Software Design Specifications
Version 1.0

Date: February 12, 2021

Team Name: SongBird

Project Sponsor: Paul Flikkema

Team's Faculty Mentor: Andrew Abraham

Team Members: Kevin Imlay, Daniel Mercado, Yasmin Vega-Nuno, Anqi Wang

Table of Contents

1. Introduction

Birds play an essential role in the health and development of many ecosystems around the world. Pest and insect populations are regulated by bird predation, dead animals are disposed of by bird scavenging, and many seeds rely on birds for distribution and priming for sprouting. For example, in India, vultures (*Cathartes aura*) are responsible for an estimated $34 billion worth of clean-up of cow carcasses. In Sweden, it is estimated that it will cost $9,400 per hectare for human seed dispersal services if the Earasian Jay (*Garrulus glandarius*) were to disappear from their oak forests. In Jamaica, bird predation on pest insects increases coffee bean production and quality by $310 per hectare per year.

Despite the extreme importance of birds, the scientific community still has numerous questions surrounding the behavior of birds and the ways that they communicate. Questions such as where birds eat and nest, how they move from area to area, and how they communicate with each other remain unanswered. More importantly, it is not known how these behaviors are changing in response to sound pollution, contact with humans, and climate change.

Dr. Flikkema, a professor of Computer Science and Electrical Engineering at Northern Arizona University, is working with a research team at Politecnico di Milano, in Italy, with the aim of helping scientists conduct more meaningful research on birds. This team is creating an Artificial Intelligence tool to classify bird sounds based on audio recordings. This tool will be able to identify bird species, individual birds, and how many birds are present. Currently, however, they are developing this tool using a library of audio recordings that don't include features such as environmental and human-made noise such as vehicles, people, and wind. BiVo will allow Dr. Flikkema and other scientists to deploy low-cost and open-source sensors to gather data on bird vocalizations and features as explained in the sentence prior. With this data, scientists will be able to conduct meaningful research towards our understanding of birds and their behaviors.

Together with Dr. Flikkema, we proposed a structure for the functionality and collaboration of BiVo devices and user interfaces. The structure is an open source and fully documented project to allow users to add their own modules.

The current state of play of existing recording devices does not meet our client's needs. Current in person monitoring solutions are costly and many low cost alternatives are difficult to use and have limited support for extensibility. Moreover, they may not be open-source. Popular recording devices include the AudioMoth, a cheaper and popular alternative to other audio sensors on the market. Although, even this device lacks the extensibility our client is looking for, nor does it have the ability to filter out audio that does not contain bird vocalizations on site.

We are developing BiVo, a front-end-back-end system for recording bird vocalizations. BiVo will be open-source and modular for easy modification and addition of functionality. The BiVo device will be capable of streaming audio and saving recordings for later retrieval. Furthermore, the BiVo device will use the EFM32GG12 Thunderboard for main recording and audio analysis on site. The desktop application will be able to operate alone or interface with Matlab for easy data analysis. The user interface will mix the Matlab application designer and Python to interact with the BiVo device to collect data from it.

In this report, we will discuss our implementation overview, architecture overview, module and interface description, and our implementation plan of our project.

2. Implementation Overview

While working together with Dr.Flikkema, we have come up with a structure for how the BiVo device and user interface will function and work together that is open source and well documented enough for anyone to add modules of their own. This BiVo device will be using the EFM32GG12 Thunderboard to do the main recording and audio analysis on site. The user interface will use a mixture of Matlab application designer and Python to interact with the BiVo device in order to gather data from it. This is explained more in detail in our Technology Feasibility document where we explain why we chose the EFM32GG12 Thunderboard as well as a Matlab application designer with Python integration.

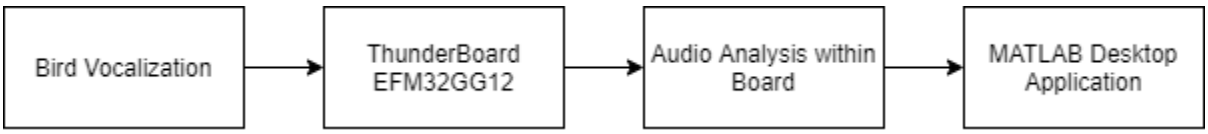

Figure 1. Flow chart of the general process of gathering audio data with the ThunderBoard.

Our solution is described by Figure 1. First going into the BiVo device is a bird vocalization that will then be analyzed by the board shortly after it is picked up to be within the frequency range. After this, the audio snippet will be discarded or saved based on if there was an audio snippet. Then, the user will be able to connect BiVo to a computer with the user interface installed. From there, the user will be able to take out specific audio snippets that they need from the BiVo device for later analysis. This will help create a standalone device for later recovery.

2.1. Technology

In order to fully explain the BiVo system along with the user interface, these are the technologies we are using to implement them:

EFM32GG12 Thunderboard- This is an embedded system that uses two small PDM microphones and a EFM Giant Gecko 12 MCU. This is used to capture and analyze data on site in order to tell what a bird call is. This will be the central system to gathering audio clips of bird vocalization.

Matlab application designer- This is the Matlab supplied system that we will design the whole graphical user interface in so that people can fully use Matlab to import and export the audio clips of birds. It allows us to integrate python scripts into it as well to allow for more complicated scripts in audio analysis and communication.

Python integrated with Matlab UI- This will contain functions that should be easy to edit and adjust to whatever needs the user has for the data. This will allow the application to be extensible and well documented on what functions to change if they require something different.

3. Architecture Overview

BiVo's architecture is divided into two main groups: the sensor that is responsible for recording, analyzing, and saving or streaming audio recordings of bird vocalizations, and the desktop application that lets the user interact with the sensor and download the audio data it collects. The desktop application can be further divided into a graphical user interface in Matlab and a python script routine that interfaces the sensor with Matlab.

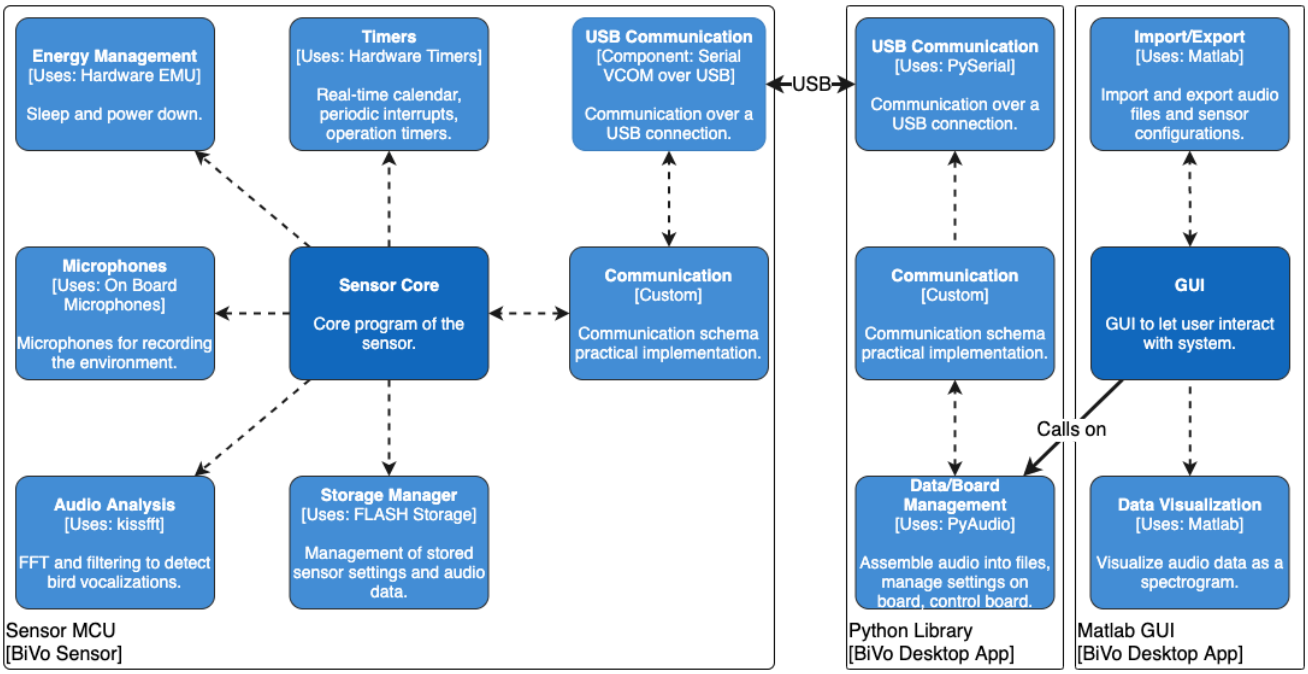

Figure 2. Bivo system component architecture. Blue boxes represent modules and white boxes represent components. Dashed arrows represent dependencies between modules and the solid arrows represent communication between components.

The sensor component is the most important element of the entire system. This component is responsible for everything related to recording bird vocalizations and saving or streaming to the desktop application. The sensor component will use:

- A module to manage the on-board microphones for recording,
- A module for audio analysis for determining if audio contains bird vocalizations,
- An energy management module to sleep the sensor for power saving,
- A timer module for working with a real-time calendar an periodic event triggers,

- A module for managing the saving and retrieval of audio data and sensor data to device storage,
- A module to manage communication over a USB connection,
- A module for the practical implementation of the communication schema.

Audio from the microphones will be passed on to the audio analysis module to determine if bird vocalizations are in the audio. If there are, the audio data will be passed to the storage module for later retrieval or to the communication modules for streaming.

