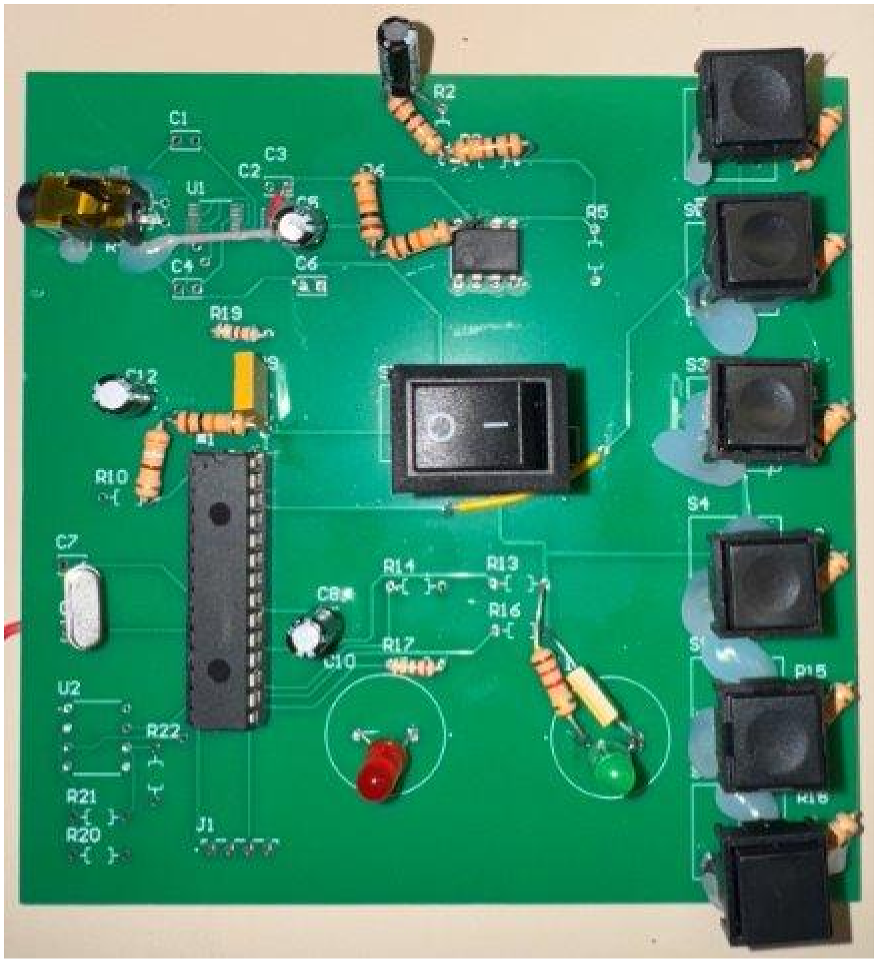
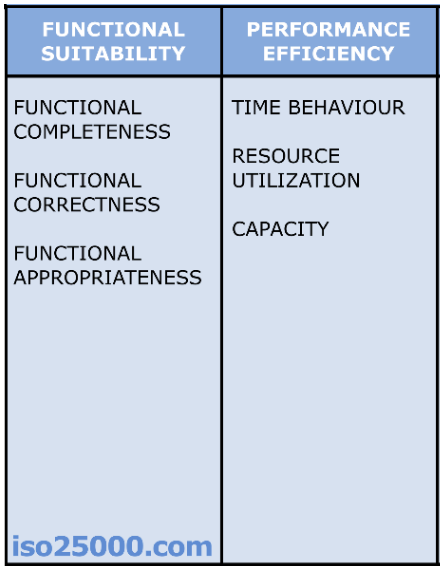


Figure 22. Assembled PCB

7.2 Software System Testing

For our software quality and testing evaluation standard, we chose ISO 25010 due to its rigorousness and flexibility for assessing the performance of multi-device software systems (ISO/IEC 25000, n.d.). Particularly, we focused on suitability and performance (detailed in figure 23.).



FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY
FUNCTIONAL COMPLETENESS	TIME BEHAVIOUR
FUNCTIONAL CORRECTNESS	RESOURCE UTILIZATION
FUNCTIONAL APPROPRIATENESS	CAPACITY

iso25000.com

Figure 23. ISO 25010 Standard Guidelines

ISO 25010 vs *SheetSavvy* General Software Testing

Category	Description	SheetSavvy Application	Status
Functional Completeness	Degree to which the set of functions covers all the specified tasks and intended users' objectives.	<u>User's Objectives</u> - Use all peripheral controls - Easily make sheet music from played notes	Fully Functional
Functional Correctness	Degree to which a product or system provides accurate results when used by intended users.	<u>Timing and Pitch Detection Accuracy</u> - Degree of correct pitches reported - Degree of correct timings reported	Above Average
Functional Appropriateness	Degree to which the functions facilitate the accomplishment of specified tasks and objectives.	<u>Peripherals to demonstrate accomplished tasks</u> - Recording LED - Power LED - Metronome Countdown - Start/Stop Button	Fully Functional
Time Behaviour	Degree to which the response time and throughput rates of a product or system, when performing its functions, meet requirements.	<u>Time Sensitive Objectives</u> - Metronome generation - Sampling Loop Time - FFT computation Time	Above Average

ISO 25010 vs *SheetSavvy* General Software Testing			
Category	Description	SheetSavvy Application	Status
Resource Utilization	Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.	<u>Resource Usage</u> - On-Chip Memory profile (16 kB, 94% utilization) - Desktop memory profile (100 ± 20 MB)	Average
Capacity	Degree to which the maximum limits of a product or system parameter meet requirements.	<u>Limits of Product</u> - Keyboard range - User pressure on key press - Speed of notes played	Above Average

Table 4. Embedded vs Industry Standard Testing

Our general method for assessing full system accuracy involves examining pitch detection and timing detection accuracies separately. All comparisons are conducted between what the user played and the device output. We assessed pitch detection by comparing whether the pitch detected for a note (or notes in case of timing detection failure) matches that which was played by the user. We assessed timing detection accuracy by comparing if onsets and releases occur at the same time in both scores. In the event of an extra timing, we quantitatively graded the individual accuracy based on percentage of correct timings. For example, if the system splits a note in half, the score awarded is 2/4 since one onset/release pair is correct and the other is not.

