Module Guide for Audio360

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

November 9, 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 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
F	Functional Requirement
M	Module
MG	Module Guide
NFR	Non-functional Requirement
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Audio360	360 Audio analysis system on smart glasses
UC	Unlikely Change

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.
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.

AC2: 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.

AC3: The arrangement of the microphones relative to each other.
Rationale: The system may be deployed on other survaces other than glasses, such as a hat. As a result, the microphone arrangement can change.

AC4: 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.

AC5: The number of classification of audio sources.
Rationale: NFR5.1 specifies atleast 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.

AC6: 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.

AC7: 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.

AC8: 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

AC9: Traditional algorithms may be used over hardware acceleration.
Rationale: Hardware acceleration can consume more power than tradional software algorithms. The trade-off of slower processing over power consumption may be desirable.

AC10: Action when low confience 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.

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 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Audio360 Engine Module

M3: Audio Sampling Module

M4: Audio Spectral Leakage Prevention Module

M5: Audio Normalizer Module

M6: Audio Anomoly Detection Module

M7: Fast Fourier Transform Module

M8: Logging Module

M9: Fault Manager Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	M2 ? ? ? ? ? ? ?
Software Decision Module	M3 M4 M5 M6 M7 M8 M9

Table 1: 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 2.

[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. 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, M3, is responsible for managing this.

2. 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 M4, M5, and M7.

3. 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.

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 (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

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 (M2)

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 Sampling Module (M3)

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

7.3.2 Audio Spectral Leakage Prevention Module (M4)

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

7.3.3 Audio Normalizer Module (M5)

Secrets: The normalization algorithm.

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

Implemented By: Audio360

7.3.4 Audio Anomoly Detection Module (M6)

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

7.3.5 Fast Fourier Transform Module (M7)

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

7.3.6 Logging Module (M8)

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

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

Implemented By: Audio360

7.3.7 Fault Manager Module (M9)

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

Services: Monitors system health and manages critical faults to maintain core functionality.

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.

Req.	**Modules**
FR1.4	M9
NFR1.1	M3
FR2.3	M9
NFR2.1	M9
FR3.1	M7
FR3.2	M5
FR3.3	M4
FR3.4	M4, M5, M7
FR3.5	M6
NFR3.1	M7
NFR3.2	M3
FR4.1	M2
FR4.2	M2
FR4.3	M2
FR4.4	M2, M9
NFR4.1	M3
NFR4.2	M3
NFR4.3	M3

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	-
AC3	-
AC4	M??
AC5	-
AC6	M3
AC7	M3
AC8	-
AC9	M7
AC10	M2

Table 3: Trace Between Anticipated Changes and Modules

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]

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]

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.