The Python component is the core of the desktop application, encapsulating the functionalities of managing the connected sensor and the data to and from the sensor. The Python script will incorporate:

- A module for managing the connection with the board over a USB connection,
- A module for the practical implementation of the communication schema,
- A module for accepting audio data streams and assembling it into audio files for passing on to Matlab to use.

As data comes from the board, it will be accepted by the USB communication module and passed to the general communication module. The general communication module will process the data according to the type of data received and pass it along to the data management module for assembling the data into a format that can be passed on to Matlab or exported. Additionally, the Python component will be able to run stand-alone from Matlab to allow users without access to Matlab to still use the system from a command line interface.

The graphical user interface (GUI) component to the system will be the main method for users to interact with the system. This GUI will allow the user to manage the board's settings and download stored or streamed audio, and the GUI will let the user load audio data into Matlab for more advanced analyses. To make this possible, the GUI component will incorporate:

- A visualization module to show the downloaded audio data in a spectrogram (visual representation of frequency and loudness plotted against time) format.
- An import/export module to export and import audio files and sensor configurations on the desktop computer.

The GUI will be built on top of the core functionality of the Python component and provide graphical means to control the Python component.

4. Module and Interface Descriptions

BiVo's components can be broken down into many smaller modules that work together to carry out the needs of each component. In this section we will give a description of each module's purpose and responsibilities, a diagram showing the functionalities of the module, and a description of the public-facing interface to that module.

4.1. BiVo Sensor Component

4.1.1. Communication (General)

The general communication module in the sensor component is used to communicate with the general communication module in the Python library component on the Desktop Application. The purpose of the communication module implements the communication schema as the slave in the master-slave relationship, and is abstracted away from the specific communication medium being used. This is to allow future users to incorporate other medium-specific communication such as Bluetooth or WiFi without needing to rewrite major parts of the communication schema to work with it.

This communication module has the following responsibilities:

- Accept incoming commands from the desktop.
- React appropriately to these commands in the form of changing settings, modes, or state of the sensor, or responding with data requested by the desktop application.
- Handle connection and disconnection handshakes to properly handle connections and disconnections.

There are two main functions within the general communication module, one to handle incoming commands from the desktop application, and one separate one to send streaming data to the desktop application.

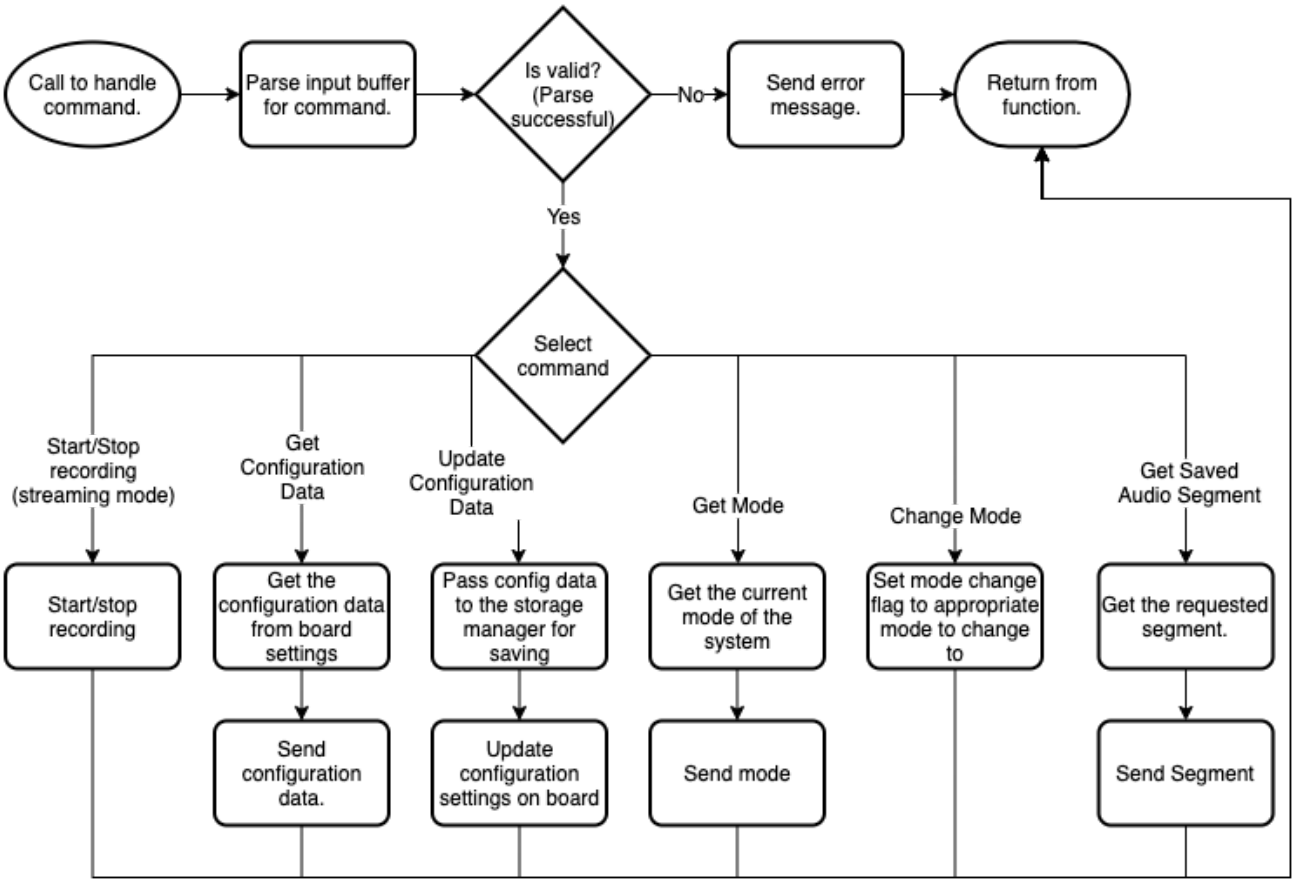

Figure 3. Callback function to handle a message that has just been received from the desktop application.

Figure 3 shows the function for accepting a message that has just been received. From the desktop application. This is called as a callback function by the receive function in the USB Communication module when a transmission has been received in full. This function also calls on the send function of the USB Communication module to send back the data the desktop application has requested or a status/error code.

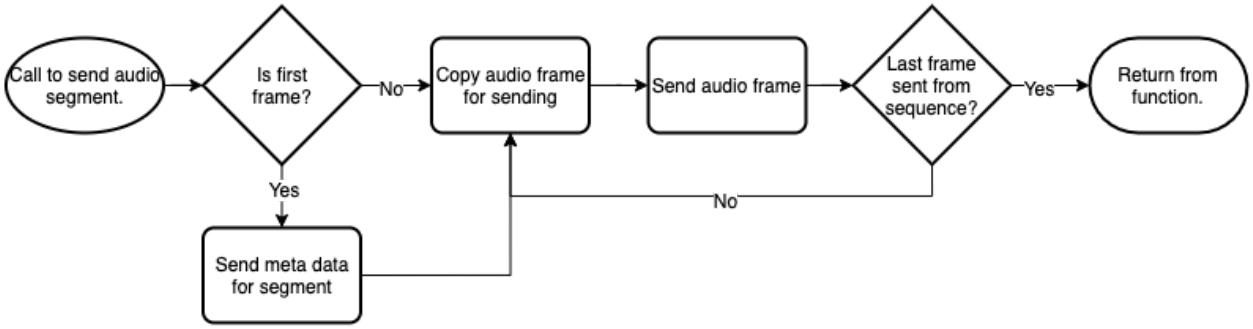

Figure 4. Function to send a segment of audio data to the desktop application.

Figure 4 shows the function for sending audio data to the desktop application as it is being recorded (streaming). This function is called by the core system of the sensor when there is audio data to be sent and calls the USB Communication module's send function to pass the data to the desktop application.

4.1.2. USB Communication

The USB Communication module in the sensor component is used to communicate with the USB Communication module in the Python library component of the desktop application. The purpose of this module is to handle all the USB-specific functionalities of communication for the general communication module to use. This allows for easy replacement of this module with another medium-specific module (such as Bluetooth or WiFi) without the need to rewrite major parts of the general communication to work with it.

This communication module has the following responsibilities:

- Accept incoming data over the USB connection from the desktop application.
- Send outgoing data over the USB connection to the desktop application.
- Call on the General Communication module when a message has been fully received.

There are three main functionalities of the USB Communication module, a function to initialize the USB serial driver, a function to begin receiving data from the desktop application, and a function to send data to the desktop application.

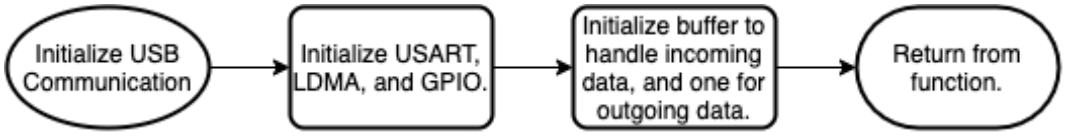

Figure 5. Initialization function for setting up the USB serial driver.

Figure 5 shows the steps taken to initialize the USB serial driver before a connection can be made with the desktop application. This needs to be called before any other USB communication functions can be used.

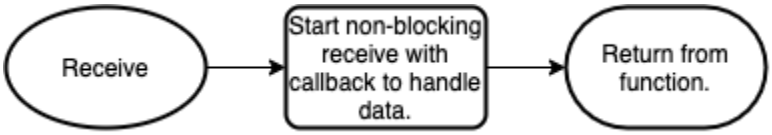

Figure 6. Function to begin a non-blocking receive over the USB connection

Figure 6 shows the steps to start receiving data from the desktop application. Once called, the receive operations run in the background using direct memory access to store all incoming data into a buffer. This allows the sensor to not need to intervene with incoming data until it is ready to be processed. Once the buffer is full, a callback function passes the message on to the General Communication module to processing.

