Module Guide for Audio360

Team #6, Six Sense
Omar Alam
Sathurshan Arulmohan
Nirmal Chaudhari
Kalp Shah
Jay Sharma

November 12, 2025

1 Revision History

Date	Version	Notes
2025-11-13	1.0	Initial write-up

2 Reference Material

This section records information for easy reference.

2.1 Symbols, Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
Audio360	360 Audio analysis system on smart glasses
DAG	Directed Acyclic Graph
FR	Functional Requirement
M	Module
MG	Module Guide
NFR	Non-functional Requirement
OS	Operating System
R	Requirement
SC	Scientific Computing
SPI	Serial Peripheral Interface
SRS	Software Requirements Specification
UC	Unlikely Change
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

Table 1: Symbols, abbreviations and acronyms used in the MG document.

See SRS Documentation at Symbols, Abbreviations, and Acronyms for a complete table used in Audio360.

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team [1]. We advocate a decomposition based on the principle of information hiding [2]. This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by [1], as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed [1]. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format of the initial input data.

AC3: The method of generating simulated microphone array input for testing and proof of concept.

AC4: The specific hardware on which the software is running.
Rationale: There will be a benefit to using a hardware that is smaller (better integration with glasses), cheaper (reduce costs for user), and stronger compute (allows for further advanced audio processing), if one exists in the market.

AC5: The number of microphones in microphone array.
Rationale: Depending on the performance on audio classification and directional analysis features, more or less microphones may be required than what is specified NFR9.2.

AC6: The arrangement of the microphones relative to each other.
Rationale: The system may be deployed on devices other than glasses, such as a hat. As a result, the microphone arrangement can change.

AC7: The output display hardware. At the moment, it is Rokid Glasses [3], which projects visuals onto glasses lens. Display may be changed to a simpler screen.
Rationale: Due to Rokid Glasses availability and cost, it may be swapped with a cheaper and easily available display that can be integrated easily onto a glasses frame.

AC8: The number of classification of audio sources.
Rationale: NFR5.1 specifies at least 3 distinct audio classification of audio sources required. This can be changed to detect a single audio classification if detecting one source is more critical.

AC9: The sample rate that audio is sampled from the environment from the environment.
Rationale: NFR7.1 states to sample at 16 kHz as that is the minimum to analyze audible sounds to human. However, the scope can expand to detect sounds outside the

human audible range to notify users of other critical sounds that human would have never been able to process on their own.

AC10: The sample size of audio data to analyze each time step. Currently specified to be up to 4096 samples from NFR3.2.
Rationale: For higher quality of audio processing, an higher sample rate is required.

AC11: The visualization content on the output display.
Rationale: Stakeholders may require alternative information or a different presentation format that is more effective and appropriate for the display. This is related to NFR6.2

AC12: Traditional algorithms may be used over hardware acceleration.
Rationale: Hardware acceleration can consume more power than traditional software algorithms. The trade-off of slower processing over power consumption may be desirable.

AC13: Action when low confidence occurs during audio source classification.
Rationale: According to FR5.4, the system shall notify user when results from classification is low confidence. However, this may be disregarded if the accuracy of the classification module is near perfect.

AC14: The SD card logging method for diagnostic audio data.
Rationale: The SD card diagnostic method is useful for bulk data logging during testing. In a production system it is going to be removed since no debug logs will be captured.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The input devices for providing audio data from the environment are microphones.
Rationale: Cheapest and easiest method of getting real-time audio from the environment of the user.

UC2: The processor works as a closed system. That is no external devices other than the microphones and display can be connected, and all processing occurs on the processor (processing is not offloaded to another device).
Rationale: Adheres to user Goal 7, user safety and privacy, as well as FR9.1.

UC3: Microphones must always be synchronized at each sample.
Rationale: Microphone synchronization is a requirement for directional analysis, which a core feature. Relates to FR1.2:

UC4: The core features interact with hardware through an hardware abstraction layer.
Rationale: This allows the achieve hardware antricipated change without system redesign.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Microphone Input Module

M2: USART Communication Module

M3: SD Card Interface Module

M4: Microphone Diagnostic Module

M5: Hardware-Hiding Module

M6: Audio Generation Module

M7: DOA Processor Module

M8: Audio360 Engine Module

M9: Audio Sampling Module

M10: Audio Spectral Leakage Prevention Module

M11: Audio Normalizer Module

M12: Audio Anomaly Detection Module

M13: Fast Fourier Transform Module

M14: Logging Module

M15: Fault Manager Module

M16: Mel Filter Module

M17: Discret Cosine Transform Module

M18: Principle Component Analysis Module

M19: Linear Discriminant Analysis Module

M20: Classification Module

Level 1	**Level 2**
Hardware-Hiding Module	M1
	M2
	M3
	M4
Behaviour-Hiding Module	M8
Software Decision Module	M9
	M7
	M6
	M9
	M10
	M11
	M12
	M13
	M14
	M15
	M16
	M17
	M18
	M19
	M20

Table 2: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be

made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

1. NFR2.1 and FR1.4 specify that the system shall detect and propagate faults to the embedded firmware layer for appropriate handling. A software fault manager module, M15, is responsible for monitoring and reacting to software error states. In addition, the M4 module monitors the microphone array for hardware faults.

