UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI SCIENZE E METODI DELL'INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Meccatronica

AN EMBEDDED SINE-SWEEP MEASUREMENT SYSTEM FOR ACOUSTIC MUSICAL INSTRUMENTS ANALYSIS BASED ON MICROCONTROLLER AND BLUETOOTH INTERFACE

Supervisor:
Prof. Ing. Fabrizio Pancaldi

Co-Supervisor:
Dr. Ludovico Ausiello

Candidate:
Andrea Catellani

Academic Year 2024 - 2025

Reggio Emilia, 9th April 2026

CONTENTS

ABSTRACT

This thesis presents the work carried out by the author during his master's degree curricular internship at the School of Electrical and Mechanical Engineering of the University of Portsmouth, United Kingdom.

The project focused on the design of a microcontroller-based embedded system for measuring the frequency response of acoustic musical instruments using the Exponential Sine Sweep (ESS) technique developed by Prof. Angelo Farina. The device is conceived as a stand-alone device for the dynamic frequency-domain characterization of resonant mechanical systems, with reference to the vibro-acoustic structures of musical instruments such as soundboards.

The main objective of the work was to design a compact and autonomous device capable of replacing the conventional measurement setup currently used for these tests, which typically requires the combined use of multiple laboratory instruments and dedicated software tools. By integrating signal-generation, response-acquisition, and data-processing functionalities within a single embedded platform, the proposed solution aims to simplify the experimental procedure while improving system portability and usability.

During this work, the complete system was designed at architectural, hardware, and firmware levels. However, the experimental implementation focused on the exponential Sine-Sweep signal generation stage and on the Bluetooth communication subsystem, which enables interaction with a dedicated smartphone application for device control and monitoring. The output signal generation chain was experimentally validated according to reference criteria, demonstrating performance comparable to that of the reference system. The remaining functionalities were defined at the design level and represent the basis for future developments toward a fully integrated portable measurement instrument.

ITALIAN ABSTRACT

Questa tesi presenta il lavoro svolto dall'autore durante il tirocinio curriculare della laurea magistrale in Ingegneria Meccatronica presso la School of Electrical and Mechanical Engineering della University of Portsmouth, nel Regno Unito.

Il progetto si è concentrato sulla progettazione di un sistema embedded basato su microcontrollore per la misura della risposta in frequenza di strumenti musicali acustici mediante la tecnica dell'Exponential Sine Sweep (ESS) sviluppata dal Prof. Angelo Farina. Il dispositivo è concepito come uno strumento stand-alone per la caratterizzazione dinamica nel dominio della frequenza di sistemi meccanici risonanti, con particolare riferimento alle strutture vibro-acustiche degli strumenti musicali, come le tavole armoniche.

L'obiettivo del lavoro è stato progettare un dispositivo embedded compatto e autonomo in grado di sostituire l'attuale configurazione di misura utilizzata per questi test, che richiede l'impiego combinato di più apparecchiature di laboratorio e software dedicati. Integrando in un'unica piattaforma embedded le funzionalità di generazione del segnale, acquisizione della risposta ed elaborazione dei dati, la soluzione proposta mira a semplificare la procedura sperimentale migliorando al contempo la portabilità e l'usabilità del sistema.

Nel corso di questo lavoro, l'intero sistema è stato progettato a livello architetturale, hardware e firmware. Tuttavia, l'implementazione sperimentale si è concentrata sulla fase di generazione del segnale Sine Sweep di tipo esponenziale e sul sottosistema di comunicazione Bluetooth, che consente l'interazione con un'applicazione dedicata per smartphone per il controllo e il monitoraggio del dispositivo. La catena di generazione del segnale in uscita è stata validata sperimentalmente secondo criteri di riferimento, dimostrando prestazioni comparabili a quelle del sistema di riferimento. Le restanti funzionalità sono state solo definite a livello progettuale e costituiscono la base per futuri sviluppi verso la realizzazione di uno strumento di misura portatile completamente integrato.

1 INTRODUCTION

The dynamic characterization of soundboards and more generally acoustic musical instruments represent a research field that is still not fully explored. It is of fundamental importance for understanding vibro-acoustic behaviour, timbral quality, and the industrial repeatability of acoustic instruments [1]. The sound produced by such instruments intrinsically depends on the ability of the soundboard to act as an acoustic radiator, converting the mechanical energy of the strings into sound waves through complex resonance phenomena, including Helmholtz resonance and the structural vibrational modes of the wooden body.

Traditionally, the measurement of frequency response in lutherie and acoustic research has relied on techniques such as the impact hammer method or the use of broadband excitation signals like Maximum Length Sequences (MLS) [2]. However, these approaches exhibit significant limitations: the impact hammer provides limited bandwidth, especially when applied to soft materials such as spruce, while MLS-based techniques are highly sensitive to system nonlinearities. To overcome these issues, the state of the art has progressively converged towards the Exponential Sine Sweep (ESS) technique, developed by Prof. Angelo Farina. This method, based on exciting the system with a sinusoidal signal whose frequency increases exponentially over time, enables an effective separation between the linear response of the system and harmonic distortion components, ensuring a high signal-to-noise ratio and improved robustness against jitter and clock misalignment.

Despite the effectiveness of the ESS technique, the conventional experimental set-up for performing measurements on acoustic instruments remains complex and poorly portable. It typically consists of a personal computer equipped with Digital Audio Workstation (DAW) software and Aurora plug-ins [3], a high-resolution external audio interface, a linear power amplifier, electrodynamic exciters, and calibrated microphones. This reliance on multiple laboratory-grade instruments and commercial software increases system complexity, limits portability, and introduces potential sources of interference related to operating systems and hardware integration.

This thesis aims to address these limitations by proposing the design and development of a dedicated embedded system for frequency response measurements.

The primary objective is the integration of all required functionalities: signal generation, response acquisition and data processing, into a single compact stand-alone and low-cost device.

The work has been carried out following a rigorous methodological approach, starting from the definition of system requirements and performance targets based on industrial and audio standards (AES17, IEC 61606). From a hardware perspective, the control unit is based on an ARM Cortex architecture implemented on the STM32L476RG microcontroller, selected for its computational capabilities in audio signal processing. The signal output chain includes a MAX9867 audio CODEC for 16-bit/48 kHz digital-to-analogue conversion and a MAX98306 Class-D power amplifier for driving electrodynamic exciters. The input chain features a custom-designed +48 V phantom power supply for condenser microphones and employs the same MAX9867 CODEC for microphone preamplification and 16-bit / 48 kHz analogue-to-digital conversion.

From a firmware standpoint, the system architecture has been designed in a modular fashion, adopting a state-machine-based approach to ensure efficient task scheduling and concurrent execution of control and processing operations. Among the implemented solutions, particular relevance is given to the use of a double-buffering strategy via DMA for real-time generation of the sine sweep signal, overcoming the memory limitations of the microcontroller. Furthermore, a Bluetooth Low Energy (BLE) communication subsystem, based on the BlueNRG-M0 chip, has been integrated to enable wireless interaction with a dedicated smartphone application for parameter configuration and system monitoring.

Although the overall system has been fully designed at the architectural level, the experimental implementation has primarily focused on the validation of the signal generation chain and the Bluetooth communication interface with smartphone. The acquisition and processing stages are left for future developments. The validation phase included a comparative analysis between the developed embedded prototype and a professional reference measurement system. Tests were conducted to evaluate signal-to-noise ratio (SNR), total harmonic distortion (THD), and jitter, as well as the quality of the power supply. Additionally, a practical measurement of the frequency response of an acoustic guitar was performed.

The obtained results demonstrate that the proposed device can generate ESS signals with a quality comparable to standard measurement systems, achieving a similar spectral response, albeit with performance limitations due to the use of lower-cost

components. These results provide a solid foundation for future developments aimed at achieving a fully integrated and portable measurement instrument.

In conclusion, this thesis documents the engineering design process of an accessible wide-band measurement ecosystem, demonstrating how modern embedded technologies can simplify complex experimental procedures in musical acoustics without compromising scientific accuracy.

2 LITERATURE REVIEW

This chapter will present and illustrate the existing literature about frequency response measure methods with focus on Exponential Sine-Sweep method and embedded devices.

2.1 Musical Acoustics

Musical acoustics studies how the physical properties of materials and the geometry of a structure influence sound generation and perception [4]. In string instruments such as guitars and pianos, the sound production mechanism is based on the sympathetic vibration of the soundboard as shows in Figure 2.1. Since strings have a very small surface area, they displace too little air to radiate sound efficiently; therefore, they transfer their energy to the soundboard through the bridge. The soundboard thus acts as an acoustic radiator, converting mechanical energy into audible sound waves.

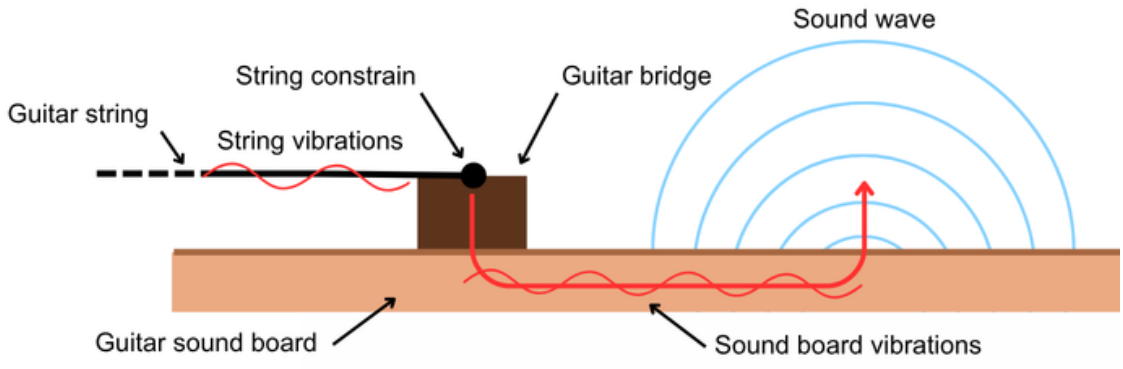

Figure 2.1. Depiction of soundboard resonance concept

Every structure exhibits specific modes of vibration, namely natural frequencies at which the system tends to oscillate when excited. The phenomenon of resonance occurs when an external force (such as the vibration of a string) excites the structure at a frequency that coincides with one of its natural frequencies, resulting in a significant increase in vibration amplitude. In acoustic instruments, the main resonances include the vibrational modes of the soundboard and the resonance of the air enclosed within the body, known as the Helmholtz resonance, which together define the overall vibrational behaviour of the instrument.

2.2 Frequency response of an acoustic instrument

Characterizing a musical instrument requires the analysis of its frequency response, which describes how the system reacts to different frequencies. It highlights peaks corresponding to resonances and valleys where energy is attenuated. This analysis is closely related to the concept of timbre, which can be defined as the perceptual attribute determined by the spectral content of a sound [4], i.e., the combination of harmonics with different amplitudes. Figure 2.2 shows the frequency response of an acoustic guitar.

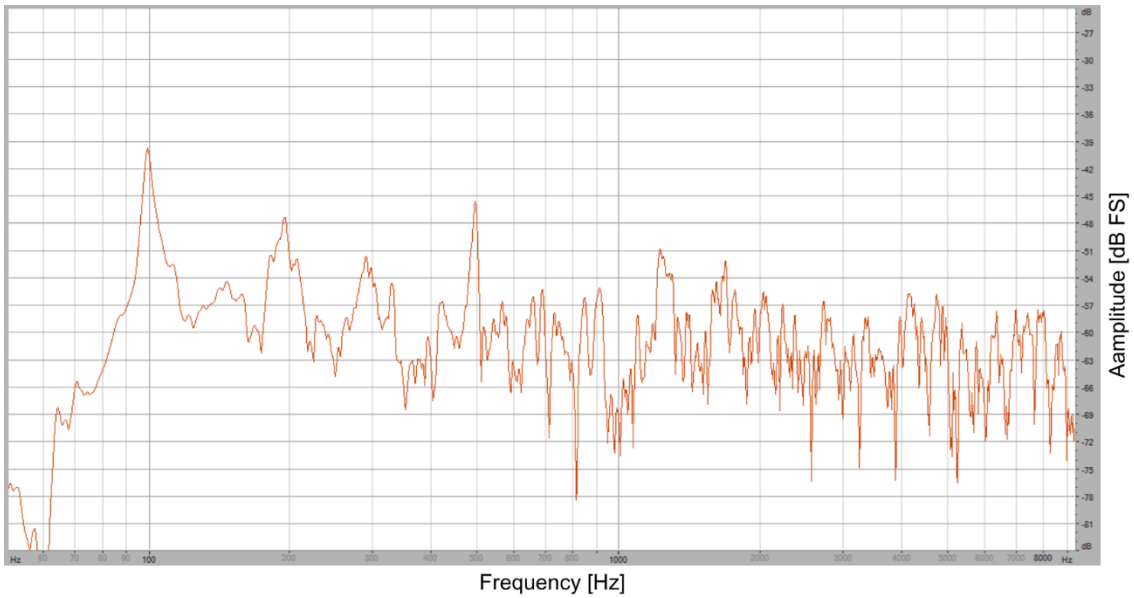

Figure 2.2. Plot with frequency response of an acoustic guitar

While the fundamental frequency determines the perceived pitch of a note, the presence and relative intensity of higher harmonics are the key factors that give an instrument its unique voice and tonal quality. Characterization is fundamental for several reasons:

- **Control of material variability:** Wood is an anisotropic material whose physical properties, such as density and stiffness, vary from piece to piece; measuring the acoustic response makes [4] it possible to compensate for these variations during production.
- **Industrial repeatability:** It allows manufacturers to define acoustic performance thresholds, ensuring that instruments labelled as “identical” exhibit consistent sound characteristics.

- **Design optimization:** It provides essential quantitative data for research and development (R&D), enabling the evaluation of the impact of structural modifications, such as changes in bracing patterns or the application of surface finishes, on the final sound quality.

2.3 Frequency response measure method

To measure the frequency response of a device, several techniques have been developed over the years. Traditionally this task was carried out using broadband signals such as Maximum Length Sequences (MLS) [5] and Time-Delay Spectrometry (TDS). In mechanical and lutherie contexts in particular, the impact hammer method was commonly employed. Although well established, these methods presented certain limitations. The impact hammer [6], for example, has limited bandwidth when used on soft materials such as spruce, while MLS and TDS are extremely sensitive to nonlinearities and time variations in the system.

To overcome these limitations, the state of the art has shifted toward the Exponential Sine-Sweep (ESS) [7] technique, which excites the system with a sinusoid whose frequency increases exponentially. The distinctive feature of this method is its ability to clearly separate the system's linear response from the products of harmonic distortion.

2.3.1 Exponential Sine Sweep method

The modern technique for measuring impulse responses (IR) has undergone a radical transformation with the introduction of the Exponential Sine-Sweep (ESS) method [7] [8], presented by Prof. Angelo Farina.

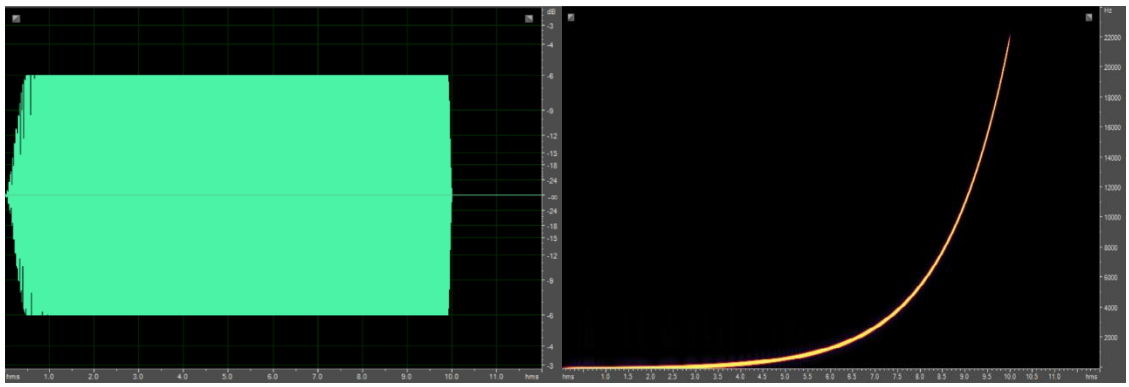

Figure 2.3. Exponential Sine-Sweep signal, time behaviour and spectrogram

Farina's method employs a sinusoidal signal whose frequency varies exponentially over time as shows in Figure 2.3. The main advantage of this approach lies in its ability to simultaneously deconvolve the linear impulse response and the responses corresponding to individual orders of harmonic distortion. Through a linear convolution process with an "inverse filter" (obtained by time-reversing the original signal and applying an amplitude modulation of +3 dB per octave), the distortion products are shifted to the left of the linear response, becoming clearly separated along the time axis. Once the impulse response is extracted, the final frequency response is calculated by applying the Fast Fourier Transform (FFT).