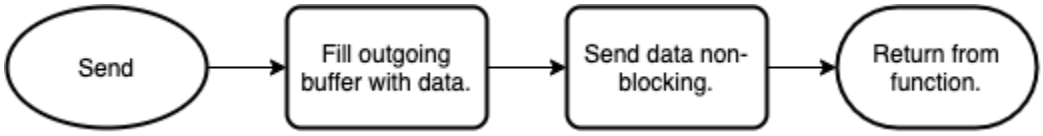

Figure 7. Function to begin a non-blocking send over the USB connection.

Figure 7 shows the steps for sending data to the desktop application. Once called, the send process runs in the background using direct memory access so the sensor can continue its operation in the meantime.

4.1.3. Timers

The Timer module is in the sensor board. The timer module is used to get the current time and set a timer when needed. The Sleeptimer driver provides software timers, delays, timekeeping and date functionalities using a low-frequency real-time clock peripheral. [1]

Getting the current time can provide users the choice of a fixed-point of time to start recording. This function has the following responsibilities:

- Convert the time into Unix timestamp
- Get current time and date
- Let users set current time in date format

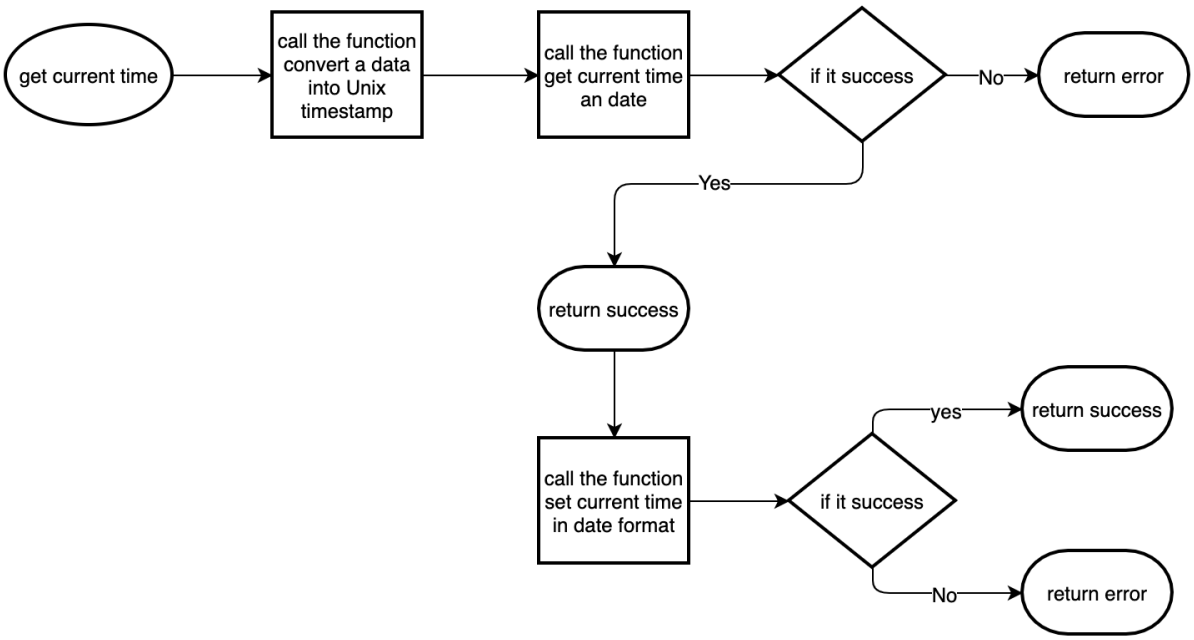

Figure 8. Function to get current time and date.

The timer can also set a timer when needed. This starting timer at the fixed time function has the following responsibilities:

- Restart the timer
- Get the status of timer
- Start a 32 bit timer

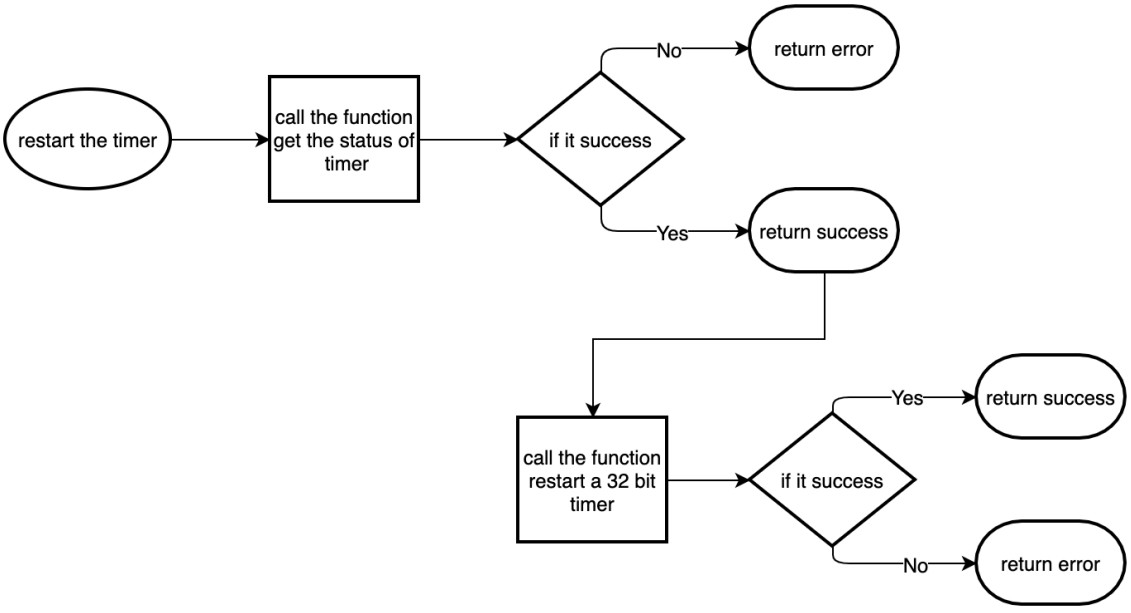

Figure 9. Function to restart the timer.

4.1.4. Data Storage

The Data Storage Management module is used to manage the storing and retrieving of audio data and board settings on the board's FLASH memory. The purpose of this module is to encapsulate the functionalities needed for platform-specific read and write functions while giving a public interface for saving and retrieving data between power cycles.

This data storage module has the following responsibilities:

- Save and retrieve sensor configuration settings.
- Save, retrieve, and delete audio segments and their respective metadata.
- Report when storage is full.

There are seven main functionalities of the Data Storage module:

- Initializing the drivers needed to perform read and write operations and loading the directories for finding saved data.
- Saving the directories of saved data for loading again after a power cycle.
- Saving sensor configuration settings.
- Reading sensor configuration settings.
- Saving an audio segment with its metadata.
- Reading an audio segment with its metadata.
- Deleting an audio segment with its metadata.

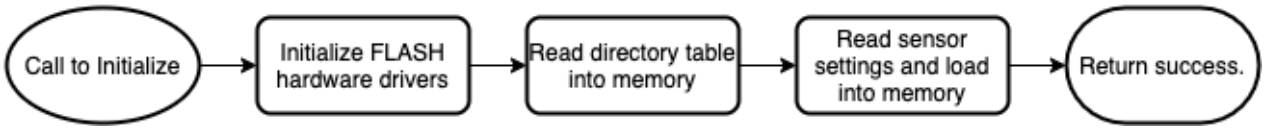

Figure 10. Function to initialize the Data Storage module before use.

Figure 10 shows the steps for initializing the drivers for reading and writing to FLASH storage on the board as well as loading the directories of saved data and configuration settings. This needs to run before any read or write functionalities can be used.

To save the directories to FLASH when powering down the sensor or before sleeping the sensor, the sensor will simply have to save the

directory table. Without saving the directories, it would not be possible to find the locations of saved audio segments once the board powers on again.

Similar to the functionality to save directories, saving the sensor's configuration settings needs to be called before the board powers down or sleeps to make it possible to retrieve the data once the board powers back on.

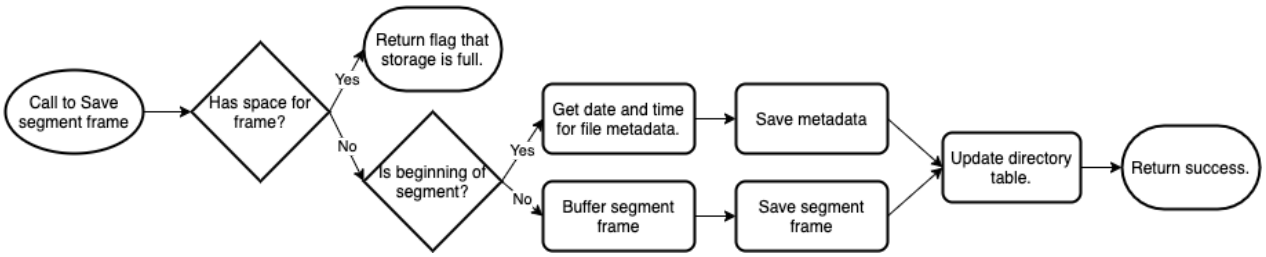

Figure 11. Function to save audio data frames as they are being passed in.

Figure 11 shows the steps taken to save a frame of an audio segment. Due to the limited RAM on the sensor, audio segments can't be handled as a full length segment but rather are split into many smaller frames that sum to create the full segment. It is not possible to store an entire segment then, so the segment must be progressively saved while it is being recorded.