2. NFR3.2 specifies that the system shall support different audio sample sizes. Sample size directly affects both the speed and quality of audio analysis. Consequently, a dedicated module, M9, is responsible for managing this.

3. FR3.4 recommends the use of hardware acceleration to leverage hardware optimizations. However, there may be cases where traditional software algorithms perform more efficiently, such as when processing smaller datasets. Therefore, specific computational tasks are encapsulated within dedicated modules that determine whether to use hardware acceleration based on the input characteristics. This design consideration affects M10, M11, and M13.

4. FR3.5 requires the system to detect anomalies in the audio stream. To address this, a dedicated module, M12, is responsible for monitoring the audio data and identifying anomalies such as clipping or dropouts.

5. FR4.5 requires that the software be portable across different hardware platforms. This directly influenced the design decision to abstract the Audio360 Engine from the hardware layer. Inputs to the Audio360 Engine are maintained in a generic format, enabling deployment on various hardware targets, including simulation environments.

6. FR5.1, FR5.4 requires the frequency analysis component to classify audio sources. The classification will be done by performing Principle Component Analysis (PCA) on extracted features from the Mel-Frequency domain and performing Linear Discriminant Analysis (LDA) on components to predict the classification. Since this is a multi-step process, speciic computational tasks are encapsulated within de-coupled modules that will be called individualy. This design affects M16, M17, M18, M19 and M20.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by [1]. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by

the operating system or by standard programming language libraries. *Audio360* means the module will be implemented by the Audio360 software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules

7.1.1 Microphone Input Module M1

Secrets: The configuration and initialization of the hardware peripherals required to read data from the microphone array.

Services: Provides a driver interface to read audio sample data from the microphone array as a buffered stream.

Implemented By: Driver Layer

7.1.2 USART Communication Module M2

Secrets: The configuration and initialization of the hardware peripherals required for USART serial communication with external devices.

Services: Provides an interface for sending and receiving serial data using the USART protocol on the serial pins of the microcontroller.

Implemented By: Driver Layer

7.1.3 SD Card Interface Module M3

Secrets: The configuration and initialization of the SPI hardware peripherals and clocks required for SD card communication. This module also hides the SD card file system management and driver used to send commands to the SD card.

Services: Provides an interface for reading and writing files to an SD card module connected to the microcontroller via SPI.

Implemented By: Driver Layer

7.1.4 Microphone Diagnostic Module M4

Secrets: The internal logic used to perform diagnostic tests on the microphone array hardware using hardware ADC converters.

Services: Provides diagnostic state information about the microphone array hardware and its connections to the microcontroller.

Implemented By: Driver Layer

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Audio360 Engine Module (M8)

Secrets: The internal coordination logic that determines the execution order and timing of core audio processing features. Also, the mechanisms that manage data flow between hardware interfaces and the software modules.

Services: Orchestrates the overall audio processing by receiving raw input data and managing module communication.

Implemented By: Audio360

Type of Module: Class

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Audio Generation Module (M6)

Secrets: The algorithm for simulating room acoustics and generating synthetic microphone array data from a given audio source and position. This includes room response calculations and spatial audio propagation models.

Services: Provides simulated four-channel microphone array output for testing and proof-of-concept development. Takes as input an audio source signal and 3D position coordinates, and outputs four synchronized audio streams representing what each microphone in the array would capture given the room acoustics and source location.

Implemented By: Python using pyroomacoustics library

Type of Module: Library

Notes: This module is temporary and intended only for proof-of-concept development. It will be removed in the final product once physical hardware microphones are integrated. The module enables algorithm development and testing without requiring the complete hardware setup.

7.3.2 DOA Processor Module (M7)

Secrets: The specific direction of arrival estimation algorithm and its implementation details, including signal processing techniques (MUSIC, SRP-PHAT, or FRIDA), frequency analysis methods, and coordinate system transformations.

Services: Analyzes four synchronized microphone audio streams to estimate the direction of arrival of a sound source. Takes as input four time-domain or frequency-domain audio signals and outputs an angle (in degrees) representing the estimated direction of the sound source relative to the microphone array's reference axis. Provides real-time directional audio analysis with target accuracy of 45 degrees as specified in the SRS (NFR5.3).