This technique ensures an exceptional signal-to-noise ratio (SNR), often more than 60 dB higher than traditional impulsive methods and is extremely robust against jitter, clock misalignment, and random background noise.

2.4 Implemented method for musical instruments measurements

The application of Sine-Sweep techniques to lutherie and the production of musical instruments, such as soundboards, represents a recent evolution aimed at replacing empirical or less precise methods. Traditionally, luthiers assess the quality of a soundboard through manual "tapping" finger percussion, a subjective and non-repeatable method. A more scientific approach is the impact hammer method, which nevertheless suffers from bandwidth limitations, often not exceeding 2–3 kHz on soft woods such as spruce and poor resistance to ambient noise [6].

Research conducted by Ausiello et all [9] has shown that the Sine-Sweep method, coupled with small electrodynamic exciters, provides very high coherence over a bandwidth reaching up to 8 kHz, enabling an objective and repeatable characterization of the frequency responses of guitars. In Figure 2.4 it is possible to see current measurement set-up.

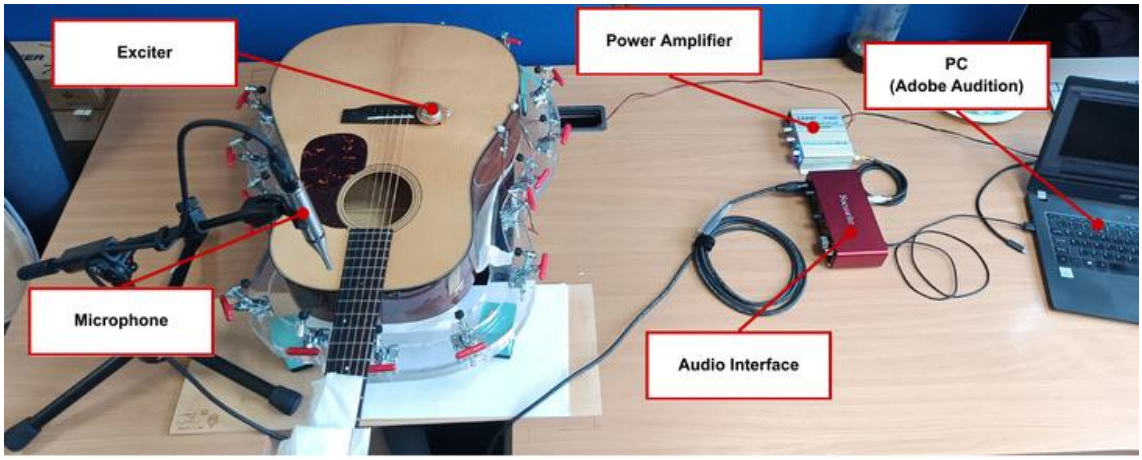

Figure 2.4. Image depicting current instrumentation for measuring the frequency response of an acoustic guitar.

An important aspect that emerged from his studies concerns the electromechanical interaction between the exciter and the soundboard. Placing the exciter at points of modal maxima, can alter the system's response causing the actuator to behave not as an inert mass but as a stiffening device. It has been demonstrated that measuring the electrical impedance [10] at the exciter terminals can be used to identify the first resonance modes of the soundboard without the need for external sensors, thereby simplifying the measurement process.

2.4.1 Current hardware set-up

The current state of art provides for the use of a measuring chain shown in Figure 2.4 composed of:

- A PC with Digital Audio Workstation (DAW) software, in this case is uses Audition and dedicated Aurora plug-in [3].
- A high quality (24-bit) external audio interface.
- A calibrated linear power amplifier.
- A mechanical actuator/exciter applied to the instrument bridge or blank board.
- A measuring microphone or accelerometer to acquire the response.

What stands out is the need for different devices and the use of dedicated software and plug-ins. All of this leads to an increased complexity in the set-up. In Figure 2.5 is shown a block diagram that resumes current measurement set-up.

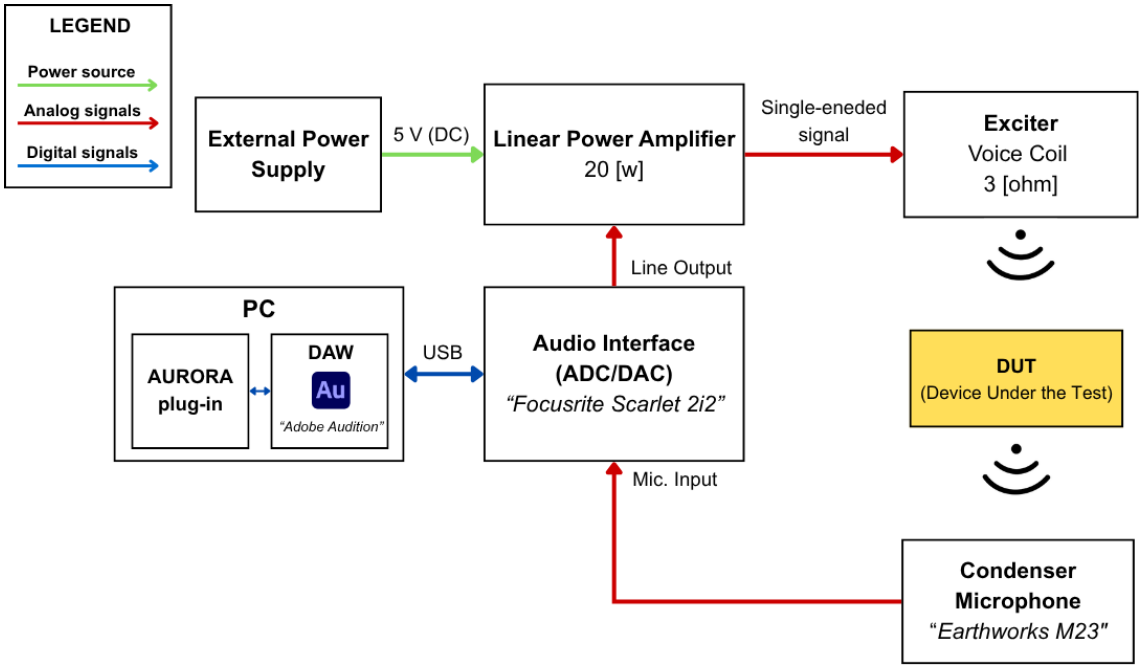

Figure 2.5. Block diagram of current measurement set-up

2.5 Embedded devices

In recent years, the evolution and implementation of increased computational power in microcontrollers chips have enabled the development of compact, high-performance embedded systems [11] in the audio field. The integration of analogue-to-digital conversion, computational capability, and advanced algorithms within a single embedded platform currently represents the state of the art for high-quality applications, such as audio signal processing and acoustic measurement.

A central element in any device operating with audio signals is the audio CODEC, which integrates both A/D and D/A conversion stages [12] [13] [14]. Modern solutions provide resolutions up to 24 or 32 Bit, with sampling frequencies exceeding 96 kHz, thereby ensuring high dynamic range and low quantization noise. In measurement applications, the quality of the analogue front-end (including preamplification, impedance matching, and anti-aliasing filtering) is as critical as the nominal resolution of the converter itself, since it determines the effective noise floor and the overall linearity of the system. These converters are typically based on Sigma-Delta technology.

From a computational perspective, the processing power of modern ARM Cortex-M microcontrollers equipped with floating-point units (FPU) enables processing of complex signals, including exponential sine sweeps (ESS), deconvolution procedures, and

harmonic analysis. Advances in embedded architectures have made it possible to implement complete processing pipelines directly on board, encompassing excitation signal generation, synchronous acquisition, and impulse response computation, thus reducing dependence on external host systems.

Spectral analysis based on the Fast Fourier Transform (FFT) constitutes a fundamental tool for extracting the frequency response. Contemporary implementations rely on optimized libraries, such as CMSIS-DSP in ARM-based environments [15], allowing efficient computation even on low-power devices. The FFT length and windowing strategy directly influence frequency resolution and spectral leakage, aspects that are particularly relevant in the modal characterization of musical instruments.

A frequently underestimated yet decisive aspect concerns power supply management. In high-sensitivity measurement systems, issues such as ground loops, capacitive coupling, and switching noise can significantly degrade performance. State of art solutions include isolated power supplies, low-noise voltage regulators (LDO), optimized PCB layout design, and the separation of analogue and digital grounds to minimize interference and disturbances.

Compared to PC-based systems, stand-alone devices offer several advantages, including portability, robustness, measurement repeatability, and reduced susceptibility to software or operating system interference. However, they also present certain limitations, such as restricted memory and computational resources, as well as increased complexity in hardware design.

The objective of this project is to fully exploit the possibilities offered by the current state of the art, to develop a high-resolution embedded device capable of integrating signal generation, synchronized acquisition, and advanced spectral analysis within a single compact unit, with particular emphasis on the quality of the analogue front-end and effective noise management.

3 DESIGN

This chapter presents the complete design process of the proposed device, covering all stages from requirements analysis to the final project definition.

In detail, the customer needs are first identified, leading to the definition of the system features and the corresponding performance targets, which are established according to current standards. Subsequently, the hardware and firmware architecture of the device is defined. These architectures are illustrated through block diagrams, which provide a clear representation of the system structure and the interaction between its main components.

Finally, the hardware components are designed in detail to meet the previously defined performance targets, together with the software logic responsible for controlling and managing the operation of the device.

3.1 Requirements analysis

The design process starts by observing the aim for our measurement system, hence collecting the needs customers might have. From this definition the needs that such a device will require have been extrapolated. Subsequently, the features were established, to which the performance targets that they will have to meet have been linked.

3.1.1 Purpose of the device

The goal of this work is to design a measurement system compliant with the requirements specified in [7] and [8] discussed in Chapter 2. The proposed solution aims to provide a more practical, portable, and cost-effective alternative to existing measurement setups. The system is designed to achieve performance comparable to the currently used solution while eliminating the need for a PC, dedicated software, and additional external hardware. By integrating all the necessary functionalities into a single device, the measurement process can be simplified and made more accessible.

To accomplish this, the signal input and output stages must be carefully designed to meet the required parameters and specifications, which will be discussed in the following

sections. Figure 3.1 shows conceptual architecture of the system, where the entire measurement chain is implemented within a single embedded device.

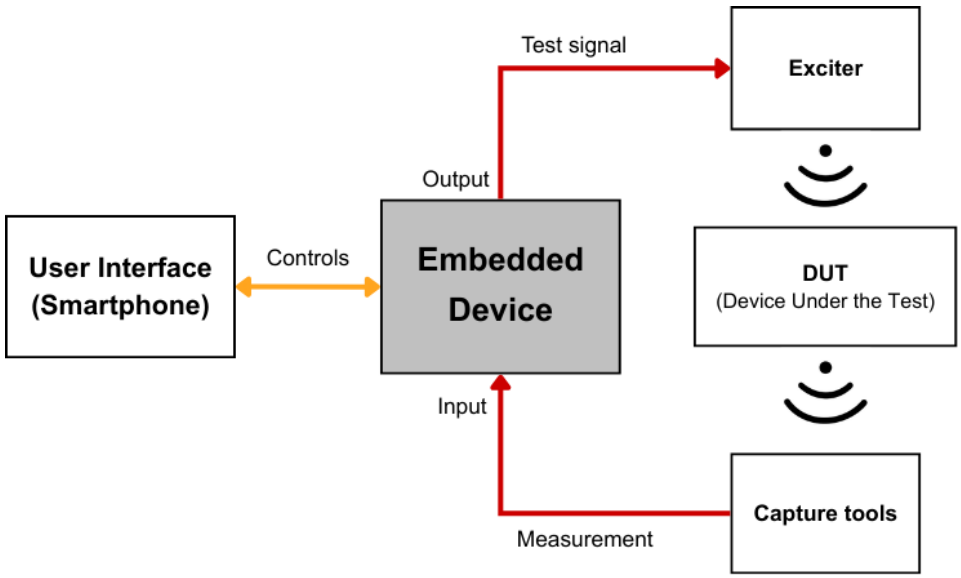

Figure 3.1. Block diagram of new system conceptual architecture

3.1.2 Needs

After defining the basic system architecture, the current measurement set-up was thoroughly analysed to define needs that the proposed device must satisfy.

To obtain a clearer overview and better organize the work, each need was classified according to three levels of priority: high, medium, and low. This classification is important for organizing the workflow and understanding which parts of the project should be given priority.

ID	Need	Priority
N1	Characterize resonant objects	High
N2	Repeatable measurements	High
N3	Accurate measurements	Medium
N4	Compact	Medium
N5	Smart	Low
N6	Low Cost	low

Table 3.1. Needs

As it can be seen from Table 3.1, the greatest emphasis was placed on the first need, as it represents the core objective of the project. Secondly, particular attention must be given to the repeatability of the measurements. Without such requirements, it would be impossible to perform validation tests on the device. Furthermore, the device would not be suitable for practical use, as users would be unable to reliably compare different measurement results.

3.1.3 System features

Once the needs have been defined and each of them has been given priority, the features that will satisfy the needs are defined.

Linked Needs	Features	Unity
N1	Frequency response	Hz
	CPU Load	%
	Storage Size	MB
N2	Digital Gain control	dB
N3	Differential Audio I/O	Vrms
	Resolution	Bit
	Sample rate	kHz
	Dynamic Range	dB
	THD	%
	Jitter	pSrms
	SNR	dB
	Power Supply Ripple	mV
	PSRR	dB
N4	Volume	dm^3
N5	Power consumption	W
N6	Low cost	€

Table 3.2. Features

Table 3.2 lists the features identified to address the defined needs. It is the result of a careful analysis of the state of the art in embedded systems and audio processing devices. Each feature was assigned a unit of measurement to allow its measurement and evaluation.

3.1.4 System requirements

This section defines the performance criteria that the system must satisfy. Establishing these requirements is essential for designing a high-quality device and for ensuring that the proposed solution can be fairly compared with currently available systems. The objective is to match, or potentially improve, the performance of the existing measurement setup. Target specifications associated with the main system features are summarized in Table 3.3.

Feature	Measurement criteria	Target Value
Frequency response	ESS	20 - 20 000 Hz
CPU Load	IEC 61131-3	≤ 70 %
Storage Size	IEC 61508	≥ 8 MB
Digital Gain control	Measure Output	≤ 2 dB
Differential I/O Audio	IEC 60268	1 Vrms
Resolution	AES17	≥ 16 Bit
Sample rate	IEC 61606	≥ 48 kHz
Dynamic Range	AES17	≤ -90 dB
THD	AES17	≤ 0,02 %
Jitter	AES17	≤ 10 pSrms
SNR	AES17	≤ -90 dB
Power Supply Ripple	IEC 61204-3	≤ 40 mV
PSRR	IEC 61204-3	≤ -80 dB
Volume	Meter	≤ 8 dm^3
Power consumption	IEC 61204-3	≤ 24 W
Low cost	Bill of Material	≤ 300 €

Table 3.3. Specification List including measurement criteria and corresponding target performance.

The metrics are assessment criteria and measurement methodologies shared between the designer and the final customer. By agreeing on the measurements criteria of the features, customer and designer find a common ground to share the final performance effectively achieved by the object of the design.

3.2 Hardware architecture

The system requirements and features have been defined, along with the relevant metrics used to evaluate the actual performance with respect to the established design targets. Based on these considerations, a high-level system architecture can now be defined.

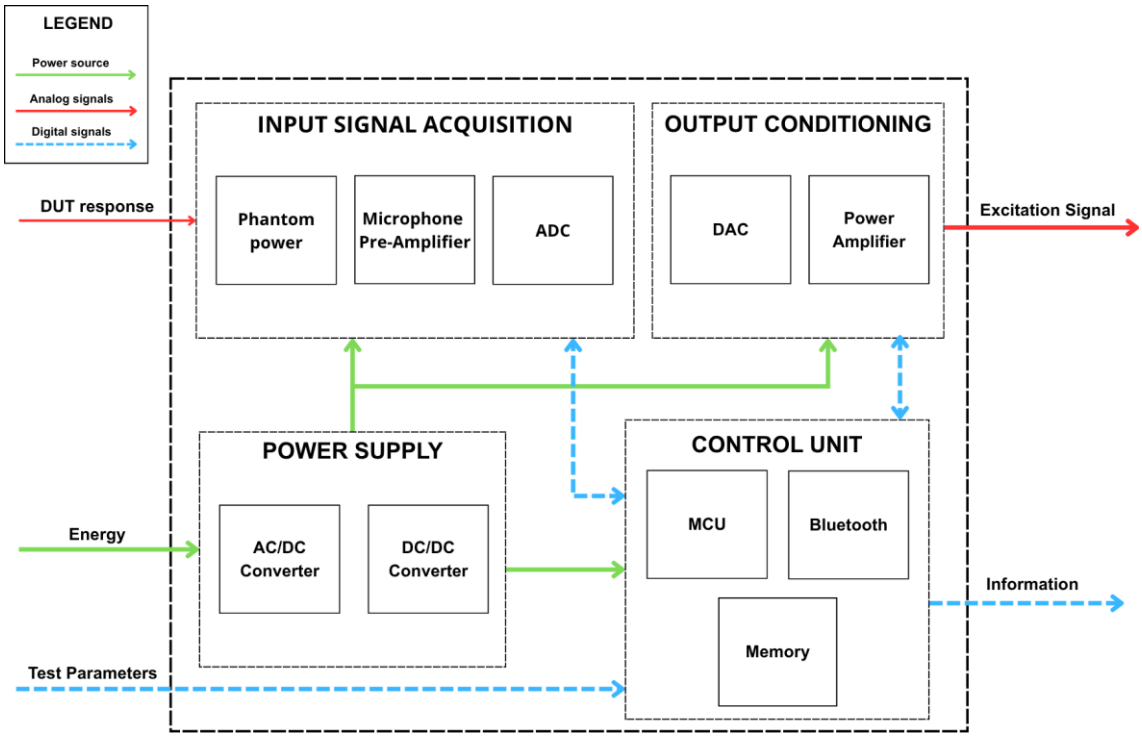

Figure 3.2. Hardware architecture block diagram of embedded device

Through a high-level representation such as the one shown in Figure 3.2, it is possible to define the main functional blocks of the system and their interactions, highlighting both the communication interfaces and signal flows.