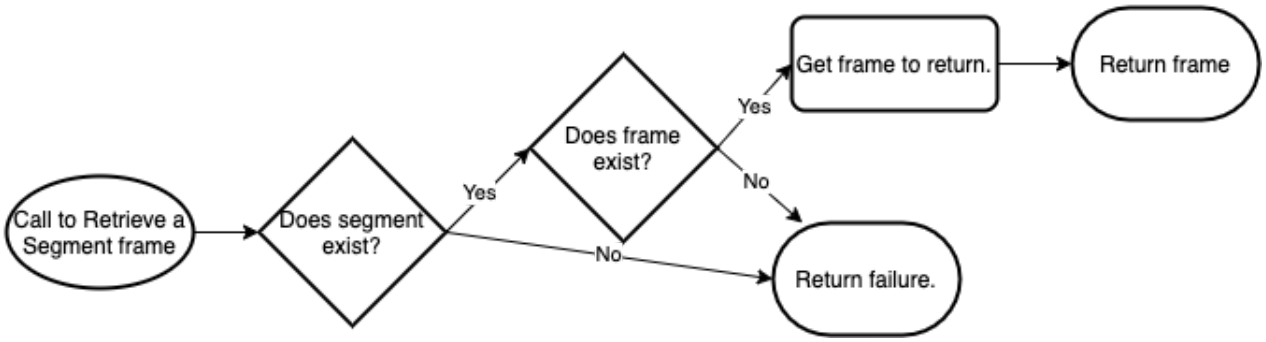

Figure 12. Function to read saved audio data frames to pass on to the desktop app.

Figure 12 shows the steps taken to read an audio segment from storage. Similarly as with saving an audio segment, reading a segment must be done so in many small frames. Reading an audio segment is only useful for sending to the desktop application, so as the segment frames are read they are passed to the general communication module by the sensor core for sending to the desktop application.

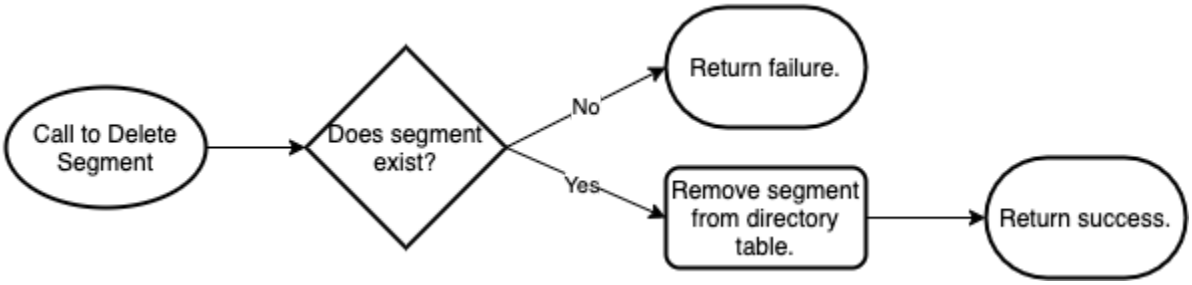

Figure 13. Function to delete an audio segment from the storage.

Figure 13 shows the steps taken to delete an audio segment from storage along with its metadata. It is not necessary to overwrite the actual location of the audio storage so deleting an audio segment is really a matter of deleting the reference to its location. This frees the location to be used by another audio segment.

4.1.5. Audio Analysis

This module is in charge of handling audio analysis on the board to tell which audio snippets are significant. This takes in an audio buffer and performs an FFT on it to then determine the frequency range of this and decide what is important within the buffer. This will pass on the data to the next module depending on the mode the sensor is set on.

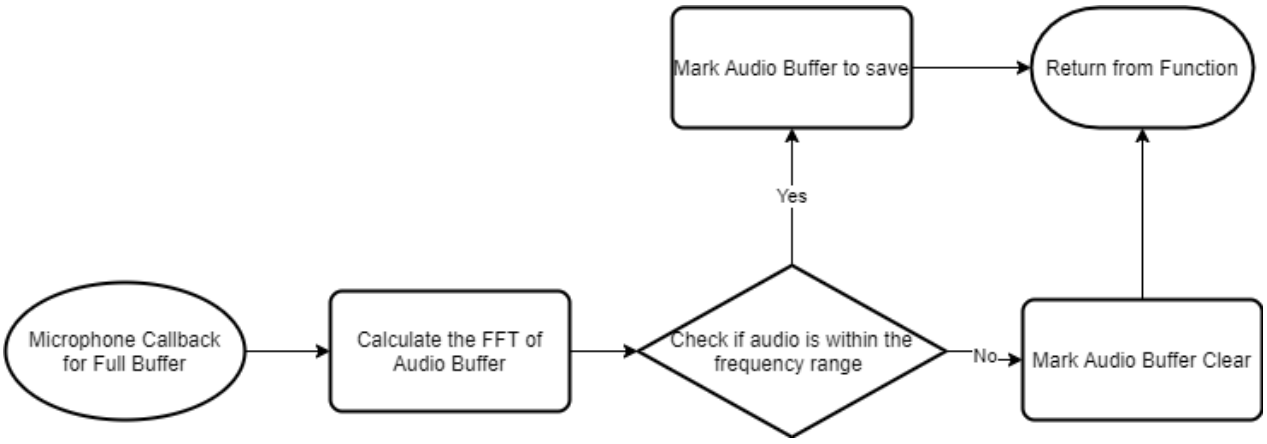

Figure 14. This chart shows the main functionality of the Audio Analysis showing the path of what happens after a Callback happens from microphones to the returning a buffer marked with either clear or save.

Some of the data this module will save and need is listed below:

- Audio Buffer to Analyze
- Frequency Range
- Audio Buffer with FFT done

The functionality of this module will be to compute the FFT and then based on that FFT be able to determine that the audio snippet is within the frequency range. Listed below is some of the public functions within this module:

- calculateFTT(audioSnippet,audioSnippetNum) - This will calculate the FFT based on the audio SnippetGiven.
- determineHit(audioSnippetFFT,audioSnippetNum) - This will determine based on the FFT if the audio has a bird call within the frequency range.
- markHit(audioSnippet,audioSnippetNum) - This will mark the audio buffer as a hit for it contains a bird call within the frequency range.

4.1.6. Recording

Recording module is used to record the voice of birds in preparation for streaming.

In the recording module, the main parts are initializing the microphone and preparing the buffer.

We need to initialize the microphone before starting recording to avoid malfunction. The initialize function has the following responsibilities:

- Initialize the microphone
- De-initialize the microphone

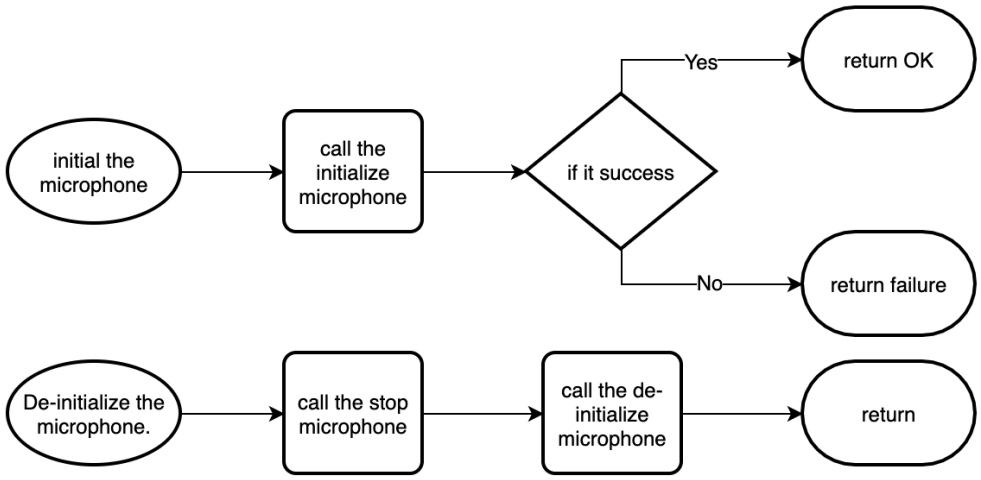

Figure 15. Function to initialize the microphone.

When the sensor starts recording, we will put the data into the buffer to prepare for steaming to a buffer or desktop application. The buffer function has the following responsibilities:

- Check if the buffer is ready
- Wait until the buffer is ready
- Put the samples into the buffer
- Start streaming