Implemented By: Audio360 (Python for prototyping, C/C++ for embedded implementation)

Type of Module: Abstract Data Type

Notes: This module is core to the system's primary functionality of providing directional awareness. It implements one or more DOA algorithms and may include algorithm selection logic based on environmental conditions or computational constraints. Ideally, there should be no changes to this module for the final product aside from the internal implementation details (programming language for efficiency and combatilibility).

7.3.3 Audio Sampling Module (M9)

Secrets: The data structure used to store sampled audio data, the scheduling of sampling times, and the memory optimization strategy for acquiring and storing samples.

Services: Provides sampling of audio data from a given source while preserving the order of individual samples.

Implemented By: Audio360

7.3.4 Audio Spectral Leakage Prevention Module (M10)

Secrets: Windowing function strategy (Windowing is the method to minimize spectral leakage).

Services: Applying windowing on audio signal to reduce spectral leakage before frequency domain processing.

Implemented By: Audio360

7.3.5 Audio Normalizer Module (M11)

Secrets: The normalization algorithm.

Services: Processes audio signals to maintain consistent amplitude across different sources.

Implemented By: Audio360

7.3.6 Audio Anomaly Detection Module (M12)

Secrets: The detection algorithms used to identify anomalies in the audio stream. Also includes the list of dependent modules that must be notified upon specific anomaly occurrences.

Services: Detects anomalies in the audio signal, such as clipping, dropouts, or empty buffer conditions, and issues appropriate notifications.

Implemented By: Audio360

7.3.7 Fast Fourier Transform Module (M13)

Secrets: The specific algorithm to compute the fourier transform, potentially including hardware acceleraion provided by the STM32 platform.

Services: Computes the discrete Fourier transform of the input signal to obtain its frequency domain representation.

Implemented By: Audio360

7.3.8 Logging Module (M14)

Secrets: The method used to route character streams to output sources, and the internal log formatting strategy, including time-stamping and file source information.

Services: Provides centralized logging of data streams to designated output destinations for debugging and monitoring.

Implemented By: Audio360

7.3.9 Software Fault Manager Module (M15)

Secrets: The internal mechanisms and decision logic used to detect, manage, and recover from software system faults.

Services: Monitors system error states and manages critical faults to maintain core functionality when possible.

Implemented By: Audio360

7.3.10 Mel Filter Module (M16)

Secrets: The specific algorithm for converting a signal from the spectral domain to the mel spectral domain.

Services: Accepts a time-frequency domain signal and outputs the corresponding time-mel-frequency signal based on the desired number of mel-bands.

Implemented By: Audio360

7.3.11 Discrete Cosine Transform Module (M17)

Secrets: The specific algorithm for computing the Discrete Cosine Transform (DCT) from a signal in the mel spectral domain.

Services: Accepts a time-mel-frequency signal, and runs the algorithm that computes the DCT of the signal. This will output the Mel Frequency Ceptral Coefficients (MFCC) that will describe the main features of the input signal.

Implemented By: Audio360

7.3.12 Principle Component Analysis Module (M18)

Secrets: The specific algorithm for transforming the MFCC feature vector into a lower-dimensional space by extracting only the most significant variance. Also includes the projection matrix and mean feature matrix learned during offline training.

Services: Accepts MFCC feature vector and produces new representation that preserves only the most signficant variance from the original MFCC vector. This reduces the computational load for the LDA (M19) module.

Implemented By: Audio360

7.3.13 Linear Discriminant Analysis Module (M19)

Secrets: The specific algorithm for applying Linear Discriminant Analysis (LDA) to determine which class the sound most likely belongs to and the probability. Also includes the project matrix and class-specific weight and bias values learned during offline training.

Services: Accepts feature vector extracted from the PCA (M18) module and performs linear discriminant classification to output the classification label corresponding to the detected sound category.

Implemented By: Audio360

7.3.14 Classification Module (M20)

Secrets: The modules and the order to call them in to get the most probable classification using raw input audio. It also contains heuristics for determining when a classification is considered low-confidence.

Services: Accepts raw input audio, passes the audio through various modules to determine the most probable classification and it's confidence. Internally determines whether the classification is considered low confidence and outputs the classification with a flag if the confidence is low.

Implemented By: Audio360

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

8.1 Modules to Requirements Traceability

The following table maps each module to the functional and non-functional requirements from the SRS that it helps satisfy. Requirements are referenced using the labels defined in the SRS document.