On the output side, two signals are identified: the excitation signal used to stimulate the object under evaluation and the information provided to the user containing the test results. On the input side, three signals are defined: the response returned by the DUT (Device Under Test), which is processed to obtain the frequency response; the

parameters set by the user to configure the test; and the power supply required for device operation.

As illustrated in Figure 3.2, four main functional blocks are identified: **Output Conditioning**, **Input Signal Acquisition**, **Power Supply**, and **Control Unit**. Each function is interconnected according to the type of interaction between the respective blocks.

The signal chain was designed so that the test signal is generated within the Control Unit and then routed to the Output Conditioning stage, where it is converted and conditioned to produce the excitation signal applied to the DUT. The response of the system under test is subsequently captured by the Input Signal Acquisition stage and sent back to the Control Unit, where it is processed to extract the relevant measurement results.

3.2.1 Output conditioning

The first sub-function represents the output stage of the system, namely how the digital test signal generated by the control unit is converted into a signal capable of exciting the DUT. Also in this case, by studying the existing measurement system, it was decided to use an exciter, specifically a voice coil. Its behaviour is like a loudspeaker: it converts an electrical voltage into diaphragm movement, which in turn induces vibrations in the DUT.

Based on these considerations, it was decided to use a digital-to-analogue converter (DAC) followed by a power amplifier, to convert the digital signal into an analogue voltage and provide sufficient power to drive the exciter.

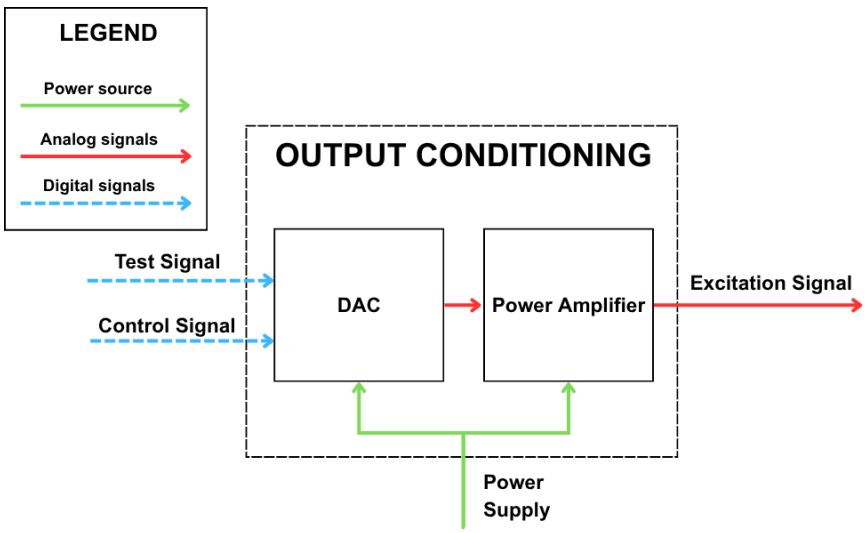

Figure 3.3. Output conditioning function

3.2.2 Input signal acquisition

The second sub-function defined represents how the signal is received and prepared to be transmitted to the control unit. By analysing the state of the art of similar devices, and particularly the measurement system currently in use, it was determined that the response of the DUT must be captured by a device capable of detecting the vibrations generated by the DUT.

A simple example is a microphone, which converts the pressure waves produced by a vibrating object into an electrical voltage. Another example is an accelerometer, which converts the motion of a surface into an electrical voltage. Based on these observations, it was decided to implement the system input to accommodate the use of a condenser microphone.

To acquire the voltage signal from the microphone and convert it into a digital signal for the control unit, three components were defined: phantom power, microphone pre-amplifier, and analogue-to-digital converter (ADC).

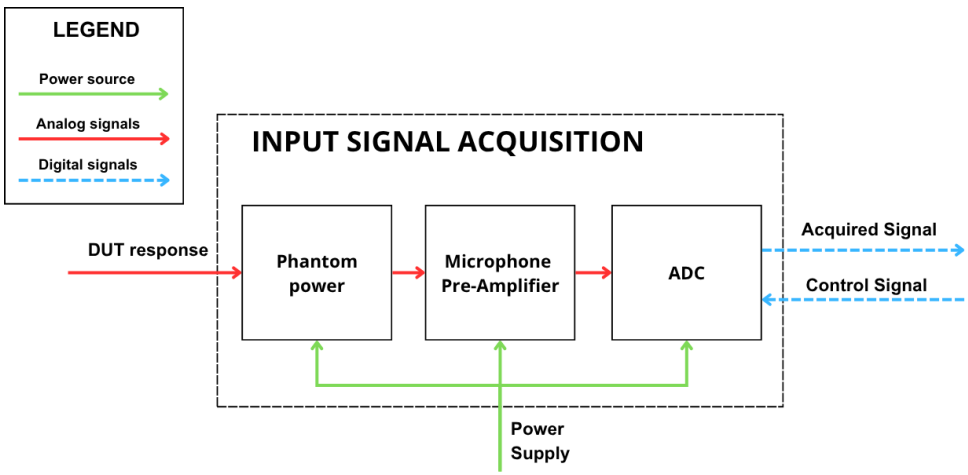

Figure 3.4. Input signal acquisition function

3.2.3 Power supply

The third sub-function concerns the system power supply. The selected solution is a wired power supply. This choice stems from the need for relatively high-power levels from specifics, which led to excluding a battery-based solution. Furthermore, the requirement for an external power source does not represent a limitation, since the tests

will be carried out in a laboratory or in facilities where access to the electrical grid is readily available.

The power supply will need to be further developed in detail to provide the different voltage levels required by all the hardware components of the device. The identified elements include an AC/DC converter, used to convert the alternating current from the mains into direct current suitable for low-voltage devices, and one or more DC/DC converters to generate the various regulated supply voltages required by the different subsystems.

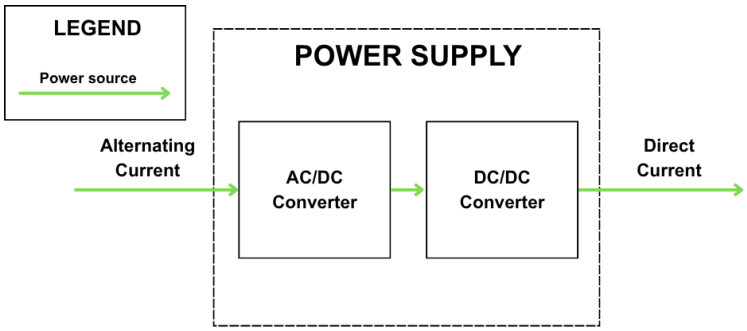

Figure 3.5. Power supply function

3.2.4 Control unit

The last function is represented by control unit. It is the core element of the proposed system and is responsible for managing the main operations required for the measurement process. Its architecture is composed of three main components shown in the Figure 3.6: a microcontroller unit (MCU), a memory module, and a Bluetooth communication interface.

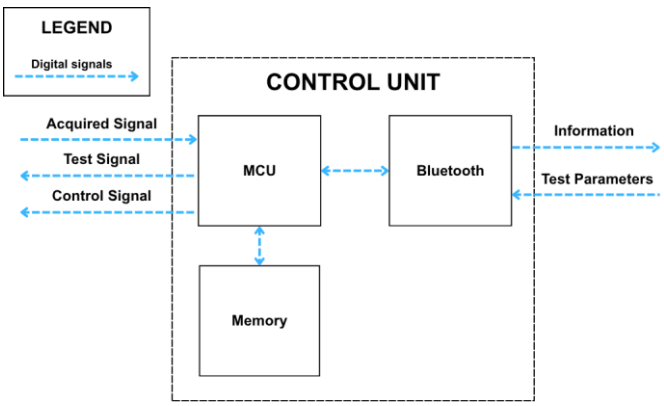

Figure 3.6. Control unit function

The **MCU** acts as the central processing element of the system. It manages the generation of the test signal, the acquisition of signal from the device under test, and the subsequent signal processing operations required to extract the measurement results. In particular, the MCU executes the algorithms necessary to compute the impulse response and derive the corresponding frequency response of the measured system as shown in section 2.3.1.

Since the measurement process requires the storage of a significant amount of audio data in the order of Mb, an external memory module is included in the control unit architecture. This **Memory** is used to store both the generated excitation signal and the acquired response signal, enabling the execution of digital signal processing operations that require access to large data buffers.

The final component of the control unit is the **Bluetooth** communication module. This interface enables wireless communication between the embedded device and a dedicated mobile application. Through this interface, the user can configure the system parameters, such as the characteristics of the excitation signal, visualize the measurement results in a user-friendly graphical environment and set the input and output gain.

3.3 Hardware components

The design process focuses on the selection and integration of the main electronic components that compose the system. Attention is given to ensuring that the chosen components satisfy the performance requirements defined during the system specification phase.

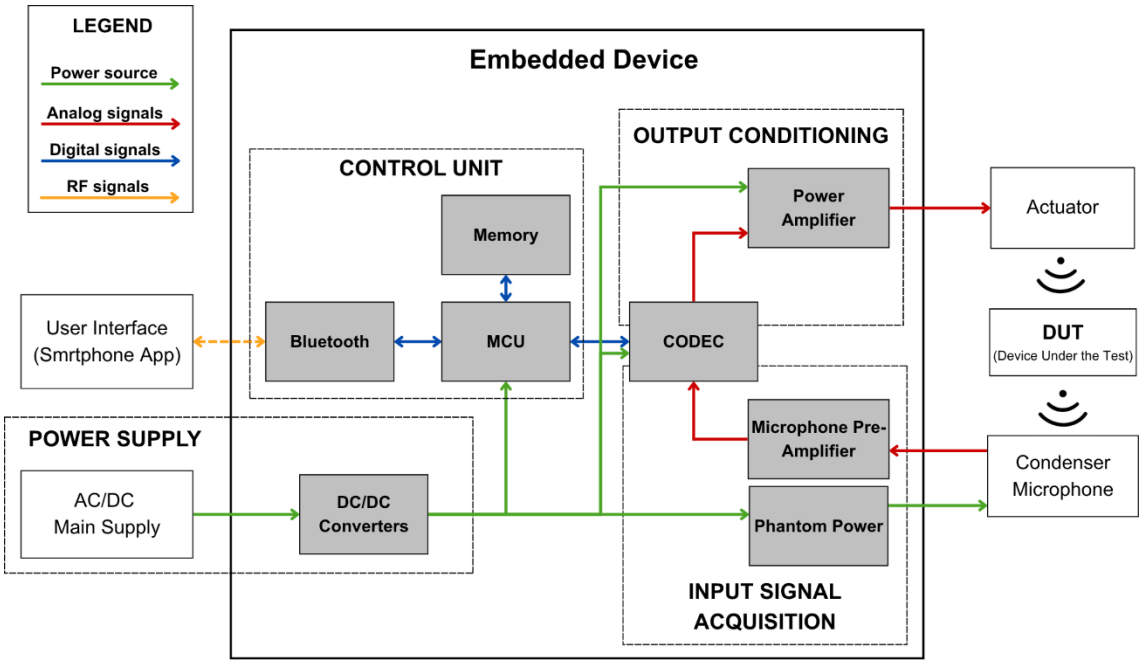

Figure 3.7. Finalized hardware layout of the embedded device

Final hardware layout is shown in Figure 3.7. It was derived from the previously defined system architecture and includes the final components selected for the implementation, as well as the way they effectively interface with each other.

The block called "CODEC" contains both the ADC and DAC. By integrating multiple functionalities within the same chip, it is possible to reduce the number of discrete components, simplify the hardware design, and improve the overall compactness of the system.

3.3.1 Microcontroller (MCU)

Based on the required specifications, a microcontroller (MCU) from the STM32 family was selected, as it currently represents one of the most suitable solutions in the embedded systems field. This choice is motivated by its wide adoption, extensive documentation, and the availability of dedicated development tools for low-level programming. STM32CubeMX significantly simplifies and accelerates code generation for peripheral configuration and data processing tasks.

Figure 3.8. Picture of STM32_L476RG (MCU) chip

Specifically, the STM32_L476RG shown in the Figure 3.8 was chosen. This microcontroller is based on an Arm Cortex-M4 core. The Arm Cortex architecture has enabled major advancements in microcontroller-based systems, making it possible to employ such devices in applications requiring considerable computational capability, including audio signal processing.

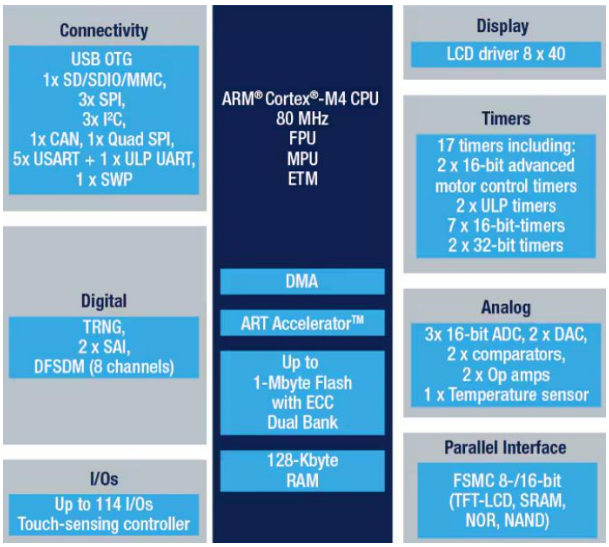

Figure 3.9. STM32_L476RG (MCU) chip features

The main features of this component are summarized in Figure 3.9 below [16]. These characteristics were carefully compared with the project requirements, with particular attention to computational performance and the availability of on-chip peripherals.

3.3.2 Bluetooth (BLE)

For the implementation of Bluetooth communication, the BlueNRG-M0 chip was selected, which supports Bluetooth 4.2 Low Energy (BLE). It is show in the Figure 3.10.

Figure 3.10. Picture of BlueNRG-M0 chip

This component was chosen due to its ease of integration and its compatibility with the selected MCU platform. In addition, BLE protocol ensures low power consumption, reliable wireless communication, and straightforward is interfacing with smartphone applications, making it well suited for the intended user's interface architecture.

Figure 3.11 illustrates Bluetooth chip architecture and its main features [17]. Communication with the MCU is handled through an SPI interface. The device operates with a +3.3 V power supply.