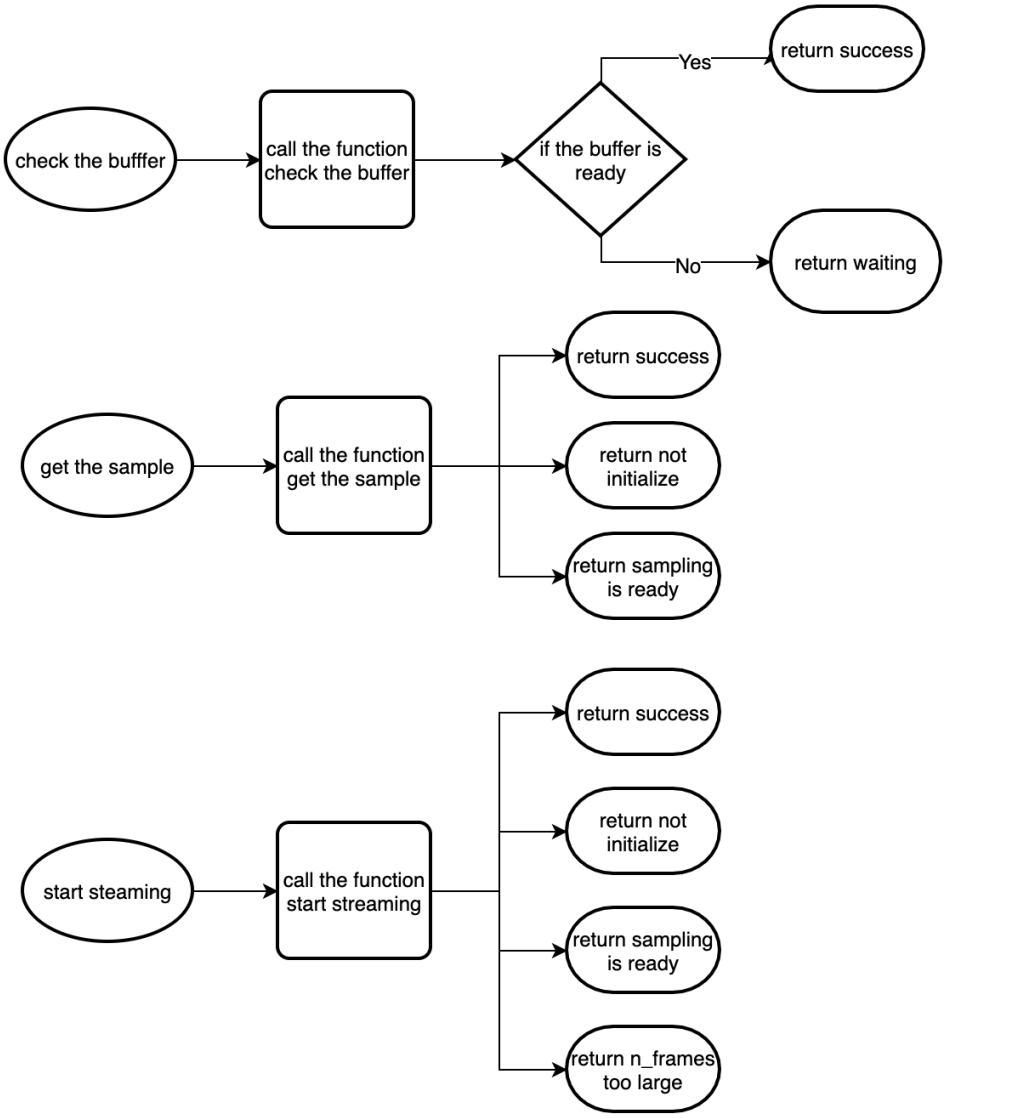

Figure 16. Function to start streaming.

4.1.7. Energy Management

Energy Management mode is in the sensor board which can help the board to save energy and stay for a longer time. Energy management module is used to let the sensor shut off or sleep to save more energy. The mode transition part is very important in the energy management module which can switch the mode when the sensor is needed.

This module has several energy modes which list as follow:

- Sleep mode(which includes normal sleeping and deep sleeping): sleeping other than working time to save energy
- Shut off mode: save data and shut off the board when the power is very low.

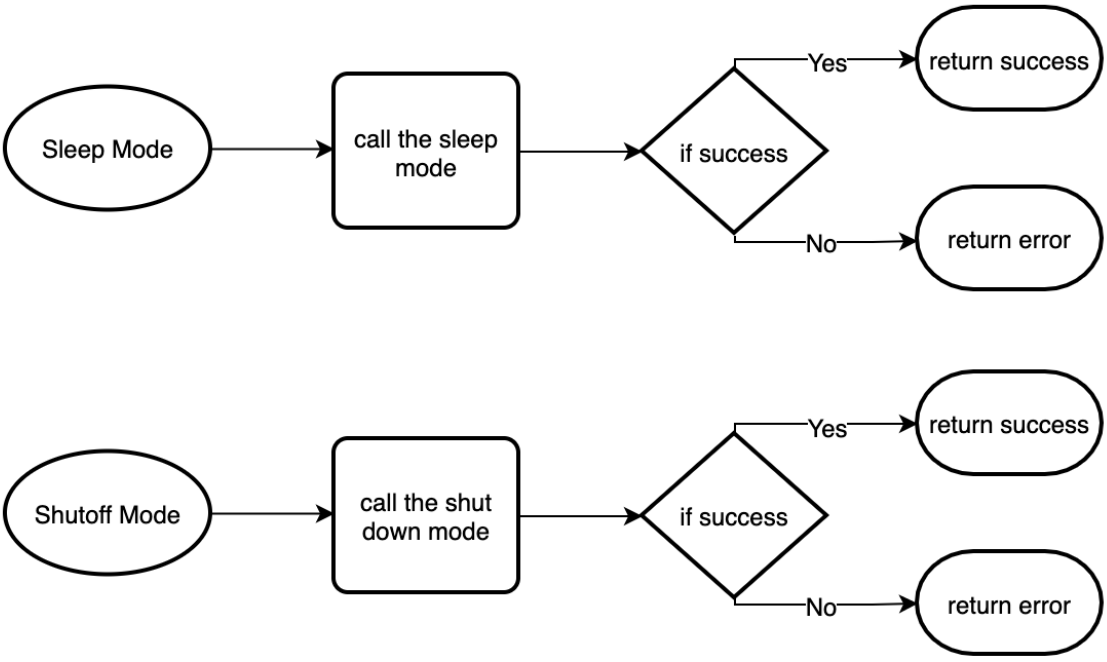

Figure 17 Sleep mode and shutoff mode.

The mode transition function has the following responsibilities:

- Initialize power manager .
- Start sleeping when the board has nothing left to do.

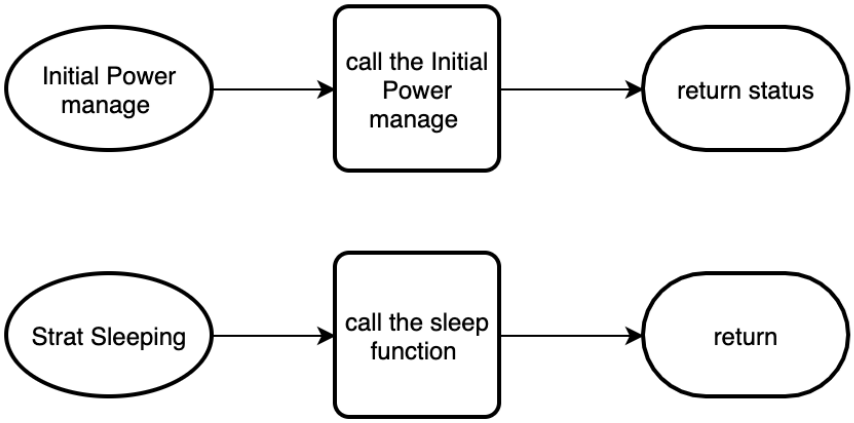

Figure 18. Functions initializing the power manager and for putting the board to sleep.

4.2. Desktop Application Python Library Component

4.2.1. Data/Board Management

The Data/Board Management module will be responsible for handling data that is exchanged between the sensor and the desktop application. This module will largely be communicating with the Communication module on the front-end. It will also communicate with the MATLAB GUI in order to play incoming audio from Streaming Mode. Responsibilities of the module include:

- Capture the audio data sent from the sensor and received from the Communication Module.
- Reassemble the audio segments received so it can be played on the GUI. This is specifically for Streaming Mode.
- Relay mode changes to the Sensor. These modes include: Time-Interval Mode, Triggered Mode, Power-Down Mode, Idle Mode, and Streaming Mode.
- Delete files from the sensor.
- Download files from the sensor.
- Communicate changes in sensor settings.

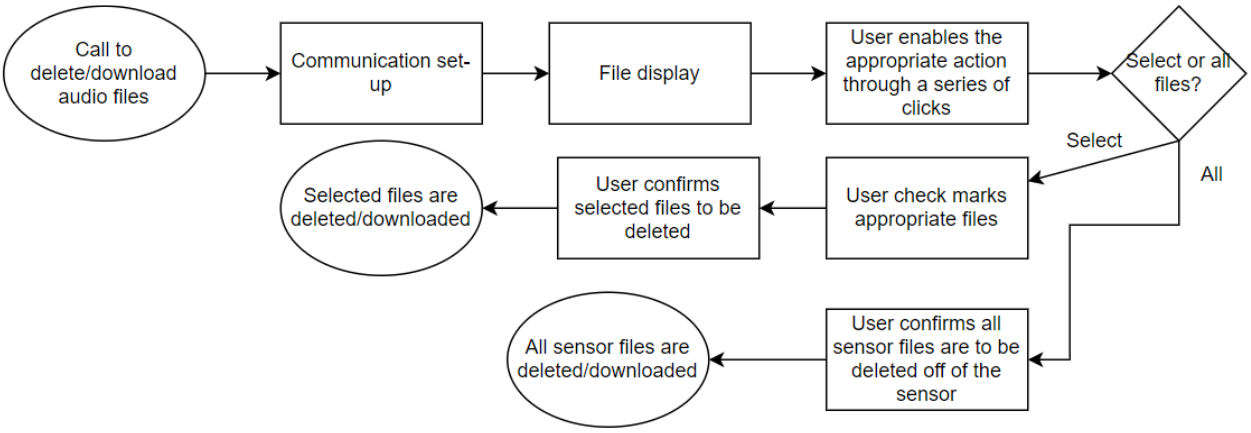

Figure 19. There will be two functions that take care of downloading audio files and deleting audio files. Both have the option of select/select all.

There will be two separate functions for deleting and downloading audio files that operate similarly. The appropriate function is called when the user makes their selection.