Req.	Modules
FR1.4	M15, M4
NFR1.1	M9
FR2.3	M15
NFR2.1	M15
FR3.1	M13
FR3.2	M11
FR3.3	M10
FR3.4	M10, M11, M13
FR3.5	M12
NFR3.1	M13
NFR3.2	M9
FR4.1	M8
FR4.2	M8
FR4.3	M8
FR4.4	M8, M15
NFR4.1	M9
NFR4.2	M9
NFR4.3	M9
FR5.1	M16, M17, M18, M19, M20
FR5.4	M20
FR7.1	M1, M9

Table 3: Trace Between Requirements and Modules

Module	Requirements
M6	Testing support for FR1.2, FR5.2, FR7.1, FR7.2, NFR7.1
M7	FR5.2, FR5.3, NFR5.3

Table 4: Trace Between Modules and SRS Requirements

Explanation of Module-Requirement Mappings:

- **Audio Generation Module (M6):** This module is used exclusively for testing and proof-of-concept development. It enables validation of requirements related to microphone synchronization (FR1.2), direction of arrival estimation (FR5.2), and audio

capture specifications (FR7.1, FR7.2, NFR7.1) without requiring complete hardware integration.

- **DOA Processor Module (M7):** This module directly implements the core direction of arrival functionality specified in the SRS:
  - FR5.2 - Direction of arrival estimation from frequency domain audio
  - FR5.3 - Angular representation of sound source direction
  - NFR5.3 - Maximum 45° error in direction estimation accuracy

8.2 Modules to Anticipated Changes Traceability

The following table maps anticipated changes to the modules that would need to be modified if those changes occur. This supports the principle of information hiding by showing which design decisions are localized to specific modules.

AC	Modules
AC3	M6
AC4	M5
AC5	-
AC6	-
AC7	M??
AC8	-
AC9	M9
AC10	M9
AC11	-
AC12	M13
AC13	M8

Table 5: Trace Between Anticipated Changes and Modules

Explanation of Anticipated Change Mappings:

- **AC3 - Audio generation method:** Changes to the simulation approach, room acoustics model, or testing methodology would only affect the Audio Generation Module. Since this module is temporary and isolated from the production system, it can be modified or removed without impacting other modules. The final product will replace this module entirely with hardware microphone drivers.

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. [4] said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

9.1 Use Relationships for Audio Generation and DOA Processor Modules

The following describes the use relationships for the two modules being documented:

9.1.1 Audio Generation Module (M6) Dependencies

Modules Used By Audio Generation Module:

- **Hardware-Hiding Module (M5):** The Audio Generation Module uses the Hardware-Hiding Module to access file I/O services for reading source audio files and writing generated microphone array outputs. This dependency ensures the module can operate across different platforms without modification.
- **Python Standard Libraries:** For numerical computation (NumPy), audio processing (SciPy), and room acoustics simulation (pyroomacoustics). These are external dependencies that provide the mathematical foundations for the simulation.

Modules That Use Audio Generation Module:

- **DOA Processor Module (M7):** During testing and proof-of-concept development, the DOA Processor uses the Audio Generation Module to obtain synthetic four-channel microphone data for algorithm validation.
- **Testing Framework:** Automated tests use this module to generate controlled audio scenarios with known ground truth for validation of the entire signal processing pipeline.

9.1.2 DOA Processor Module (M7) Dependencies

Modules Used By DOA Processor Module:

- **Audio Filtering Module:** The DOA Processor uses the Audio Filtering Module to obtain frequency-domain representations of the four microphone signals. The filtering module provides FFT-transformed, normalized, and noise-filtered audio data that is suitable for DOA algorithms.
- **Audio Generation Module:** The DOA Processor uses the Audio Generation Module to obtain the four microphone signals. The audio generation module provides the four microphone signals that are suitable for DOA algorithms. Note, as mentioned previosuly, this module dependency is expected to be removed for the final product.
- **Mathematical Libraries:** Uses linear algebra and signal processing libraries for matrix operations, FFT/IFFT, and statistical computations required by DOA algorithms.

Modules That Use DOA Processor Module:

- **Visualization Module:** Uses the DOA Processor's output (direction angle in degrees) to generate visual indicators on the display showing the user where sound sources are located.

Figure 1: Use hierarchy among modules

10 User Interfaces

[[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS] —SS]

11 Design of Communication Protocols

[If appropriate —SS]

12 Timeline

[Schedule of tasks and who is responsible —SS]
[You can point to GitHub if this information is included there —SS]

References

[1] D. Parnas, P. Clement, and D. M. Weiss, "The modular structure of complex systems," in *International Conference on Software Engineering*, 1984, pp. 408–419.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Comm. ACM*, vol. 15, no. 2, pp. 1053–1058, Dec. 1972.

[3] Rokid, "Rokid glasses," 2025. [Online]. Available: https://global.rokid.com/?srsltid=AfmBOoo-Mc8pQf3WkW2WnUC0L6rIWmC9SK6KKv4U_6GXD-AeR2IXig8y

[4] D. L. Parnas, "Designing software for ease of extension and contraction," in *ICSE '78: Proceedings of the 3rd international conference on Software engineering*. Piscataway, NJ, USA: IEEE Press, 1978, pp. 264–277.