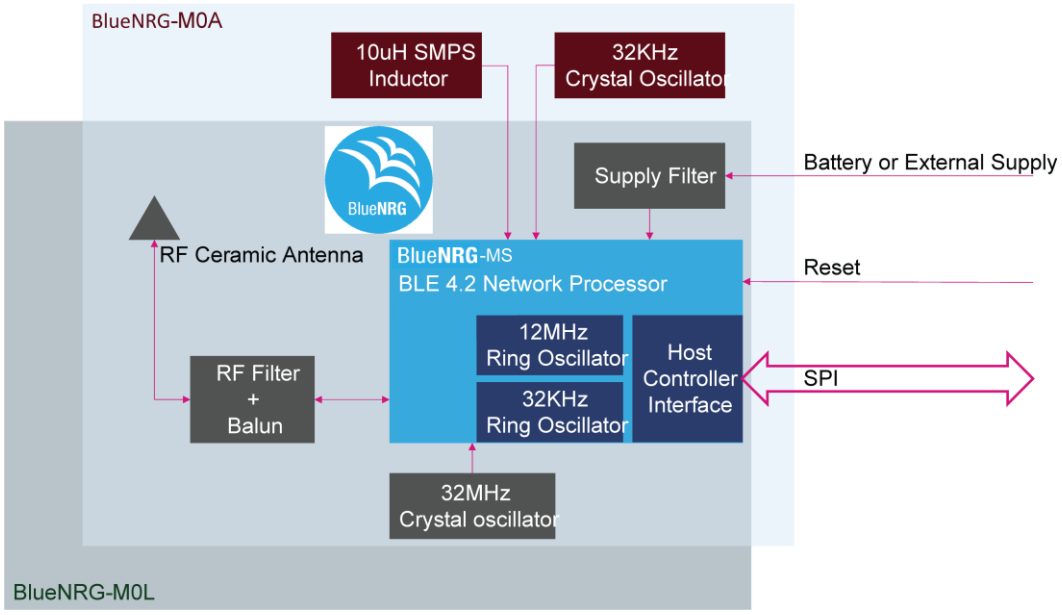

Figure 3.11. BlueNRG-M0 chip hardware architecture

3.3.3 Sample frequency and CLOCK

A key aspect in the system design was the definition of the sampling frequency F_S ,in accordance with IEC 61606. This choice was made based on the Nyquist-Shannon sampling theorem, which states that, to avoid aliasing, the sampling frequency must be at least twice the highest frequency present in the signal to be acquired. The maximum frequency that can be correctly represented is defined as the Nyquist frequency, f_N, and is expressed by the following relation:

$$f_N = \frac{F_S}{2} \qquad (1)$$

In this specific case, the system has been designed to operate within the audio band, ranging from 20 Hz to 20 kHz. By rearranging the formula and applying $f_N = 20$ kHz, we obtain:

$$F_S = 20\ [kHz] \cdot 2 = 40\ [kHz]\ \rightarrow 48[kHz] \qquad (2)$$

It follows that the minimum theoretical sampling frequency should be greater than 40 kHz. However, using a value exactly at the theoretical limit would not be sufficient in a real-world context, as the finite slope of the analogue anti-aliasing filter and the non-ideal behaviour of components must be considered.

For these reasons, a sampling frequency F_S of 48 kHz was chosen, a standard value in professional audio systems. This choice provides a guard band between the 20 kHz audio range of interest and the Nyquist frequency (24 kHz), reducing distortions near the edge of the useful bandwidth while ensuring compatibility with industrial standards. The adoption of 48 kHz thus represents an optimal compromise between theoretical requirements, design constraints, and interoperability with commercial audio devices.

Following the definition of the sampling frequency, the audio clock architecture of the system was designed. In A/D and D/A conversion applications, it is essential to have a stable, high-frequency master clock (MCLK), as timing precision directly affects jitter, signal-to-noise ratio (SNR), and total harmonic distortion (THD).

The audio clock frequency is determined based on the sampling frequency according to the general relationship:

$$f_{MCLK} = N \cdot F_S \quad (3)$$

Where:

- f_{MCLK} is frequency master clock,
- F_S is Sample Frequency,
- N integer multiplication factor (128, 256, 384 o 512).

In this specific case it has been chosen $F_S = 48$ kHz and an integer multiplication factor of $N = 256$, therefore, the clock frequency is:

$$f_{MCLK} = 256 \cdot 48\,000\ [Hz] = \ 12\,288\,000\ [Hz] = 12{,}288\ [MHz] \quad (4)$$

The choice of 12.288 MHz therefore stems directly from an exact mathematical relationship with the selected sampling frequency. Using an integer multiple allows the internal clocks of the codec and the I²S communication timing to be generated through simple digital dividers, avoiding the use of fractional PLLs, which would introduce additional phase noise and jitter.

The microcontroller used, the STM32_L476RG, has an internal high-speed oscillator of 8 MHz; however, this value is not an integer multiple of the chosen audio sampling frequency and cannot directly provide the required master clock with the necessary precision. For this reason, an external active 12,288 MHz oscillator dedicated to the audio subsystem was adopted, ensuring improved timing stability, reduced jitter, and overall better system performance.

3.3.4 Memory

A crucial aspect of the system design concerns the size of the memory space required for proper operation. This was carried out by analysing the main functions that the device must perform and determining the memory needed to support each operation.

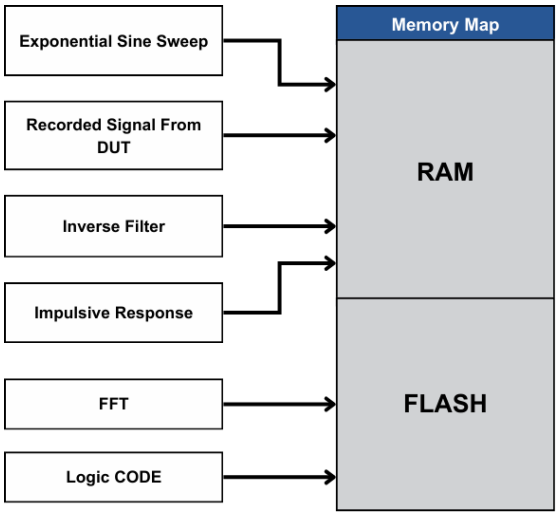

Figure 3.12. Memory map

Figure 3.12 shows the memory map, illustrating all files and data that need to be stored within the system. This includes audio files for signal generation, temporary buffers for real-time processing, configuration parameters, and any intermediate data required by the control unit. Proper memory allocation ensures smooth operation, prevents data loss, and allows the system to handle signals of the intended size without interruptions.

Memory Map	Size
Exponential Sine Sweep	1.9 MB
Recorded Signal From DUT	1.9 MB
Inverse Filter	1.9 MB
Impulsive Response	1.9 MB
RAM Total Size	8 MB

Memory Map	Size
FFT	64 KB
FFT	256KB - 1MB
FLASH Total Size	2 MB

Figure 3.13. Total memory size

In Figure 3.13 values that define the memory space required for each individual part are reported. These values were derived by means of the following equations.

In the first slot there is test signal that it will be generated by MCU. The Equation (5) shows how to calculate the dimension of memory size for this file.

$$ESS\ Size\ (Byte) = \frac{F_S \cdot N_{Bit} \cdot N_{ch} \cdot t}{8} \qquad (5)$$

Where:

- F_S is Sample Frequency.
- N_{Bit} Bit depth for sample
- N_{ch} Number of channels,
- t Signal duration.

In our specific case, a duration of $t = 10\ s$ was chosen, in line with Farina's ESS method. The sampling frequency is $F_S = 48$ kHz, with a bit depth of 16 bits and 2 channels, since the codec is stereo. Therefore, the file size can be calculated as:

$$ESS\ Size = \frac{48\ 000[Hz] \cdot 16\ [Bit] \cdot 2 \cdot 10\ [s]}{8} = 1\ 920\ 000\ [Byte] \cong 1.9\ [MB] \qquad (6)$$

Shifting now to the signal acquisition, we can state that the recorded file, the inverse filter, and the impulse response will all have the same size as the test file, approximately 2 MB each, since they are generated from the recording and processing of that signal. Therefore, for these three signals combined, 6 MB of storage space will be required.

As regards the code size, it is difficult to provide a precise value since it depends on several factors. To obtain a reasonable estimate, an external survey was considered to identify a range of values within which the required memory space for code of this type in an embedded system could fall. A range has been defined between 256 KB – 1 MB.

Finally, the file generated by applying the FFT to the impulse response signal was analysed. To reduce the memory required for spectral processing, it was decided to implement the FFT using the CMSIS-DSP libraries. In this approach, the same memory buffer is used both to store the input samples and to store the FFT result, avoiding the allocation of a separate output array.

Considering a window of length $N = 16{,}384$ samples per channel and processing in float32 format (4 bytes per sample), the memory required for the computation buffer is:

$$FFT\ Size = 16384 \cdot 4\ [byte] = 65536\ [byte] = 64\ [kB] \quad (7)$$

The use of CMSIS-DSP libraries [15] therefore allows for a significant reduction in RAM usage, enabling the execution of the FFT while maintaining a large analysis window.

The total memory required to perform all operations is shown in Figure 3.13 updated memory map is presented below, showing the size of each file.

Based on the calculations performed and considerations discussed, the most suitable choice is an SDRAM memory of at least 8 MB. This type of memory offers high read and write speeds, which are essential since the data must be continuously updated and overwritten during each measurement cycle.

As for storing the results (FFT), the internal Flash memory of the microcontroller can be used. Flash memory is not a volatile memory and allows data to be retained even after the device is powered off, making it suitable for saving measurement results or configuration parameters that need to persist over time.

3.3.5 CODEC + Microphone pre-amplifier

For signal conversion stage, as discussed at beginning of section 3.3, a codec integrating both analogue-to-digital (ADC) and digital-to-analogue (DAC) conversion in a single component was chosen. After a careful evaluation, the MAX9867 was selected.

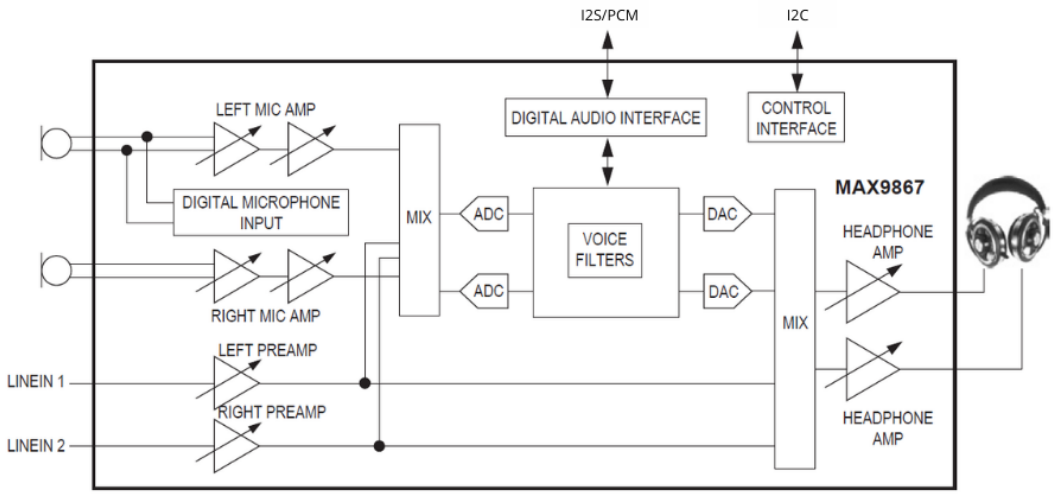

Figure 3.14. MAX9867 CODEC + Mic. Pre-amp. hardware architecture

Figure 3.14 depicts codec architecture with also a microphone preamplifier inside, a key feature that further simplifies the device design by integrating the ADC, DAC, and microphone preamp into a single component. This reduces the number of discrete components, saves board space, and streamlines the overall system architecture.

Regarding the interfacing with the microcontroller, communication will take place via the I^2C interface for parameter control, such as adjusting input and output volumes and configuring operating settings. As for the audio signal path codec will interface with the microcontroller via I^2S protocol in slave mode. This configuration allows the transmission and reception of stereo digital audio signals, with the MCU acting as the master that provides the necessary clock signals for synchronization.

Regarding the codec performance [18] shown in Figure 3.15, reference was made to the product datasheet, which allowed verification of its main specifications. The most significant parameter is the signal-to-noise ratio (SNR), specified as 90 dB.

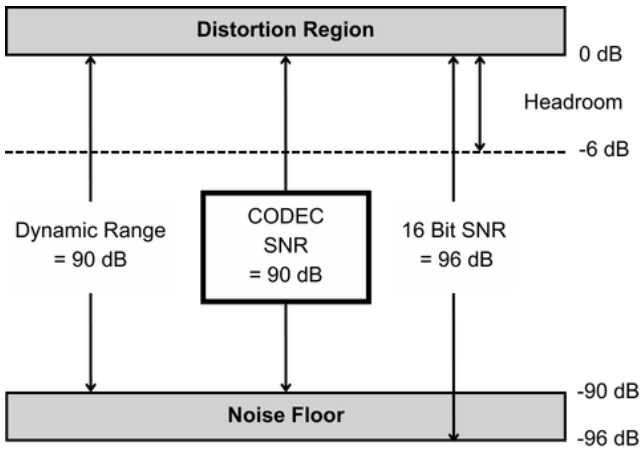

Figure 3.15. MAX9867 CODEC SNR diagram

Therefore, the CODEC operates close to the theoretical limit imposed by the 16 Bit resolution, representing the best achievable performance for a device in this category. In Chapter 5 dedicated to device testing, the extent to which the final implemented system deviates from this theoretical SNR value was evaluated.

3.3.6 Power amplifier

For power amplification stage, a Class D power amplifier was selected that meets the project specification. They are to drive an exciter with a nominal impedance of 3 Ω and a current absorption of approximately 0.5 A.

Using equation (8), the minimum required output power of the amplifier was determined:

$$Power = R \cdot I^2 = 3 \cdot 0{,}5^2 = 0.75\ [W] \qquad (8)$$

Based on this calculated minimum power requirement and the project specifications related to audio quality, the MAX98306 amplifier was selected. This device can deliver up to 3.7 W into a 3 Ω load, which largely exceeds the minimum required power of 0.75 W [19]. This margin ensures reliable operation, sufficient headroom for dynamic signal peaks, and reduced distortion when operating below its maximum output capability.

Selection of this amplifier therefore guarantees both adequate power delivery to the exciter and compliance with the desired audio performance standards.

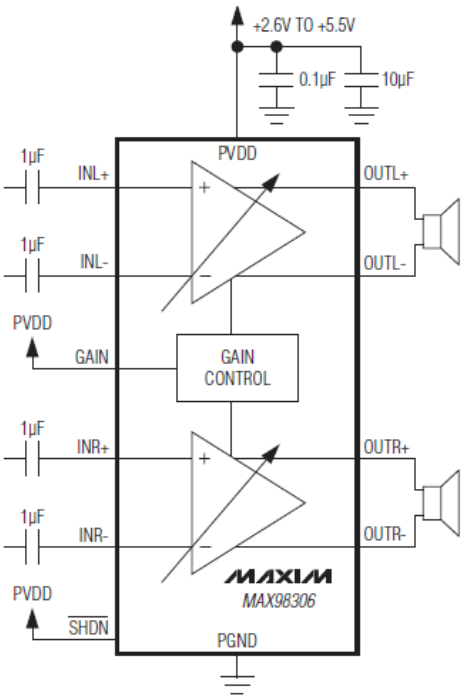

Figure 3.16. MAX98306 Class D power amplifier hardware architecture

In Figure 3.16, hardware architecture of the device is shown. The amplifier features two differential output channels and provides gain control in 6 dB steps.

3.3.7 Phantom power

Another fundamental part of the device design is the phantom power circuit. This circuit provides a +48 V DC and 20 mA supply to the condenser microphone used for measurements, with the power delivered over the same cables as the analogue audio signal.

For the design of the phantom power circuit, after a study of the state of the art [20] [21] [22], it was possible to design a system capable of supplying +48 V to the microphone via the same lines used for the audio signal. The schematic of the designed circuit is shown in Figure 3.17.

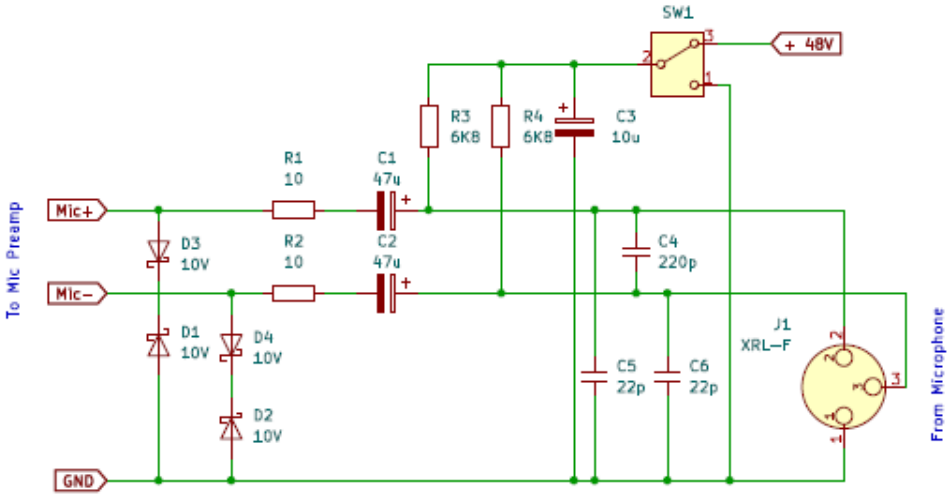

Figure 3.17. Application circuit schematic of phantom power (+48 V)

3.3.8 Power supply

This paragraph describes the design of the power supply. As stated in section 3.2.3, the device was intended to operate from the mains supply, which requires the use of an AC/DC converter since the system operates at low voltage.