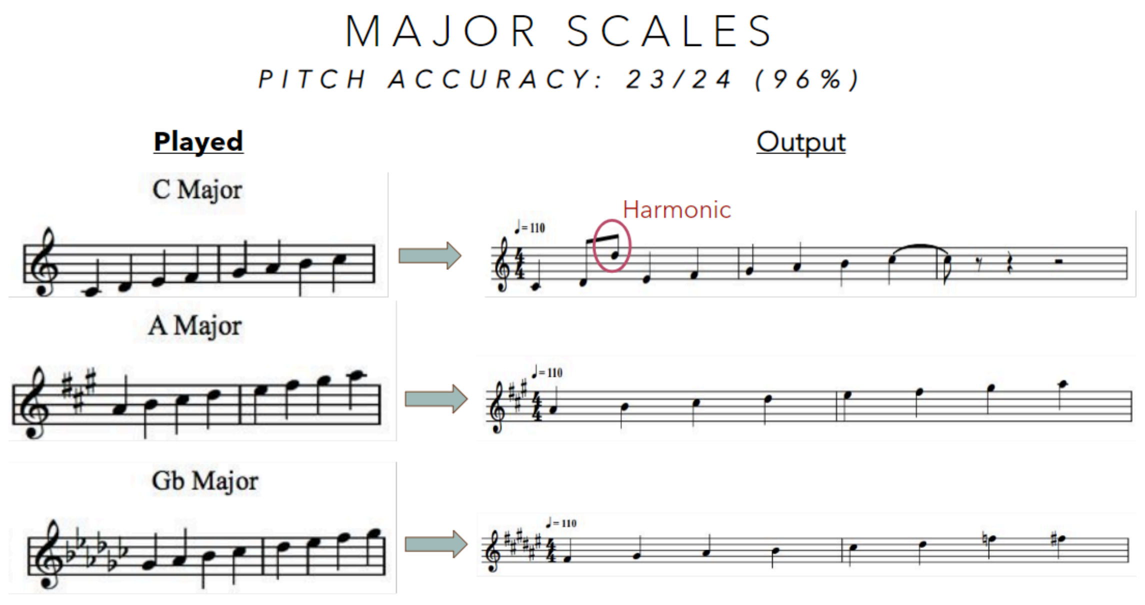


Figure 27. Major Scales Output vs Played

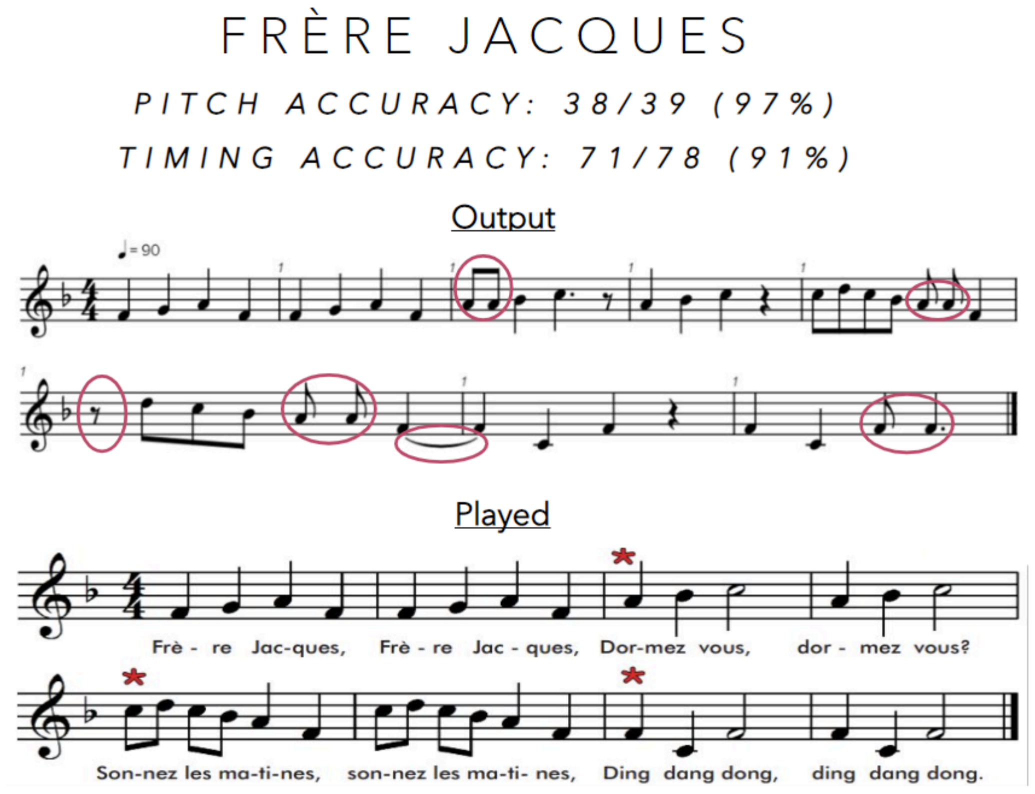

FRÈRE JACQUES
PITCH ACCURACY: 38/39 (97%)
TIMING ACCURACY: 71/78 (91%)
Output
♩= 90
Played
Frè - re Jac-ques, Frè - re Jac - ques, Dor-mez vous, dor - mez vous?
Son-nez les ma-ti-nes, son-nez les ma-ti- nes, Ding dang dong, ding dang dong.