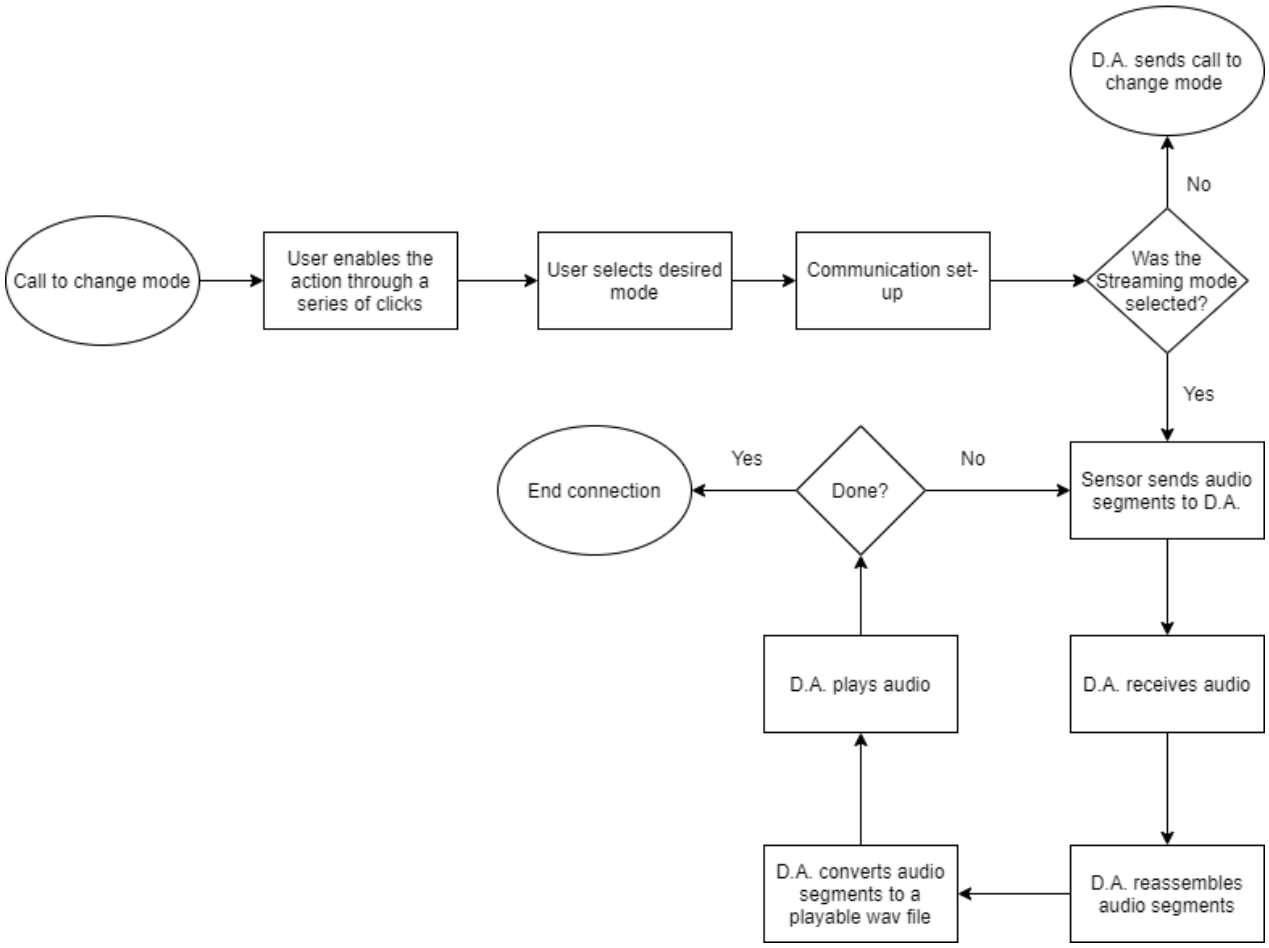

Figure 20. There will be a function in charge of relaying mode change data to the sensor via communication module. The handling of audio data will be in a separate function. The conversion into a wav file will also be a separate function.

To capture audio and assemble that audio into files, the module will be going back and forth with the Communication Module. There will be a check to see if any more audio is incoming. If there is, a function will get the audio from the Communication Module. Another function will turn that audio into a wav file. The file will be handed over to the GUI to play. There are MATLAB functions in place that will read the wav file and play it. If there is no more incoming audio, the GUI will cease its operation of playing audio. To change modes, the module will simply let the sensor know to commence the operation of the specified mode at that end via the Communication module. This is part of the master/slave relationship between the two components. Streaming Mode is different from the other modes because it requires more data management, which is why it isn't a simple call to the sensor to implement the mode.

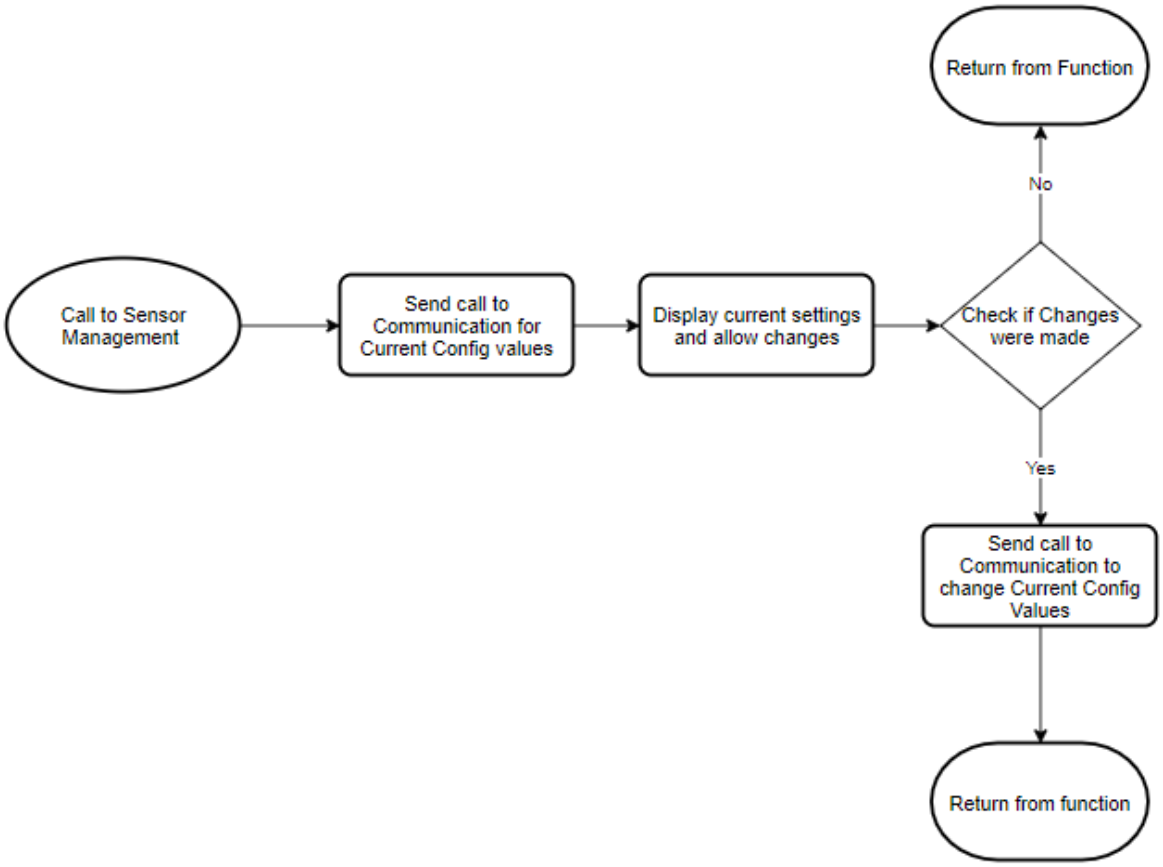

Figure 21. There will be a function in charge of fetching current setting configurations. Another function will be in charge of relaying changes in setting configurations to the sensor via communication module.

To fetch current settings, a function will make a call to the communication module to do so. To change sensor settings, a function will deliver the specified changes to the Communication Module.

4.2.2. Communication (General)

The communication module is in charge of transmitting data back and forth between the data management module and the USB communication module on the front-end. The purpose of this front-end communication module is to implement the communication schema as the master in the master-slave relationship. It is abstracted away from the specific communication medium being used. This is to allow future users to incorporate other medium-specific communication such as Bluetooth or WiFi without needing to rewrite major parts of the

communication schema to work with it. Below are the responsibilities of this module:

- Send commands from the desktop application to the back-end
- Accept incoming data from the back-end
- Handle connection and disconnection handshakes to properly handle connections and disconnections.

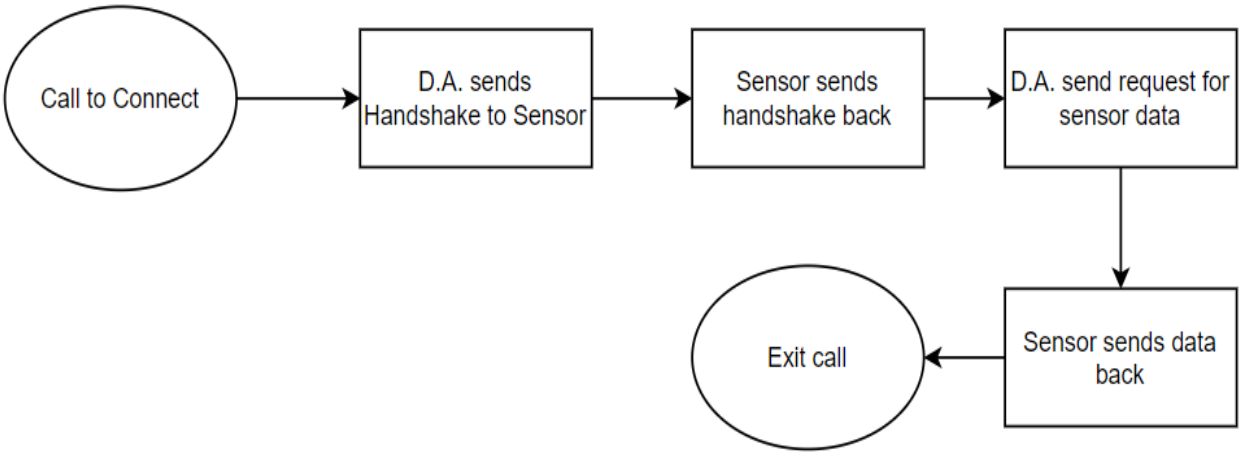

Figure 22. A function will take care of the handshake portion of communication. Another will be handling data received by the sensor.