A crucial aspect considered during the power supply design was how the various subsystems are connected to the supply line. After careful analysis, a star-ground configuration was selected as the most effective method to minimize interference and ground loop issues. In this approach, a single central power rail is defined, and all subsystems are grounded directly from this common node (case). Series connections between devices are avoided, as they may introduce unwanted voltage drops, noise coupling, and ground loop currents.

In addition to the grounding strategy, particular attention was devoted to the architecture of the power conversion stages. The system employs a combination of DC/DC converters and linear regulators to generate the required voltage levels for the different subsystems. Specifically, switching DC/DC converters, both step-down (Buck) and step-up (Boost), are used to efficiently derive intermediate voltage rails from the main +12 V supply. Buck converter is utilized to generate lower voltage levels required by digital and control circuitry, ensuring high efficiency and reduced power dissipation, while Boost converter is employed to generate higher voltage levels, such as the +48 V rail required for phantom power in condenser microphones.

However, due to the intrinsic switching nature of DC/DC converters, which introduces high-frequency noise and ripple, additional filtering stages and low-noise regulation are necessary for sensitive digital sections. For this reason, Low Drop-Out (LDO) linear regulators are employed downstream of the switching converters to provide clean and stable voltage supplies for noise-critical components, such as the audio CODEC and MCU. This hybrid approach allows combining the high efficiency of switching regulators with the low-noise performance of linear regulators.

Overall, the adopted power supply architecture ensures an effective trade-off between efficiency, noise reduction, and system stability, which is essential for achieving reliable performance in audio measurement applications. Figure 3.18 illustrates power supply layout and summarizes the adopted distribution.

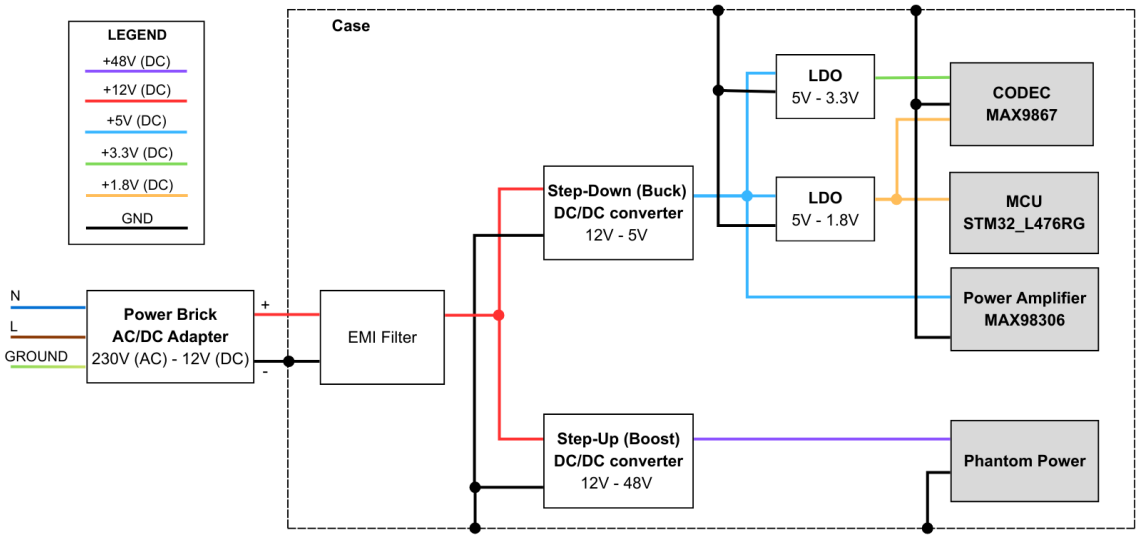

Figure 3.18. Block diagram of the power supply circuit

Table 3.4 shown max current absorption of each component as reported by datasheets [16] [18] [19].

Components	Max Current [mA]
CODEC (MAX9867)	12
MCU (STM32L476RG)	800
Power Amplifier (MAX98306)	800
Phantom power	20

Table 3.4. Max current absorption of each device

Once the overall power supply layout was defined, including the various components and the maximum power consumption of each subsystem, the design of the individual stages was carried out.

POWER BRICK (AC/DC ADAPTER)

An external power adapter was chosen to minimize interference generated by the AC/DC conversion stage, which could otherwise introduce distortion into the audio signal path. The external supply must be capable of delivering a peak current of 2 A to ensure proper operation under worst-case load conditions in accord to Table 3.4.

EMI FILTER

To ensure high-quality audio performance and reliable operation of the system, an EMI (Electromagnetic Interference) filter has been implemented on the +12 V power supply line. The primary purpose of the EMI filter is to suppress high-frequency noise generated by the switching power supply or external sources, which could otherwise propagate into the audio circuitry and degrade parameters such as signal-to-noise ratio, total harmonic distortion, and jitter. In addition, the filter prevents the device itself from injecting noise back into the power network.

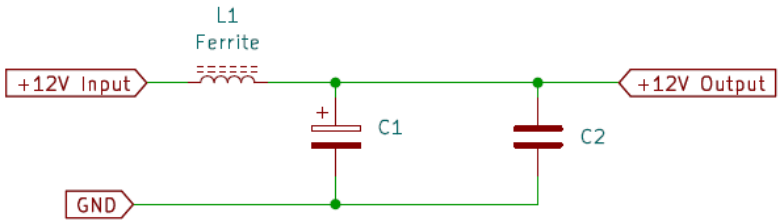

Figure 3.19. Base circuit of an EMI filter

The EMI filter design with a classical LC or π topology is shows in Figure 3.19. Electrolytic capacitor (polarized) is responsible for ensuring a stable voltage and absorbing slow or transient current variations. In place, ceramic capacitor (non-polarized) allows the decoupling and filtering of high-frequency noise.

First step in the design process is select cut-off frequency target. For this application ripple noise from power supply must be limited to a threshold of approximately 1 kHz – 2 kHz [11]. For a simple LC filter, the cut-off frequency is calculated as:

$$f_c = \frac{1}{2\pi\sqrt{LC}} \qquad (9)$$

Where L is the ferrite and C is the capacitance of the total bypass capacitor. In this design it was chosen a ferrite of $L1 = 5\ \mu H$, an electrolytic capacitor of $C1 = 1000\ \mu F$ and a ceramic capacitor $C2 = 100\ nF$. Cutting frequence f_c will have a value of:

$$f_c = \frac{1}{2\pi\sqrt{5 \times 10^{-6} \cdot 1000 \times 10^{-6}}} \cong 2250\ Hz \approx 2.25\ kHz \qquad (10)$$

This approach allows the system to maintain a clean power supply, which is essential for accurate audio measurements and reliable device operation, providing a solid foundation for high-fidelity signal generation and processing.

STEP DOWN (BUCK)

Regarding the buck converter, its purpose is to step down the +12 V input to +5 V to power the amplifier and the two LDO regulators, which will further reduce the voltage for the low-voltage digital and analogue sections. The buck converter must support a peak current of 1 A. The selected component for this stage is the LM2596.

Rather than performing a fully detailed component-by-component design, the implementation was based on reference circuits already validated by the manufacturer [23], ensuring compliance with the project specifications. The schematic of the circuit implementing this converter is shown in Figure 3.20.

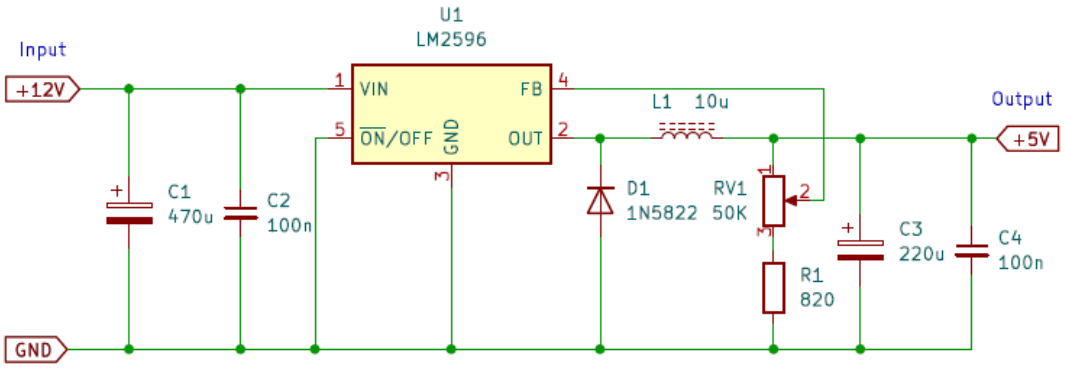

Figure 3.20. Application circuit schematic of the LM2596 step-down DC/DC converter

STEP-UP (BOOST)

For the design of the boost circuit, the specifications were first defined: an input voltage of $V_{in} = +12$ V, a desired output voltage of $V_{out} = +48$ V, and a maximum load current of $I_{load} = 20$ mA, since a single microphone requires about 10 mA and the system is designed to power two microphones. Other key design considerations include limiting the output ripple to around 10 mV and setting a switching frequency of $f_s = 100$ kHz.

The components to be dimensioned include the inductor, the MOSFET controlling the duty cycle, the diode, and the filter capacitor. In Figure 3.21 there is basic configuration of the boost converter.

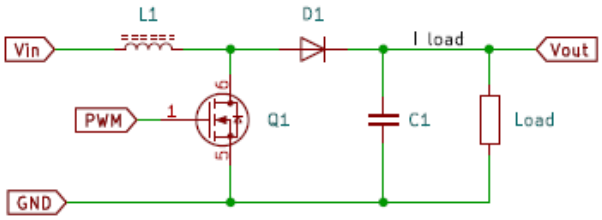

Figure 3.21. Base circuit of a step-up DC/DC converter

First step is to calculate duty cycle required for the conversion, given by equation (11).

$$D = 1 - \frac{V_{in}}{V_{out}} = 1 - \frac{12}{48} = 0.75 \qquad (11)$$

Next, the average inductor current $I_L$was determined using:

$$I_L = \frac{V_{out} \cdot I_{out}}{V_{in}} = \frac{48 \cdot 0.02}{12} = 0.08 \text{ A} \qquad (12)$$

For inductor sizing, assuming a current ripple of $\Delta I_L = 30\% \cdot I_L = 0.024$ A and a switching frequency of $f_s = 100$ kHz, the value was calculated as:

$$L = \frac{V_{in} \cdot D}{\Delta I_L \cdot f_s} = \frac{12 \cdot 0.75}{0.024 \cdot 100{,}000} \approx 3.75 \text{ mH} \qquad (13)$$

The output capacitor, designed to limit the voltage ripple to $\Delta V_{out} \leq 10$ mV , was calculated using:

$$C_{out} = \frac{I_{load} \cdot D}{f_s \cdot \Delta V_{out}} = \frac{0.02 \cdot 0.75}{100{,}000 \cdot 0.01} \approx 15\ \mu F \qquad (14)$$

For the diode, a Schottky type was chosen with a reverse voltage $V_R > 60$ V and an average current $I_{avg} \geq 20$ mA. For PWM generation, the LM2577 controller was adopted, allowing the realization of an adjustable boost converter capable of generating voltages from +5 V up to +55 V [24], ensuring easy implementation and low dissipation. In Figure 3.22 it is possible to see final schematic of boos circuit with also the components to improve the performance.

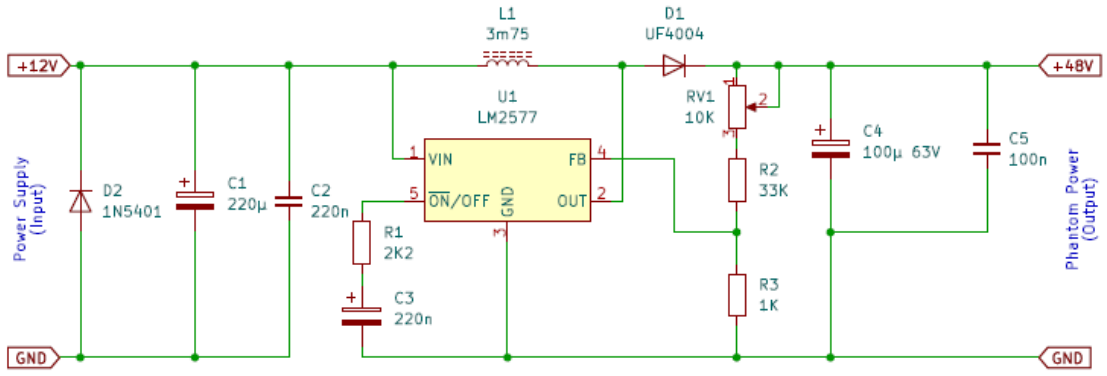

Figure 3.22. Application circuit schematic of the LM2577 step-up DC/DC converter

The entire design was carried out with safety margins in mind, to guarantee low power dissipation and reduced noise. These calculations allowed for accurate sizing of the main components of the converter, providing a solid foundation for designing an efficient circuit compatible with the low-noise requirements needed for the phantom power supply of the condenser microphone.

LDO (LOW-DROPOUT REGULATOR)

The last component discussed in the power supply design is the LDO (Low-Dropout Regulator). This device is used to directly power MCU and CODEC. LDO regulators provide a very clean supply voltage, which is essential since these chips operate with high-speed digital signals and require stable and reliable voltage references.

In this design, two LDOs were required: the LD39050 to step-down from +5 V to +3.3 V [25], and the LT3008 to step-down from +5 V to +1.8 V [26]. Schematic of the circuit required for their operation is shown in Figure 3.23. The design was carried out according to widely validated guidelines and reference implementations already established for these devices on the market.

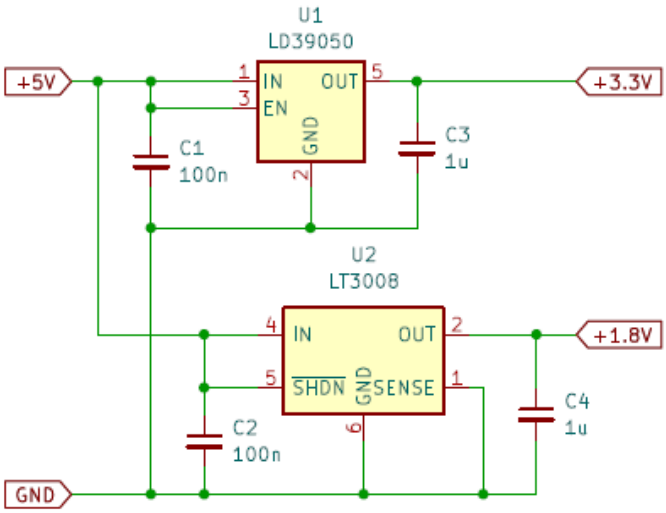

Figure 3.23. Application circuit schematic of LD39050 and LT3008 LDO regulators

3.4 Firmware architecture

After completing the hardware design of the system, the firmware design phase was carried out by defining firmware architecture and its different layers. An embedded architecture based on a microcontroller typically consists of three main sections:

- **Application layer**, which contains the program logic based on a state machine and high-level control algorithms.
- **Middleware layer**, which includes libraries that allow interaction between the application and the low-level drivers.
- **Driver layer**, which contains the low-level code directly interfacing with the hardware peripherals. In this section, HAL (Hardware Abstraction Layer) [27] libraries were used to simplify the development and configuration of the drivers.

Focusing on the program itself, as previously analysed, the device must generate a test signal, record the acoustic response of the environment, and process the acquired signal. Supporting operations include the generation of the inverse filter, communication with CODEC and a smartphone application, and the transmission of processed results to the app. The firmware code is based on a state machine.

Figure 3.24 shows the system architecture diagram summarizing firmware operation and its interaction with hardware components.

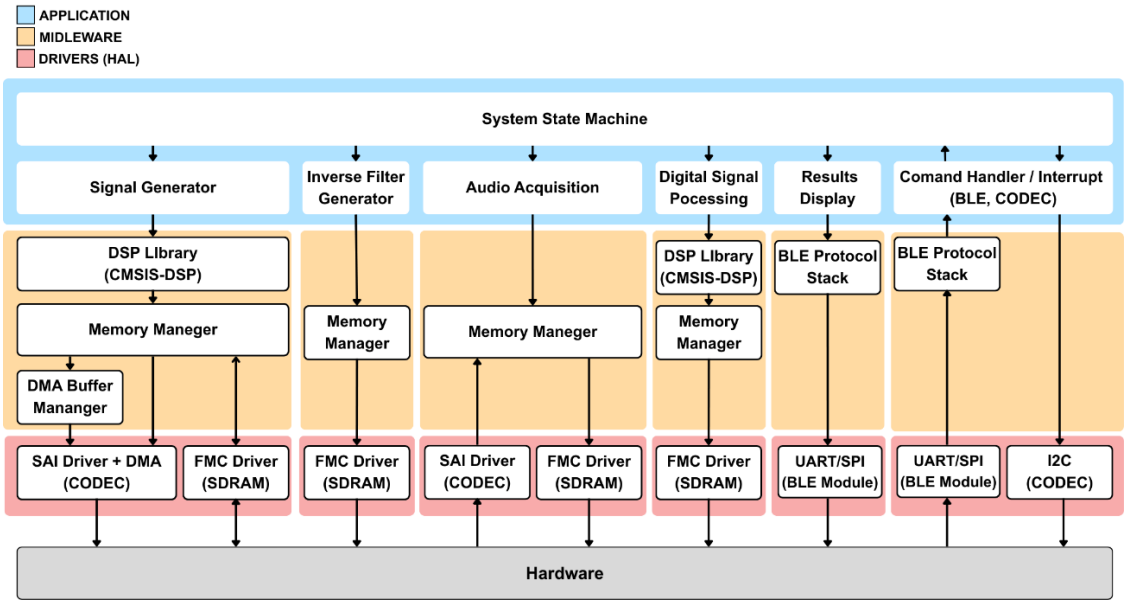

Figure 3.24. Block diagram with designed firmware architecture

3.5 Firmware application model

As shown in Figure 3.24, the high-level control logic, corresponding to the application layer, is based on a state machine that manages the various functions of the device. This approach, compared to a traditional polling logic, optimizes the performance of the microcontroller by allowing multiple tasks to be executed concurrently.