Figure 28. Frere Jacques Output vs Played Notes

8 New Skills Acquired and Learning Strategies

8.1 Emmy Ploskina

In junior design and ECE 1212, I was briefly introduced to PCB design and filter design. For this project, I relied heavily on the Altium Designer tutorials and handbook to ensure the integrity of the PCB, and for op amp filter resources, I researched filter solutions using a 2nd order Butterworth filter model for SheetSavvy applications.The final and most important skill I adopted throughout the semester was circuit troubleshooting and failure analysis, as many unexpected challenges arose when assembling the PCB that required creative troubleshooting to achieve the desired functionality.

In junior design, I learned the basics of PCB design, but did not understand the value of the rules to ensure the PCB that is designed performs as best as possible for the application. From the Altium handbook, I learned how to apply ground pours to make sure the grounds throughout the circuit are consistent, I learned how to create vias to maximize the number of components that could fit on the PCB, and I learned the importance of trace length matching within the auxiliary input domain, which was important for the V+ and V- balancing of the signal from the keyboard. (Altium, n.d.)

In ECE 1212, I was introduced to the world of signal filtering, but had to refresh my memory on how to apply a DC offset to shift the signal above the 0V axis, and referenced Electronics Tutorials and Texas Instruments op amp filtering guides to fine tune the second order bandpass filter. (Electronics Tutorials, n.d.) (Carter, 2022) Also, throughout the semester my troubleshooting skills and ability to determine root cause failure by tracing circuits, perform continuity tests, measure signals on an oscilloscope, and test components for functionality and tolerances drastically increased.

While much of my research at the beginning of the semester led to dead ends, the items listed above were the reason for the success of the hardware module and the end product as well. The filtering sources helped me to design a second order low pass filter to create an intentional DC offset, while still applying the desired gain of 10 and maintaining the band passing frequency range the design needed. And for the PCB designing, while there were several design changes between ordering and receiving the PCB, the design practices applied from the Altium tutorials were extremely beneficial for the integrity of the signal and the ability to send data to and from the microcontroller.

8.2 Evelyn Pitts

With my past ECE classes and co-op, I have experience with embedded C coding. However, for this project, I became much more familiar with the MPLAB IDE and better understood common embedded coding errors. For example, common errors that occurred during this project were usually related to available memory whether that be the FFT size, overlap buffer, etc… At first, I struggled a lot with being able to find the source of these errors as it doesn't specifically tell you until I learned to adjust the sizes of these variables on a global level to make my life much easier so I could always revert back.

As simple as it may sound, difficulties in the actual process of coding the microcontroller took up most of my concentration during the first half of the semester. This may seem like an unnecessary setback, but once I have those things figured out I feel more capable of problem solving in the future when I'm coding on a lower level rather than on a development board. I also increased my proficiency in dissecting data sheets and documentation through the process of using the Microchip DSP library. (Microchip, n.d.) (Microchip, n.d.)

Additionally, I haven't taken an ECE class regarding Fourier Transforms since ECE0402. This meant that during the course I was learning new tricks that some other students may have been taught during classes like Digital Signal Processing. The research I needed to do for this was pretty expansive as I needed to become familiar with more terminology even being able to implement it. Firstly, I learned about common frequency bins that will always have peak noises that are good to be filtered out through hardware and software. An example of this is when I was giving our midterm presentation and displayed a large peak in the zero-th bin because I didn't fully understand how to read those different graphs yet in the frequency domain. (NTI Audio, n.d.)

After I got my baseline FFT working, I began doing additional research on windowing to see which window would apply the best to our application and the specific problems we were already facing. Initially, I debated between Hanning and Hamming due to their ability to suppress similar side lobes on the frequency graphs. In the end, we settled on the Blackman window as it is very similar to Hamming, but has a much clearer difference in amplitude and also helps with our timing detection. (National Instruments, n.d.)