4.2.3. USB Communication

The USB Communication module in the Python library component is used to communicate with the USB Communication module in the sensor component. The purpose of the USB Communication module is to provide the general communication module the USB-specific communication functionalities needed. This is to abstract the medium-specific protocols from the practical implementation of the communication schema so the communication medium can be changed without needing to rewrite most of the general communication aspects.

This USB Communication module has the following responsibilities:

- Accept incoming data over the USB connection from the sensor.
- Send outgoing commands and data over the USB connection from the desktop application to the sensor.
- Return to the General Communication module when a message has been fully received.

There are three main functionalities of the USB Communication module, a function to initialize the USB communication, a function to send a command, and a function to receive a command.

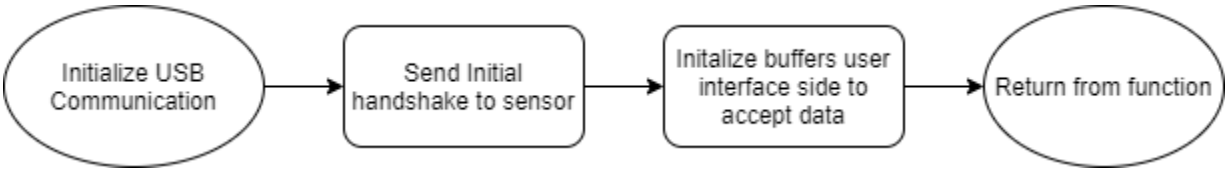

Figure 23. This diagram shows the basic USB Communication Initialization.

Figure 23 shows the steps to initialize a communication with the sensor from the user interface. First it would have to send the initial handshake to be able to send and receive data.

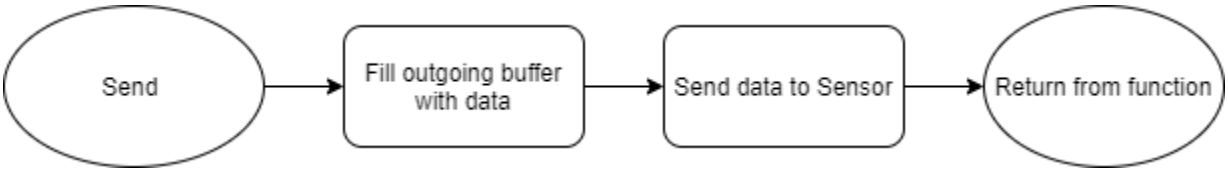

Figure 24. This diagram shows sending data and commands to the sensor.

Figure 24 shows the steps to send a command to the sensor through this USB connection. This will fill a buffer and send data to the sensor and return.

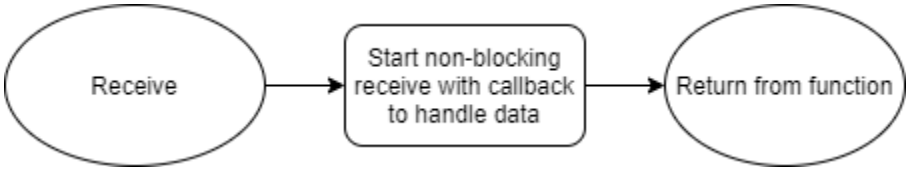

Figure 25. This diagram shows the user interface receiving data from the sensor.

Figure 25 shows the steps to receive a command from the sensor through the USB connection. This simply allows a command to come through and sends the data through a callback to another function to handle it with ease.

4.3. Desktop Application Matlab GUI Component

4.3.1. Import/Export

This would handle any exporting of data from the board to any place on the users computer. This is called when you hit the export button on the User Interface. The purpose of this module is to have an easy method of exporting files so that the User can analyze this on their own or to keep the files for further use.

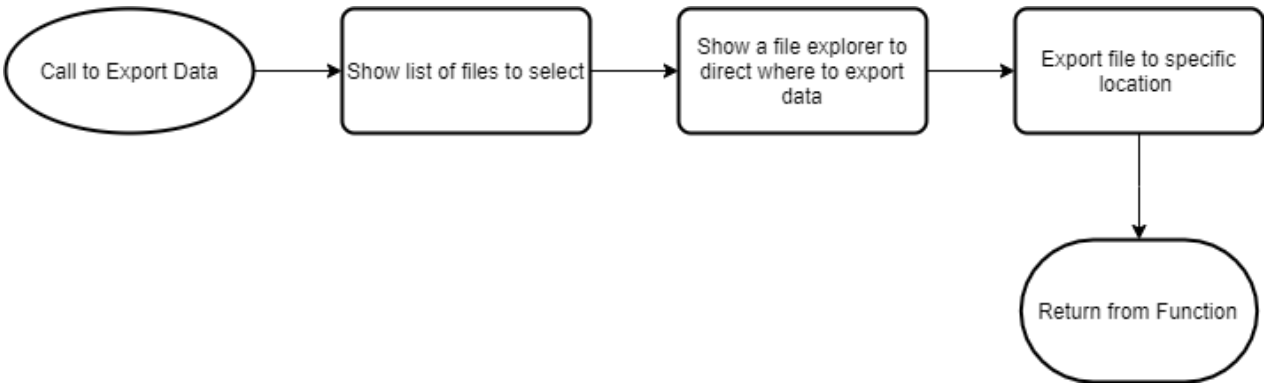

Figure 26. Diagram showing the path that import export will take in order to ensure that you select where a user would like to save these files.

Some of the data required and saved by this module is listed below:

- User selected files to export onto a selected path.
- User selected path to export the user selected files to.

The main functionality of this module will be exporting data from the sensor to a users path that requires it to only call from the user interface meaning no communication needed at this point.

- ExportFiles(files,path) - Export the selected files to the path provided by the user return 0 if successful.
- SelectPath() - Select the path that the user would like to export the files to and return that list.
- SelectFiles() - Select the files that a user would like to export and return that list of files.

4.3.2. Data Visualization

The Data Visualization module does not have the Communication module as a middleman. This is because its responsibilities do not rely on exchanges between the sensor and the desktop application. The purpose of this module is to display audio file information. Below are the responsibilities of this module:

- Filter through the audio segment files based on sensor ID, time, and date
- Allow the user to interact with the audio segment files.
- The module should allow the ability to display the audio file's respective spectrogram and metadata, which includes the sensor ID that recorded the audio, and time and date of recording and
- Allow the user to manipulate the audio segment file by playing, resuming, stopping, and pausing the audio.

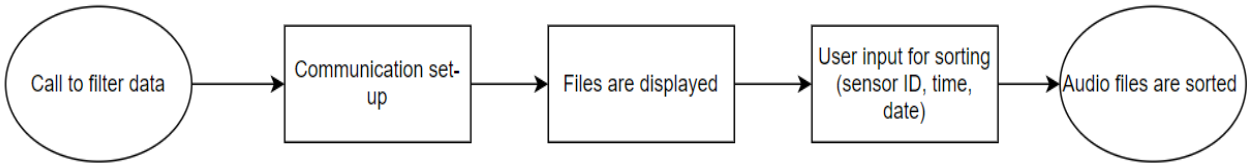

Figure 27. Function to sort audio data based on the user's selection.

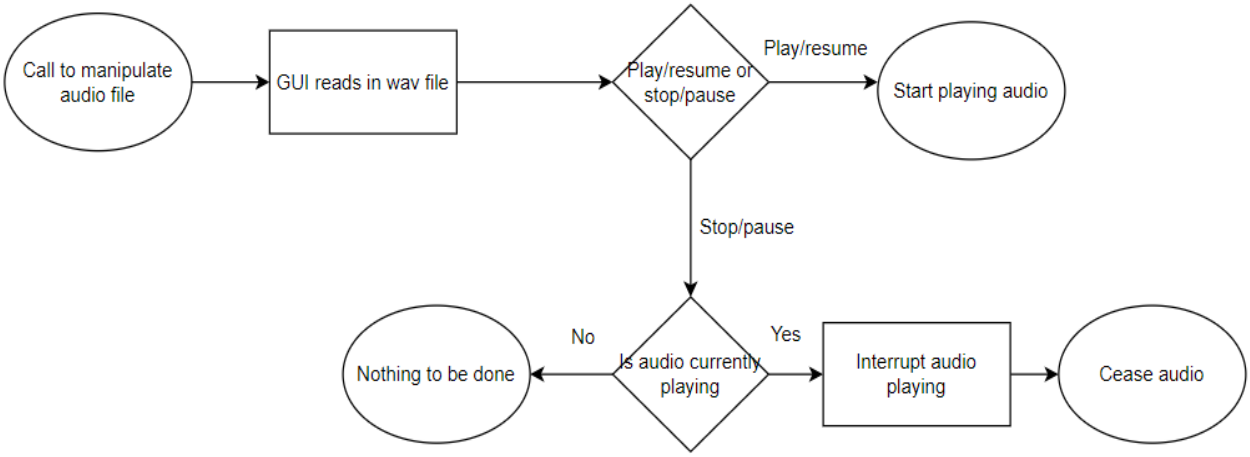

Figure 28. Function to start playing the audio file. Nonblocking and will have a callback function to cease playing audio if the user has chosen that.