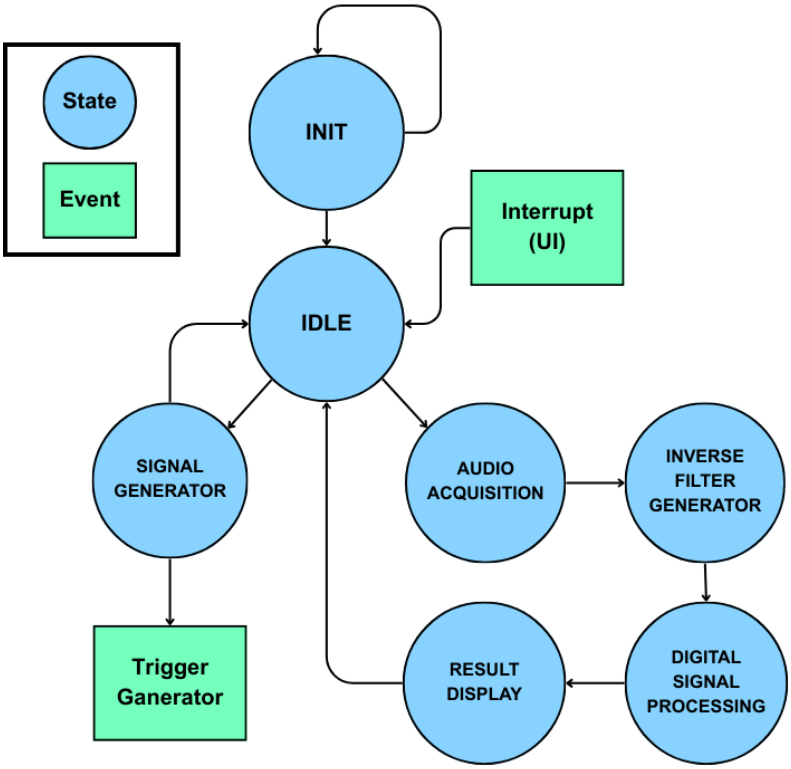

Figure 3.25. Block diagram of firmware logic based on state machine

In Figure 3.25 the main state machine is shown. It is designed using eight distinct states. These include not only the states implemented in the current prototype, but also those related to the acquisition and signal processing chain, which have not yet been fully developed. The latter were initialized to ensure code scalability, allowing for straightforward future improvements and integration of the missing functionalities. This architectural choice ensures that the firmware is modular, extensible, and ready for subsequent development phases.

3.6 Firmware modules

Following the design of the firmware architecture and the definition of the logic behind the application, the main modules that will form the final firmware were designed and defined.

3.6.1 Sine-Sweep generator

At the core of the entire system lies the theory of impulse response measurement using the Exponential Sine-Sweep (ESS) method (section 2.3.1). As discussed in the literature review chapter, this is a well-established and highly reliable technique for characterizing the frequency response of a vibrating object. This theory is based on the use of an Exponential Sine-Sweep signal, which is precisely the signal that the device is designed to generate and reproduce. The law that describes the Sine-Sweep signal is describe by equation (15).

$$x(t) = \left[\frac{\omega_1 \cdot T}{ln\left(\frac{\omega_2}{\omega_1}\right)} \cdot \left(e^{\frac{t}{T} ln\left(\frac{\omega_2}{\omega_1}\right)} - 1\right)\right] \tag{15}$$

Where ω_1 and ω_2 are respectively the starting and ending angular frequencies of the Sine-Sweep, and T represents the total duration of the generated signal.

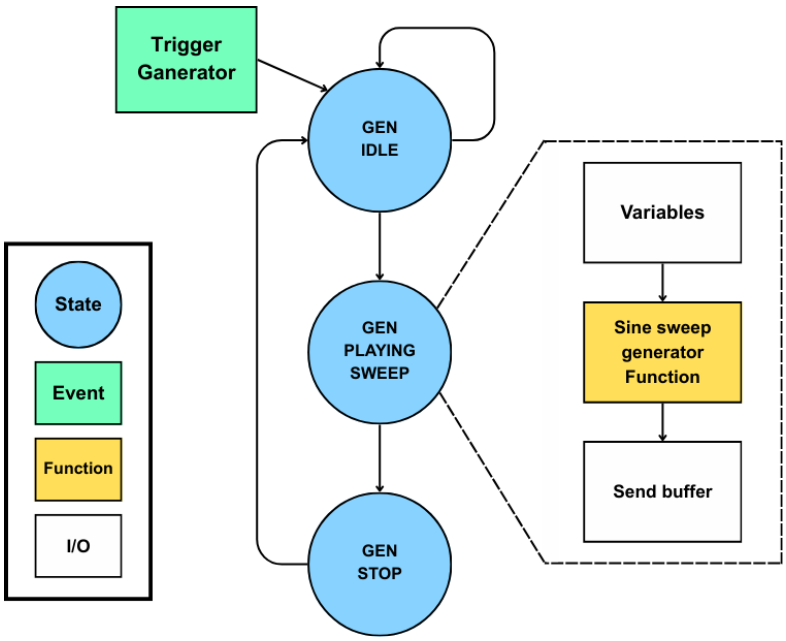

Figure 3.26. Block diagram of Sine-Sweep generator logic

Figure 3.26 shows the block diagram describing logic of the Sine-Sweep generator. The system is based on a state machine that switches state when an external trigger occurs. When the trigger is received, the system transitions to the generation state, where the function responsible for generating the Sine-Sweep, based on the corresponding equation, is executed. Once the signal generation is completed, the system moves to the STOP state and then returns to the IDLE state, where it waits for a new trigger event.

As regards the variables involved, they have been defined below:

- Start Frequency
- Stop Frequency
- Duration
- Amplitude

As regards the output buffer it will be explained in the next section 3.6.4.

Another important aspect considered during the signal generation stage is the application of fade-in and fade-out windows to the excitation signal, as recommended in the documentation related to the sine sweep measurement method. The purpose of this operation is to avoid abrupt amplitude transitions at the beginning and at the end of the signal, which could introduce spectral leakage and unwanted artifacts in the measured response. To ensure a smooth amplitude transition, a Hann window was adopted for both the fade-in and fade-out sections of the sweep signal [8]. The Hann window is defined with the following equation:

$$w[n] = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right), 0 \leq n \leq N-1 \qquad (16)$$

Where N represents the number of samples of the window and $n$is the sample index. In the proposed implementation, the Hann window is applied to a predefined number of samples at the beginning and at the end of the generated sine sweep signal. The fade-in is obtained by multiplying the first $N_f$samples of the sweep signal by the corresponding Hann window values:

$$x_{fade\_in}[n] = x[n] \cdot w[n], 0 \leq n < N_f \qquad (17)$$

Similarly, the fade-out is implemented by applying the reversed Hann window to the last samples of the signal:

$$x_{fade\_out}[n] = x[n] \cdot w[N_f - (n - (N - N_f))] \quad (18)$$

Where N is the total length of the sweep signal and N_f is the number of samples used for the fade region.

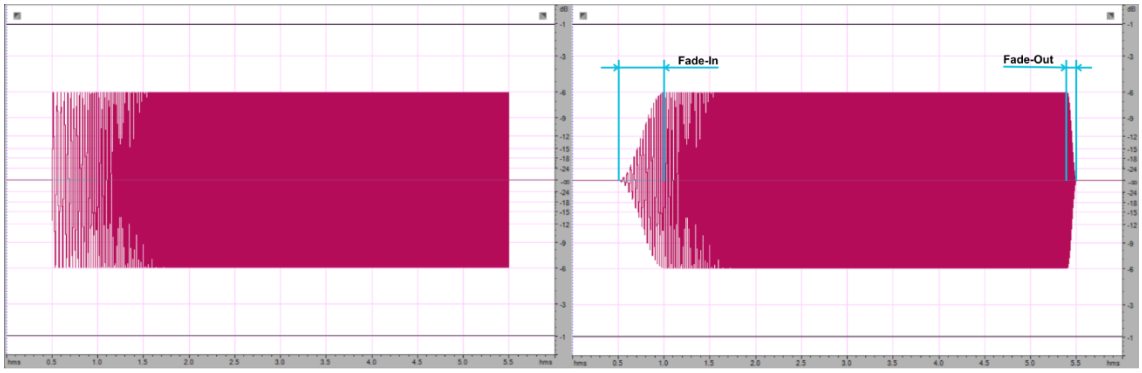

Figure 3.27. Comparison between Sine Sweep audio tracks with and without fade-in and fade-out Hann type

This approach guarantees a gradual rise and decay of the signal amplitude, as is possible to see in Figure 3.27, preventing discontinuities in the excitation signal and ensuring a more accurate estimation of the impulse response.

3.6.2 Audio transmission (I²S)

The I²S (Inter-IC Sound) protocol [18] was selected to implement the audio communication protocol between the microcontroller and the CODEC. This choice was based on the protocol's performance characteristics and its widespread adoption as one of the most used standards for digital audio transmission.

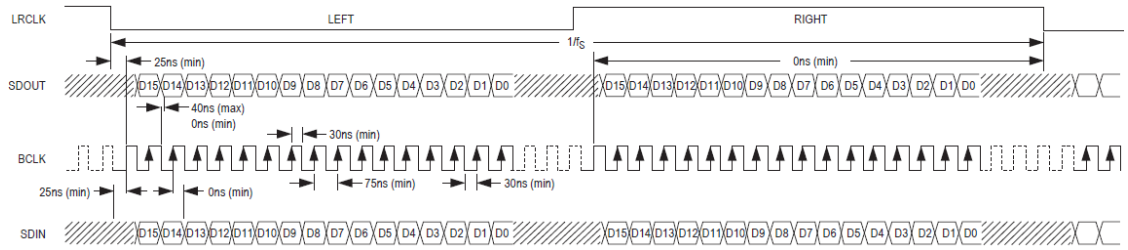

Figure 3.28. I²S communication timing diagram

The selected CODEC features a stereo I²S interface. As shown in Figure 3.28, each data buffer sent (SDOUT) or received (SDIN) is divided into two channels using the LRCLK signal: the first half corresponds to the left channel, and the second half corresponds to the right channel. The reconstruction of the audio stream is synchronized by the BCLK, which defines the sampling frequency (48 kHz) set by the microcontroller according to the Master Clock (12,288 MHz), previously designed and specified in the section 3.3.3.

Regarding the microcontroller, it is equipped with an SAI (Serial Audio Interface), as illustrated in the figure, which is fully compatible with the I²S protocol.

3.6.3 Communication with CODEC peripheral (I²C)

The I²C (Inter-Integrated Circuit) protocol [18] was chosen to implement communication with the peripheral (CODEC), due to its high versatility and efficiency. Additionally, it requires only two wires for operation, simplifying the hardware connections. Figure 3.29 below shows the timing diagram of the CODEC, which must be strictly followed to establish proper communication with the microcontroller.

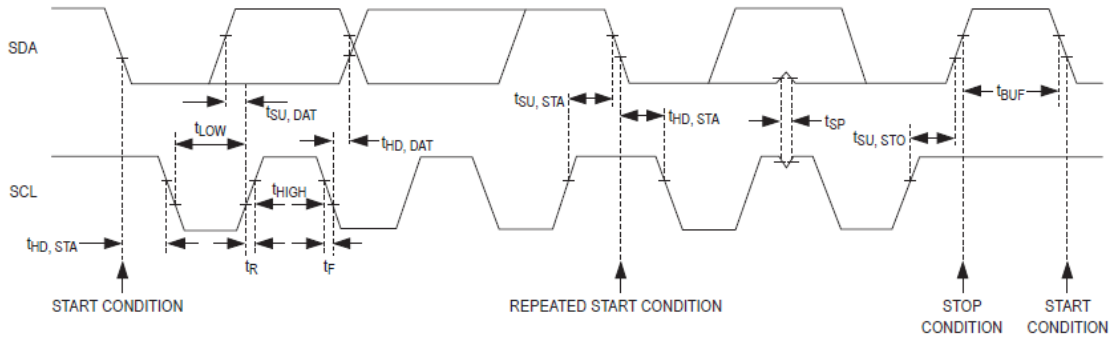

Figure 3.29. I²C communication timing diagram

3.6.4 Memory management (DMA)

A double-buffering strategy combined with DMA (Direct Memory Access) [11] was adopted to address the memory constraints of the selected MCU. Since the complete exponential sine sweep (approximately 1.8 MB) cannot be stored in the 128 KB internal RAM [16], the signal is generated and transmitted in smaller blocks rather than being allocated as a single array in memory.

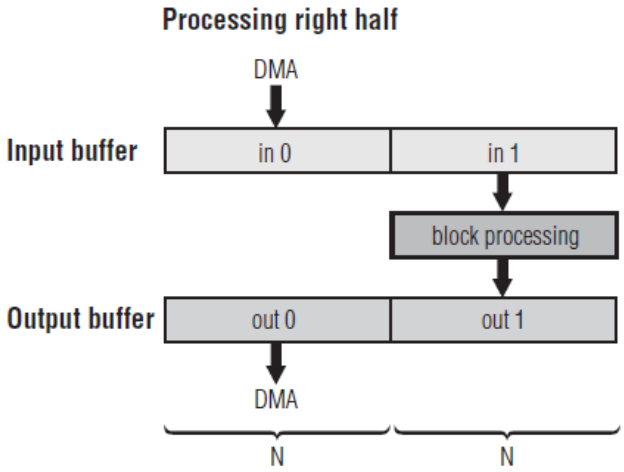

Figure 3.30. Double-buffering method (DMA) operating scheme

In the implemented double-buffering (ping-pong) configuration, audio stream is divided into two memory buffers of size N, as it is possible to see in Figure 3.30. While DMA transfers the content of one buffer to the codec, MCU simultaneously computes and fills the second buffer with the next block of samples. Once the transmission of the first buffer is completed, the roles are swapped: the newly filled buffer is transmitted, and the other is updated with fresh data. This alternating mechanism guarantees a continuous and uninterrupted audio stream, ensuring real-time signal generation despite the limited internal memory of the microcontroller.

3.6.5 Bluetooth communication (BLE)

Bluetooth Low Energy communication subsystem is implemented using the BlueNRG-M0 [17]. In this architecture, BLE protocol stack runs internally on the BlueNRG-M0 device, while the STM32 microcontroller executes the application firmware and manages interaction with the system.

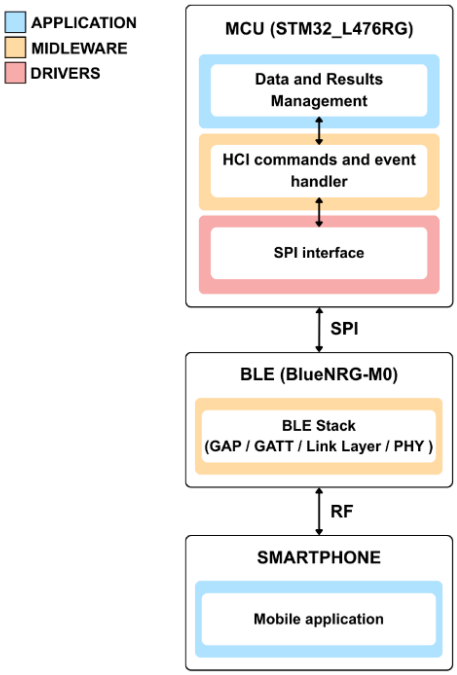

Figure 3.31. Firmware architecture of Bluetooth communication [28]

As shown in Figure 3.31, the firmware running on MCU is structured in multiple layers. At the lowest level, the hardware drivers handle SPI communication interface used to exchange commands and data with BLE module. Above this layer, a middleware component implements Host Controller Interface (HCI), which allows MCU to communicate with BLE stack running on the BlueNRG-M0. Application layer then implements logic required to configure communication services and manage data exchange with the external mobile application.