8.3 Justin Pacella

During this project, I enhanced and broadened my engineering skill set. One skill I learned (and had little prior experience with) is digital signal processing (DSP) in the context of embedded systems. My teammates and I were able to craft a clearly defined DSP pipeline for detecting the pitches, onsets, and releases of musical notes from an analog audio signal. A few sub-skills I acquired are

- Analog to digital conversion
- High performance fixed-point arithmetic (Q15 and Q16 data types)
- Time domain signal windowing functions
- Time domain signal overlap design
- Time domain signal magnitude calculation (RMS energy)
- High performance fourier transforms for digital signal controllers (including frequency binning and frequency resolution computation)
- Low memory harmonic product spectrum computation
- Frequency domain signal magnitude calculation (Spectral power)

Another skill I learned was the use of Microchip software and hardware. This includes the use of MPLAB IDE for embedded software development and part of its extensive set of design tools. It also includes register configuration specifics for our microcontroller and other controllers in the same family. Finally, I was able to develop my music composition skills and write my very own magnum opus.

9 Conclusions and Future Work

This semester taught our entire group so much new knowledge, strategy, and teamwork skills we will take with us after graduation. As we previously stated in the introduction, the requirements we set for ourselves were high but doable and we did achieve the majority of them. Our device had all of the user controls we set out to provide and nearly met our pitch and timing requirements of consistent 95% accuracy. The final product was able to perform well during our live demo and produce sheet music.

Within our group, our best ability was simply understanding our strengths and applying them accordingly. Despite the range of musical notes being less than we desired, the abilities that our device was able to accomplish within the notes we detected were robust and consistent. Harmonics on notes was the biggest limitation/setback that we faced throughout the semester in multiple areas of our design. Although we sought advice from Dr. Jacobs and implemented the harmonic product spectrum, they still showed up throughout our testing. Overall, we certainly learned the time needed to go through the necessary trial and error which is the engineering design process.

If we had to start at the beginning of this project again, we would put more focus and manpower towards pitch detection at the beginning of the semester since we have found out that timing detection is very reliant on pitch. As we've discussed for many other projects during our undergraduate careers, we would have also started testing and integrating sooner so we could learn more and learn faster to be able to adapt near the end of the semester. On the other hand, if we simply had more time with the project we have already started then the focus would go towards maximizing the memory we have available and applying that to our embedded module, as we did not realize just how much memory pitch detection would use. This could be done by researching more microcontrollers from the same company so we could continue to use the same IDE or implementing an EEPROM chip to provide more nonvolatile memory.

10 References

Altium. (n.d.). *Tutorial - A Complete Design Walkthrough with Altium Designer*. Altium. Retrieved December 14, 2024, from https://www.altium.com/documentation/altium-designer/tutorial-complete-design-walkthrough

Bello, J. P., & Daudet, L. (2005, 8 15). *A tutorial on onset detection in music signals*. IEEE Explore. Retrieved 9 25, 2024, from https://ieeexplore.ieee.org/abstract/document/1495485

Benetos, E., & Dixon, S. (2011, 7 11). *Polyphonic music transcription using note onset and offset detection*. IEEE Explore. Retrieved 9 16, 2024, from https://ieeexplore.ieee.org/document/5946322

Carter, B. (2022, February). *Designing Gain and Offset in Thirty Seconds*. Texas Instruments. https://www.ti.com/lit/an/sloa097/sloa097.pdf?ts=1734234374766&ref_url=https%253A%252F%252Fwww.bing.com%252F

Dharia, A., & Gummattira, R. (2009, June). *Signal Processing Examples Using the TMS320C67x Digital Signal Processing Library (DSPLIB)*. TI.com. Retrieved September 18, 2024, from https://www.ti.com/lit/an/spra947a/spra947a.pdf?ts=1726703026609&ref_url=https%253A%252F%252Fwww.google.com%252F