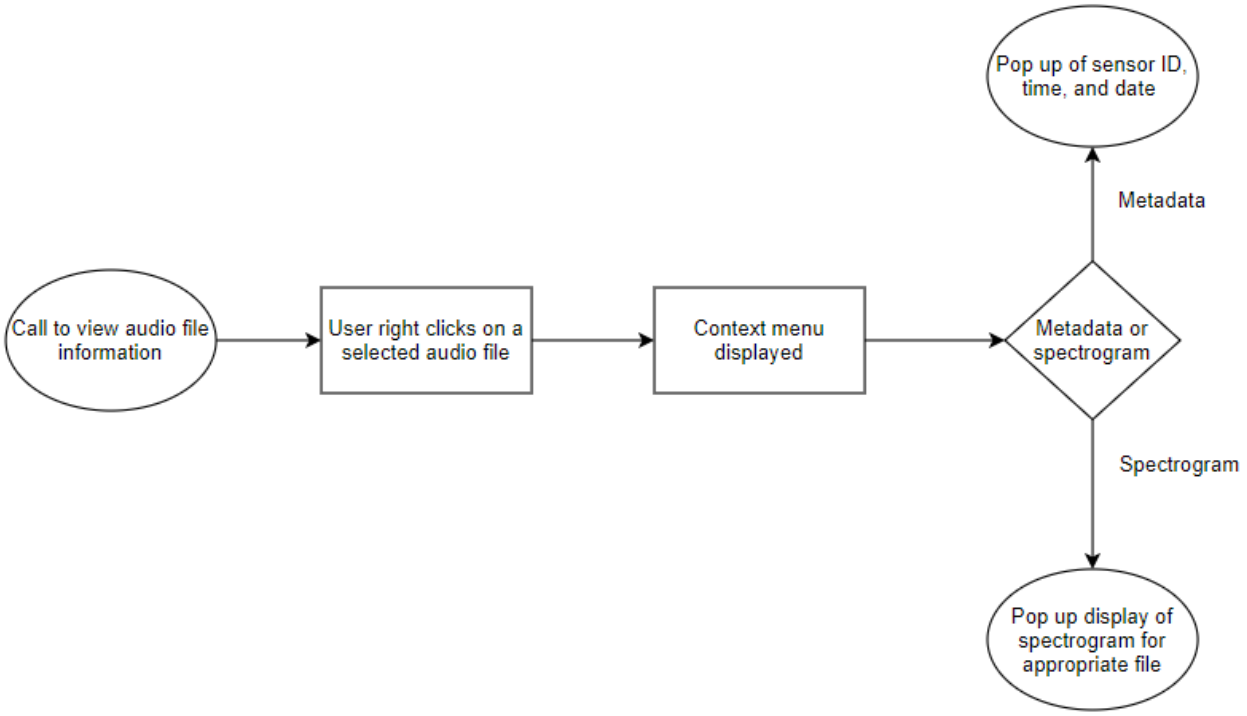

Figure 29. There are two functions that can be called depending on what selection was made on the context menu. Those two functions had the display of audio file metadata and spectrogram of the audio file.

The module discussed is for the most part MATLAB GUI specific. The user will navigate themselves through a series of clicks on MATLAB components (buttons, context menus, drop-down menus) in order to get to their desired action. After the user selects their preferred sorting method, whether they want to see the spectrogram or metadata of an audio file, or whether they want to play, resume, stop, or pause audio, then callback functions will be called after GUI components are interacted with. These callback functions will do the sorting based on the user's preference, manipulate audio, and display audio file information. There will be a callback function for every action possible. For the spectrogram specifically, there will be a function to generate one from an audio file.

5. Implementation Plan

We are working towards creating a working and usable product before the deadline. In order to ensure that outcome, we have set deadlines in place for every part of our project. Below we discuss the details of implementation and testing.

Our Gantt chart is composed of three main components: Document Schedule, Module Implementation, and Module Testing Schedule. We are expecting substantial parallelism in our work as the due dates for the work under the three main components will intertwine.

The Document schedule component serves as our schedule for completing deliverables and documents. These documents are listed according to their due date in ascending chronological order.

The Module Implementation component focuses on the twelve modules that make up our project as a whole. Those modules are Communication (Sensor), Communication (Desktop App), Microphones and Recording (Sensor), USB Communication (Sensor), Data Management (Desktop App), Audio Analysis (Sensor), File System (Sensor), Data Visualization (Desktop App), Energy Management (Sensor), Sensor Management (Desktop App), Import/Export (Desktop App), and Time and Date (Sensor). Our Team is divided into two main subgroups: Yasmin on the desktop application, Kevin and Anqi on the sensor, and Daniel who will be on the end that needs help at any point during the implementation of the modules. Each module is listed in the order of how we will be starting each one. In other words, the modules that are listed first from top to bottom are to have progress first before the modules that follow. As such, they are higher in priority in order to ensure the success of our product and to avoid major problems that may ensue. Due dates vary based on perceived difficulty.

The Module Testing Schedule component is lined up exactly like the Module Implementation component in terms of the order of modules. The testing of the first five modules starts when their implementation ends. The rest of the

modules start their testing roughly in the middle of implementation. Due dates for the end of testing vary based on the team's judgement of how long it would take to ensure the module is working properly. Modules perceived to be harder and to take longer to implement will get more testing than others that are not considered so.

There is one plan detail that is not visible on the chart. We did not include any integration of modules into our chart, but we will be doing so progressively throughout implementation when appropriate. For example, Communication on both the sensor side and the desktop application side are related to each other, and therefore, will be integrated with one another after successful implementation. They will be tested together as well as separately.

Task Name	Start Date	Estimated Completion Date	January				February				March				April			
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Document Schedule																		
Alpha Prototype	3-1-2021	3-15-2021																
Project Acceptance Demo	4-12-2021	4-23-2021																
Final (As Built) Report	4-18-2021	4-29-2021																
User Manual	4-26-2021	4-29-2021																
Final Project Delivery	4-26-2021	4-29-2021																
Module Implementation Schedule																		
Communication (Sensor)	1-25-2021	2-15-2021																
Communication (Desktop App)	1-25-2021	2-15-2021																
Microphones/Recording (Sensor)	1-25-2021	2-15-2021																
USB Communication (Sensor)	1-25-2021	2-15-2021																
Data Management (Desktop App)	1-25-2021	3-1-2021																
Audio Analysis (Sensor)	2-15-2021	3-15-2021																
File System (Sensor)	2-15-2021	3-15-2021																
Data Visualization (Desktop App)	2-15-2021	2-26-2021																
Sensor Management (Desktop App)	2-15-2021	2-26-2021																
Energy Management (Sensor)	3-1-2021	3-15-2021																
Import/Export (Desktop App)	3-1-2021	3-15-2021																
Time and Date (Sensor)	3-1-2021	3-15-2021																
Module Testing Schedule																		
Communication (Sensor)	2-1-2021	2-15-2021																
Communication (Desktop App)	2-1-2021	2-15-2021																
Microphones/Recording (Sensor)	2-1-2021	2-15-2021																
USB Communication (Sensor)	2-1-2021	2-15-2021																
Data Management (Dekstop App)	3-2-2021	3-16-2021																
Audio Analysis (Sensor)	2-20-2021	3-20-2021																
File System (Sensor)	2-15-2021	3-1-2021																
Data Visualization (Desktop App)	2-26-2021	3-1-2021																
Sensor Management (Desktop App)	2-15-2021	2-28-2021																
Energy Management (Sensor)	3-7-2021	3-20-2021																
Import/Export (Desktop App)	3-7-2021	3-20-2021																
Time and Date (Sensor)	3-7-2021	3-20-2021																

Figure 30. Gantt chart for our project implementation including documents schedule, module implementation schedule, and module testing schedule. The red line represents the current week.

6. Conclusion

Birds play a very important role in ecosystems around the world. Without birds, many plants and animals would not survive and ecosystems would fall apart. Consequently, it is imperative to learn more about the behaviors of birds and how human interactions are influencing them.

There are current limitations in place that hinder our study of bird behavior. For example, the machine learning methods that people are using are limited by a lack of data. In addition, audio recording devices on the market do not offer the extensibility , on site analysis, ease of use, or are open source and low cost as we would like.

Together with Dr. Flikkema, we proposed a structure for the functionality and collaboration of BiVo devices and user interfaces. The structure is an open source and fully documented project to allow users to add their own modules. The BiVo device will use the EFM32GG12 Thunderboard for main recording and audio analysis on site. The user interface will mix the Matlab application designer and Python to interact with the BiVo device to collect data from it.

BiVo's architecture is divided into two major categories: sensors responsible for recording, analyzing, saving or streaming audio of bird sounds, and desktop applications that allow users to interact with sensors and download collected audio data. Desktop applications can be further divided into graphical user interfaces in Matlab and python script routines that interface sensors with Matlab.

Preparing a plan for the future of this project is crucial to avoid potential risks and challenges. Our Gantt chart is composed of three main components: Document Schedule, Module Implementation, and Module Testing Schedule. We are expecting substantial parallelism in our work as the due dates for the work under the three main components will intertwine.

Moving forward we are excited to begin developing BiVo, we expect challenges in the future but are optimistic to overcome these challenges. Our next step is getting the streaming mode working to show an alpha prototype.

References

1. Silicon Labs Library API Documentation, https://docs.silabs.com/mcu/latest/efr32mg12/group-SLEEPTIMER.

Osprey on a limb in Flagstaff, AZ. Imlay, K. 2017.