Interaction between MCU and BLE module is event driven. MCU sends HCI commands through SPI interface to configure BLE services and characteristics, while asynchronous events generated by BLE stack are received through interrupt signals and processed by the firmware event handler. All these parts allow a radio frequency communication with a smartphone device where there is an application that manage connection with device BLE and user interface.

4 IMPLEMENTATION

This chapter describes the implementation and prototyping phase of the device. Due to time and budget constraints, certain design choices were made that differ from the specifications initially defined during the conceptual phase.

Regarding the hardware, most of the system was fully developed; however, some components were selected that introduce limitations to the overall project performance. These choices were driven by availability, cost considerations, and the need to accelerate the prototyping process.

On the firmware side, the signal generation block and the communication/interface module with a smartphone application via Bluetooth have implemented. As previously mentioned, these decisions were primarily dictated by limited development time and economic constraints.

The rationale behind this approach was to deliver a functional and coherent prototype that could serve as a solid foundation for future developments, enabling a step-by-step refinement of the system. The primary objective was to realize a prototype capable of generating an exponential Sine-Sweep signal comparable to that produced using traditional measurement methods. In parallel, a preliminary smartphone application prototype was developed to allow communication with the device and configuration of key parameters, such as CODEC settings and test signal characteristics.

4.1 Hardware

This section addresses all aspects related to the hardware implementation of the prototype. The design choices adopted for implementing the application circuits of each component previously selected and dimensioned in the design chapter will be presented and discussed in detail.

To simplify and accelerate the prototyping phase, commercially available breakout boards (Evaluation Boards) have been used instead of designing custom PCBs. The only self-built circuits have implemented on Veroboard and include the power supply stage and phantom power circuit. This decision was driven by the need to develop the hardware as quickly as possible to focus on the firmware, which represents the most complex and central part of the project.

In a second phase, once the prototype has been fully completed and validated, a custom PCB can be designed to integrate all components into a single board, making the device more compact, efficient, and suitable for potential market deployment. In Figure 4.1 it is possible to see block diagram with the final hardware circuit with inside and outside connection and the specific boards used that they will be explained in the next sections.

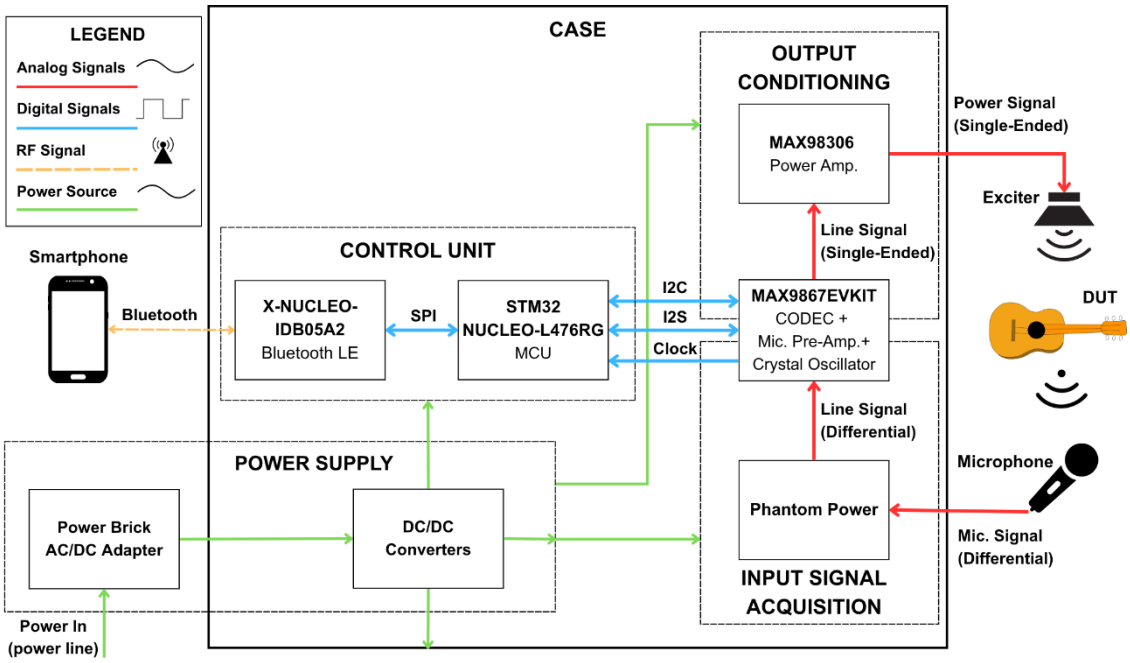

Figure 4.1. Block diagram of implemented hardware circuit with component's boards of embedded device

In Figure 4.2 it is possible to see block diagram with a zoom on the power supply circuit of the device with specific boards used to implement it.

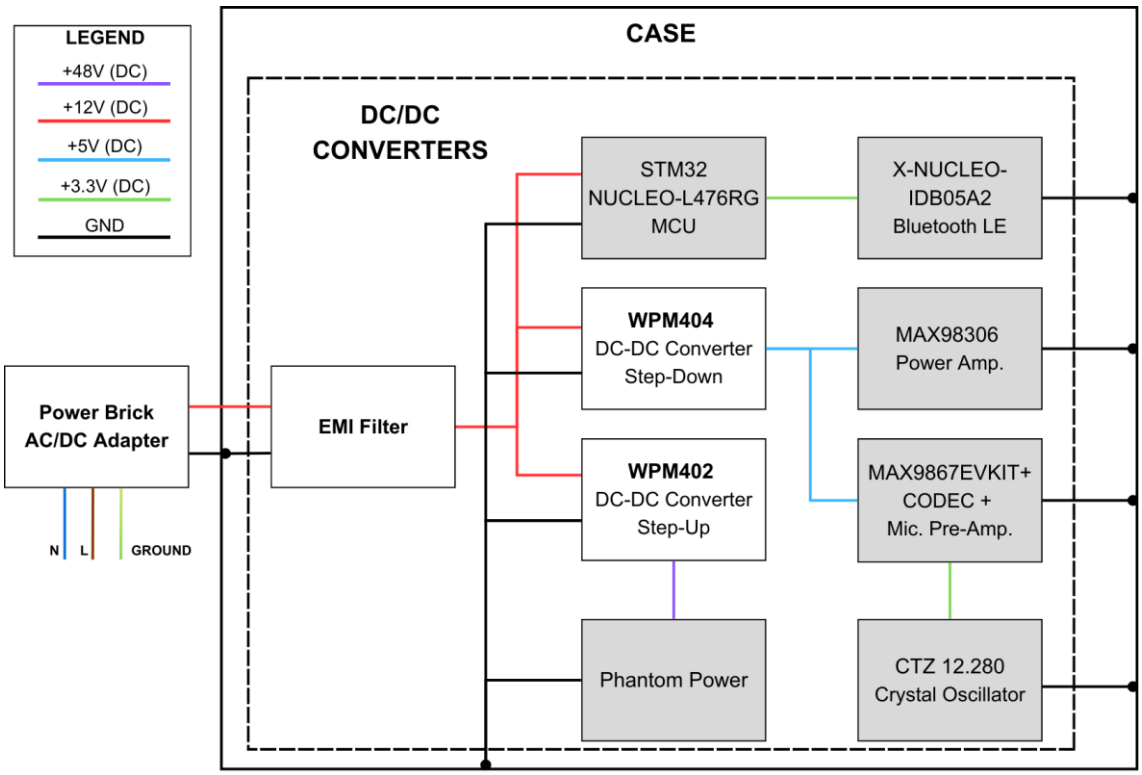

Figure 4.2. Block diagram of zoom on implemented power supply circuit with component's boards of embedded device

Figure 4.3 shows a picture of the implemented hardware circuit of the device based on block diagram shown in Figure 4.1 and designed in section 3.2

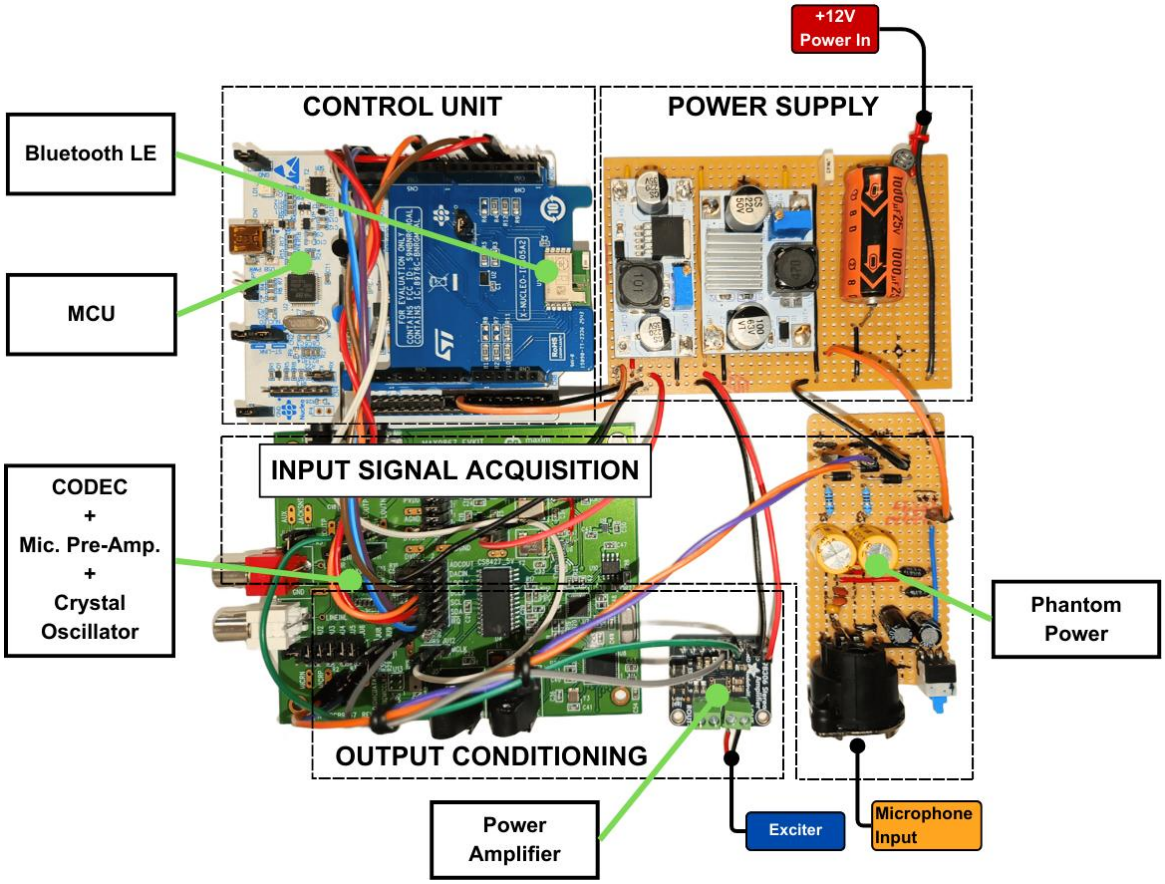

Figure 4.3. Picture of the final implemented circuit of embedded device

In Figure 4.4 it is possible to see zoom on the implemented power supply circuit based on block diagram shows in Figure 4.2 and designed in section 3.3.8.

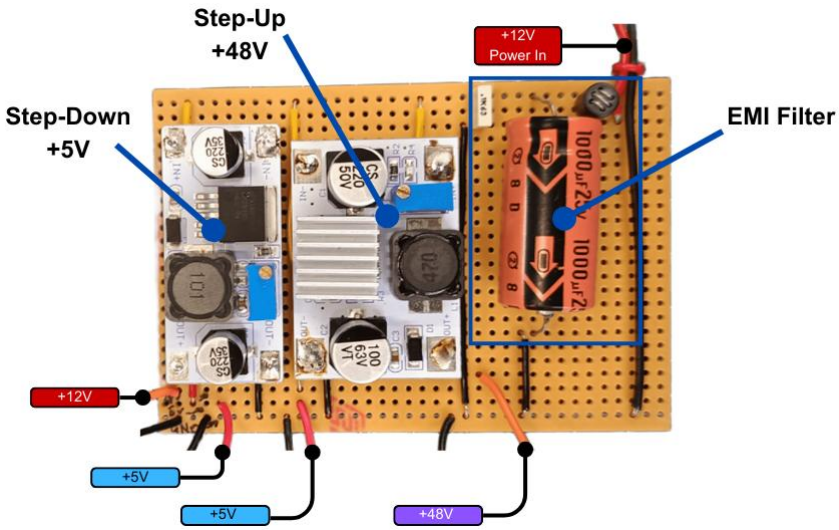

Figure 4.4. Picture of zoom on implemented power supply circuit of embedded device

4.1.1 STM32_NUCLEO-L476RG (MCU)

To implement the circuit for MCU it has been used an evaluation board of STM32 the NUCLEO-L476RG show in the Figure 4.5. In this board there are all application circuits [29] to ensure the proper operation of MCU. The board is powered with 12V and LDO on the board providing the chip supply voltage. To implement this behaviour, it is important that jumper JP5 in on pins 2 and 3 (E5V). In Figure 4.5 is shown a picture of the board.

Figure 4.5. Picture of STM32 NUCLEO-L476RG (MCU)

Regarding the board programming it has a ST-Link located on the top side of the PCB, allowing both debugging and connection of the board to a PC running the programming software.

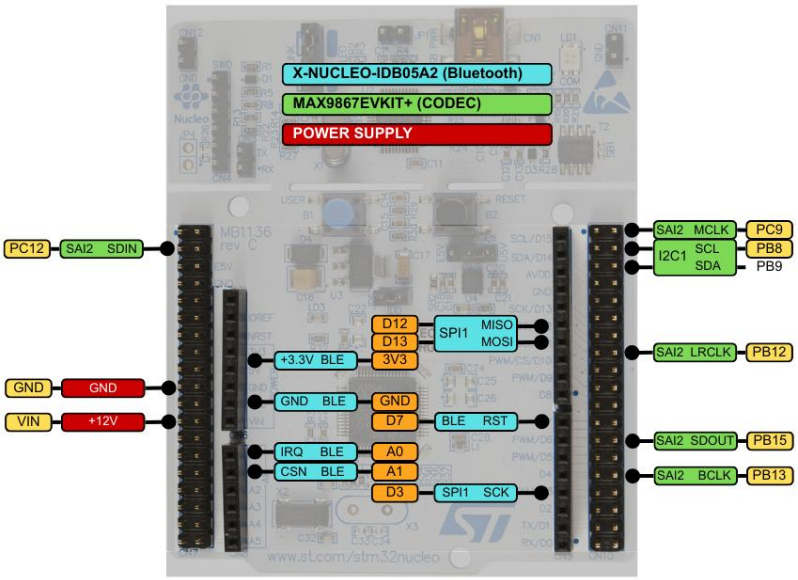

Figure 4.6. STM32 NUCLEO-L476RG pin-out

In Figure 4.6, the board pin-out is shown. The pin names correspond to those reported in the board datasheet, and the diagram also illustrates how the board interfaces with the other components of the device.

4.1.2 X-NUCLEO-IDB05A2 (BLE)

To implement the circuit required for the proper operation of the BlueNRG-M0 chip, the X-NUCLEO-IDB05A2 expansion board was selected. This board produced by STMicroelectronics is specifically designed for easy integration with STM32 development boards [30]. It simplifies both hardware interfacing and firmware development. In Figure 4.7 it is possible to see a picture of this board.

Figure 4.7. Picture of X-NUCLEO-IDB05A2 (Bluetooth)

In Figure 4.8 there is pin-out of the board with all connections to MCU. The pin names correspond to those reported in the board datasheet.

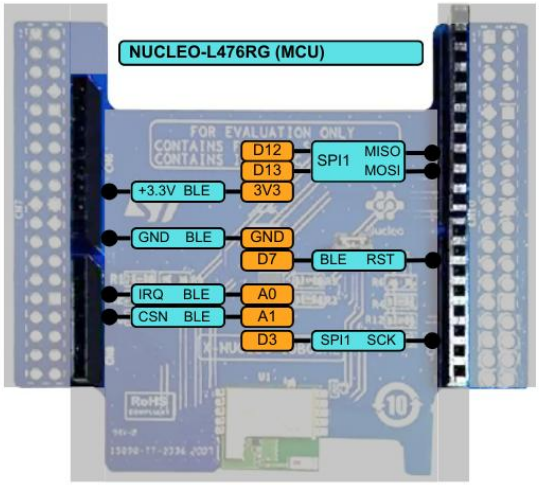

Figure 4.8. X-NUCLEO-IDB05A2 pin-out

4.1.3 MAX9867EVKIT (CODEC)

To implement the circuit required for managing the codec the MAX9867EVKIT breakout board [31] was selected. This board was developed by the same manufacturer of the MAX9867 chip, ensuring full compatibility and compliance with the recommended application design.