Drawlib Developers. (2024). *Drawlib*. Welcome to the Drawlib Documentation! — drawlib 0.2 documentation. Retrieved September 25, 2024, from https://www.drawlib.com/docs/v0_2/

Electronics Tutorials. (n.d.). *Second Order Filters | Second Order Low Pass Filter*. Electronics Tutorials. Retrieved December 14, 2024, from https://www.electronics-tutorials.ws/filter/second-order-filters.html

Fergus O'Kane Microchip Technology Inc. (2015, July 14). *Implementing the Fast Fourier Transform (FFT) on dsPIC® Digital Signal Controllers*. Microchip.com. Retrieved September 9, 2024, from https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ApplicationNotes/ApplicationNotes/90003141a.pdf

Good, M. D. (2009). *Using MusicXML 2.0 for Music Editorial Applications | Songs and Schemas*. Michael Good. Retrieved September 23, 2024, from https://michaelgood.info/publications/music/using-musicxml-2-0-for-music-editorial-applications/

Hotta, S. (2021, January 22). *Difference Between Active and Passive Filter (with Comparison Chart)*. Circuit Globe. Retrieved September 26, 2024, from https://circuitglobe.com/difference-between-active-and-passive-filter.html

ISO/IEC 25000. (n.d.). *ISO 25010*. ISO/IEC 25000. Retrieved December 14, 2024, from https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Library of Congress. (2024, 5 8). *MusicXML File Format Family*. Library of Congress. Retrieved September 25, 2024, from https://www.loc.gov/preservation/digital/formats/fdd/fdd000358.shtml

MakeMusic. (n.d.). *MusicXML Publications*. MusicXML. Retrieved September 25, 2024, from https://www.musicxml.com/publications/

Marwana, U. (2024, 9 11). *Passive Low Pass Filter*. Electonics Tutorials. Retrieved 9 25, 2024, from https://www.electronics-tutorials.ws/filter/filter_2.html

Microchip. (n.d.). *Implementing the Fast Fourier Transform (FFT) on dsPIC® Digital Signal Controllers*. microchip.com. https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ApplicationNotes/ApplicationNotes/90003141a.pdf

Microchip. (n.d.). *MPLAB XC16 Libraries Reference Guide*. microchip.com. https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ApplicationNotes/ApplicationNotes/90003141a.pdf

Millennium Circuits Limited. (n.d.). *IPC Standards for Printed Circuit Boards | PCB Design*. Millennium Circuits Limited. Retrieved December 14, 2024, from https://www.mclpcb.com/blog/ipc-standards-for-pcbs/#standars

Minor, K. A., & Kartowisastro, I. H. (2022, August 6). Automatic Music Transcription Using Fourier Transform for Monophonic and Polyphonic Audio File. *Ingénierie des Systèmes d'Information*, *27*(4), 1-7. International Information and Engineering Technology Association. https://doi.org/10.18280/isi.27041

National Instruments. (n.d.). *Understanding FFTs and Windowing*. https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf

NTI Audio. (n.d.). *Fast Fourier Transformation FFT - Basics*. www.nti-audio.com. https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft service bc. (2024, August 1). *Balanced XLR Cables: Audio Fidelity at Its Best*. Better Cables. Retrieved September 26, 2024, from https://bettercables.com/blogs/learn-about-cables/balanced-xlr-cables-the-pinnacle-of-audio-fidelity-for-audiophiles

SoundSlice. (n.d.). *Free MusicXML viewer from Soundslice*. Soundslice. Retrieved September 23, 2024, from https://www.soundslice.com/musicxml-viewer/

Texas Instruments. (n.d.). *INA 165x SoundPlus High Common-Mode Rejection Line Receivers datasheet (Rev. B)*. https://www.ti.com/lit/ds/symlink/ina1650.pdf?ts=1727296305927&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FINA1650