The board includes the power supply circuitry for the chip, the system clock essential for proper management of the digital audio signal (I^2S) and the microphone pre-amplifier. Using this evaluation kit significantly simplified the hardware integration phase and reduced the risk of design errors. An image of the board is shown in Figure 4.9.

Figure 4.9. Picture of MAX9867EVKIT (CODEC, Mic. Pre-Amp. and Clock)

Figure 4.10 shows the pinout, including the connections required to interface the board with the MCU board and the analogue peripherals (microphone and exciter).

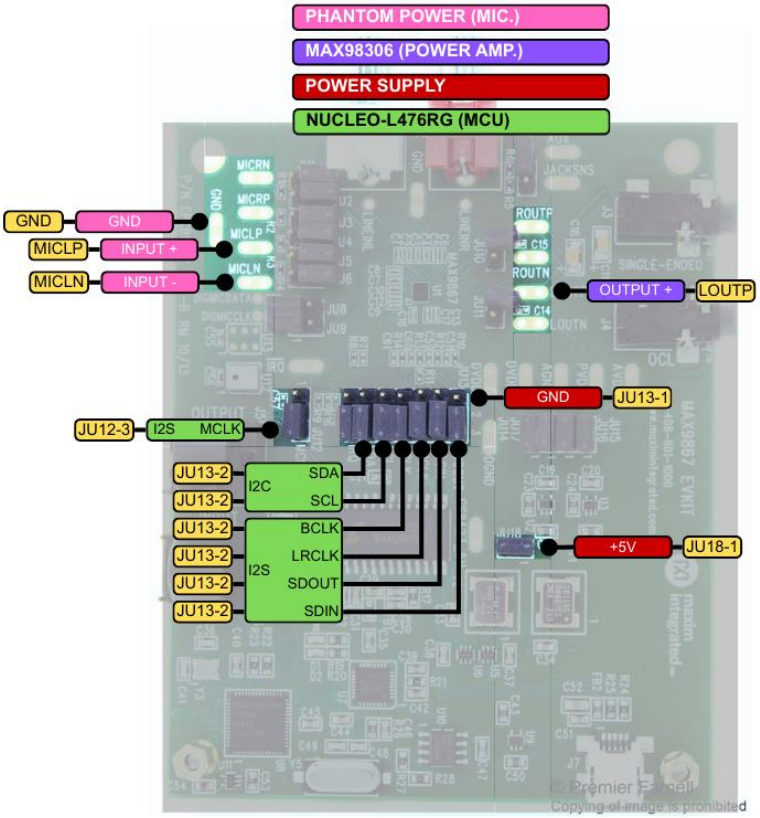

Figure 4.10. MAX9867EVKIT pin-out

To ensure the correct operation of the board several modifications were required. These concerned the clock circuitry, the pull-up resistors for the I²C interface and the jumper configuration.

The board is equipped with a 12,288 MHz oscillator which, according to the calculations performed during the design phase, complies with the required clock specifications. In the original configuration the oscillator is managed by an on-board control chip that allows configuration via USB using dedicated PC-software. To avoid the need for external software control hardware-level modifications were implemented to bypass this management chip and directly provide the required clock signal. The schematic in Figure 4.11 illustrates the modifications applied to the original circuit [31]. The hardware connections to be made are shown in red and the clock signal flow arriving at pin JU12 in green.

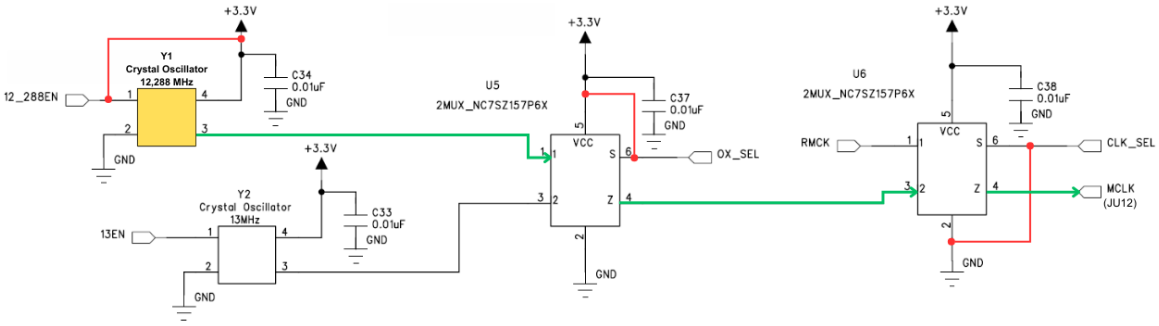

Figure 4.11. Schematic of audio clock on MAX9867EVKIT board with the changes made

To enable the I²C interface it was necessary to solder two 1.5 kΩ pull-up resistors on pads R15 and R16. This change is indicated in the board datasheet. The existing resistors are only used when the on-board control chip is active. Therefore, external pull-ups were required in the modified configuration. This modification ensures proper communication with the MCU.

Finally, the board includes several configuration jumpers that must be set according to the intended use. Table 4.1 summarizes the jumper positions required for correct operation in the proposed configuration.

Jumper	Configuration
JU1 – JU9	Open
JU10, JU11	1-2
JU12	2-3
JU13	Open
JU14 – JU17	1-2
JU18	Open

Table 4.1. Jumper position of MAX9867EVKIT board

4.1.4 MAX98306 (Power amplifier)

To implement the circuit required for the proper operation of the MAX98306 power amplifier an Adafruit breakout board was selected. Following Figure 4.12 shows the board used in the prototype.

Figure 4.12. Picture of MAX98306 (Power Amplifier) breakout board

This board is responsible to drive the exciter. The gain control is hardware-based only for this reason the amplifier gain was kept at the minimum value (+6 dB), while the CODEC's gain was adjusted via I²C. As stated in the datasheet, if no jumper is installed, the amplifier operates in the minimum-gain configuration. The following Figure 4.13 shows the board pinout along with the required connections for proper operation.

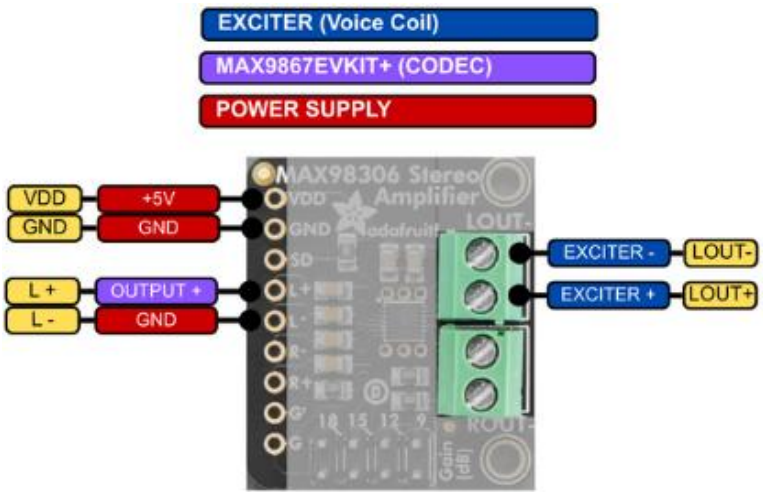

Figure 4.13. MAX98306 breakout board pin-out

4.1.5 Phantom power

The phantom power circuit is one of the two custom-built circuit implemented on a Veroboard. This circuit was designed in section 3.3.7 in the design chapter. It provides the necessary +48 V supply to power the condenser microphone. Figure 4.14 shows a picture of the implemented circuit.

Figure 4.14. Picture of phantom power circuit on Veroboard

Figure 4.15 shows the board pinout along with the required connections for proper operation.

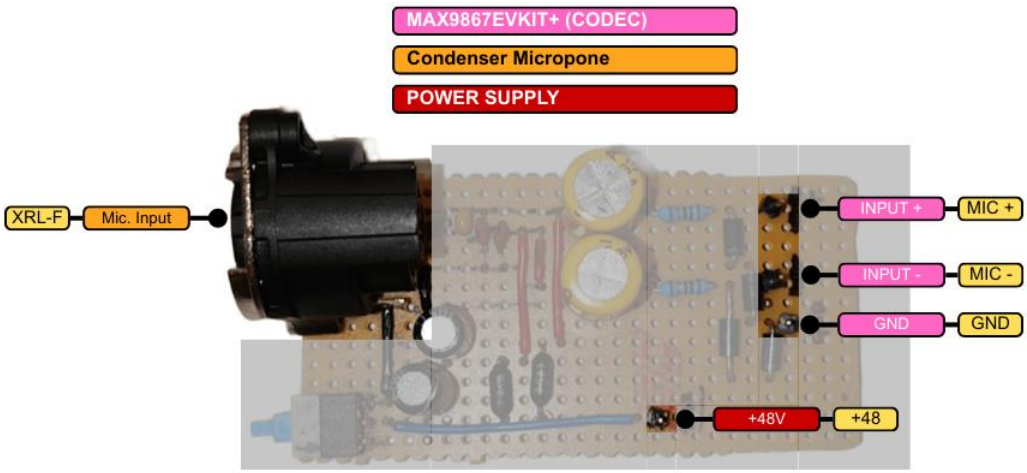

Figure 4.15. Phantom power board pin-out

4.1.6 Power supply

In this section all boards used to implement power supply circuit are illustrated. Table 4.2 shows total power consumption of all chips and devices on which we relied for the choice of the boards.

Components	Voltage [V]	Max Current [mA]	Max Power [W]
CODEC (MAX9867EVKIT)	5	12	0,06
MCU (STM NUCLEO_32L476RG)	12	800	9,6
Power Amp. (MAX98306)	5	800	4
Phantom power	48	20	9,6
Total			**23,3 [W]**

Table 4.2. Max power consumption of implemented embedded device

WPM402 (DC-DC CONVERTER STEP-UP)

To implement the step-up circuit, the WPM402 module was used. This board features an inductor that is undersized compared to the value defined in the design phase. This choice was driven by the need to use pre-developed modules and to reduce development time, as the main objective was the realization of a functional prototype.

The deviations from the original design specifications will be considered during the testing phase and carefully evaluated for future improvements and subsequent hardware revisions. Figure 4.16 shows a picture of the board.

Figure 4.16. Picture of WPM402 (Step-Up)

WPM404 (DC-DC CONVERTER STEP-DOWN)

For the step-down circuit, the WPM404 module was selected, featuring specifications adequate to meet the project requirements. The same reasoning applied to the step-up board also applies to this module. The reason is rapid prototyping using pre-existing components. Figure 4.17 shows a picture of the board.

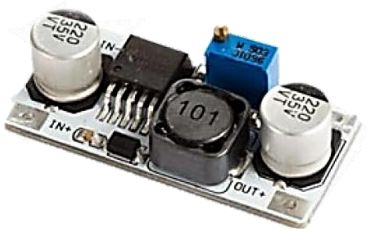

Figure 4.17. Picture of WPM404 (Step-Down)

POWER BRICK (AC/DC ADAPTER)

For the implementation of the AC to DC conversion circuit, an external power brick shows in Figure 4.18, was used. Specifically, the RS PRO 12 V DC power supply 3 A with C14 Connector.

Figure 4.18. Picture of external power brick RS PRO 12 V DC

4.2 Firmware

This paragraph describes the implementation of the firmware that was designed in sections 3.4, 3.5 and 3.6 of Chapter 3. As previously explained it was not possible to develop the complete system within the available time. The implemented functionalities include the signal generation block, communication with the CODEC, and Bluetooth communication with a smartphone. In Figure 4.19 it is possible to see the implemented firmware compared to complete firmware.

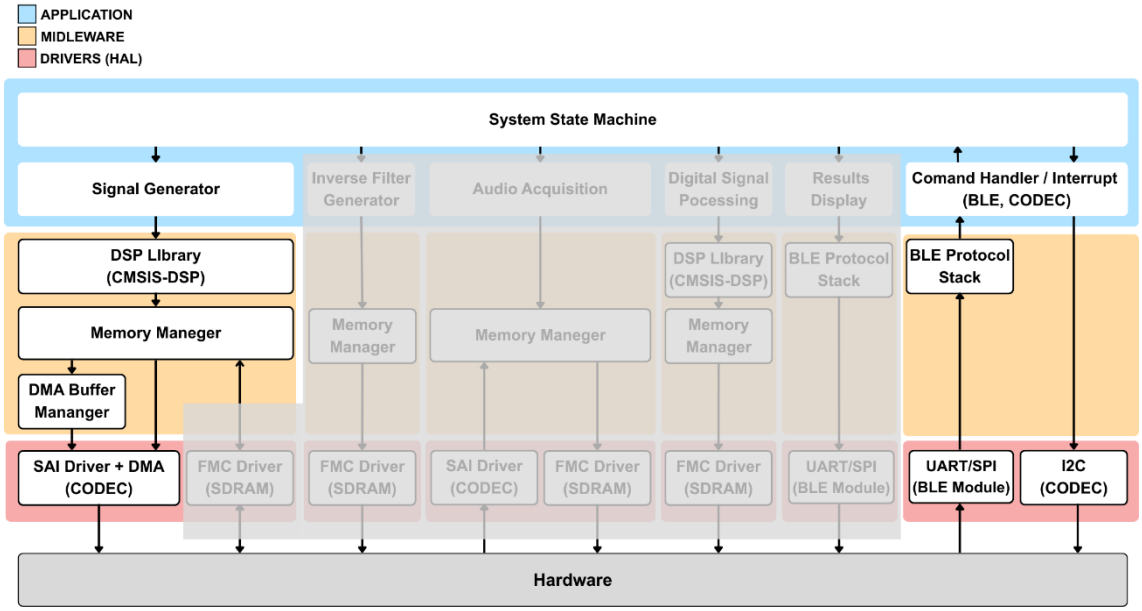

Figure 4.19. Block diagram of the firmware architecture with the implemented part highlighted

The developed firmware represents a foundational framework for future expansions and the implementation of the complete system functionality.

Firmware was developed using the STM32CubeIDE [32]. For creation of the low-level firmware components, such as hardware drivers and middleware, the STM32CubeMX [33] platform was used. This software, developed by STMicroelectronics, allows the low-level code structure to be configured through a graphical interface, simplifying code generation and reducing development time. The generated code is based on the HAL (Hardware Abstraction Layer) libraries.

For these reasons, the following sections do not focus on the low-level firmware implementation details. Instead, the discussion primarily addresses the application layer, i.e., the high-level firmware responsible for implementing the system functionalities.

4.2.1 Clock configuration on microcontroller

The system clock configuration was carried out using STM32CubeMX. As illustrated in the previous sections 3.3.3 and 3.3.5, the audio clock is generated by an external crystal oscillator (12,288 MHz) and provided to both the CODEC and the MCU in order to ensure proper synchronization of the audio data transmission.

In contrast, the clock used to manage the internal operations of the MCU is generated internally by the microcontroller.

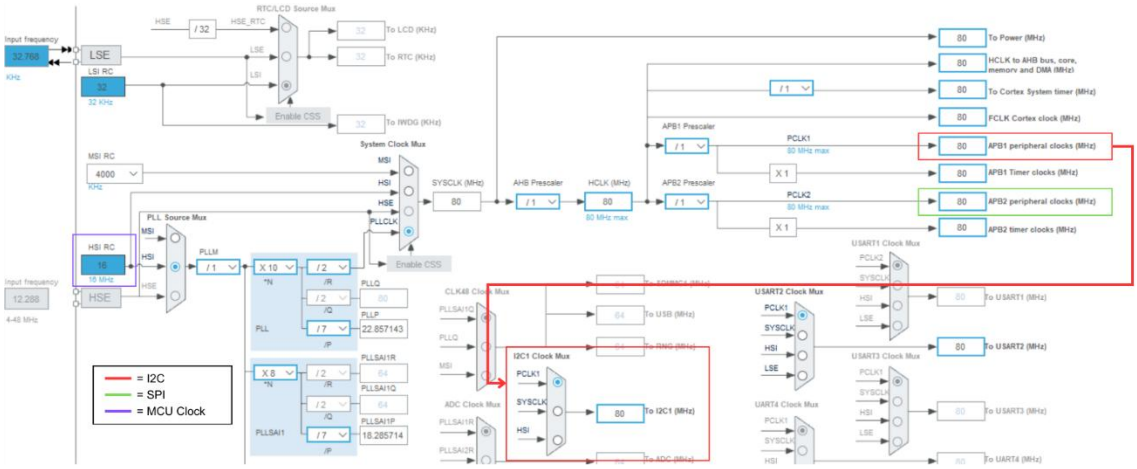

Figure 4.20. Clock configuration interface on STM32CubeMX (I^2C and SPI)

Figure 4.20 shows configuration interface where Internal MCU clock (HSI) and peripherals clock (I^2C and SPI) can be graphically defined.

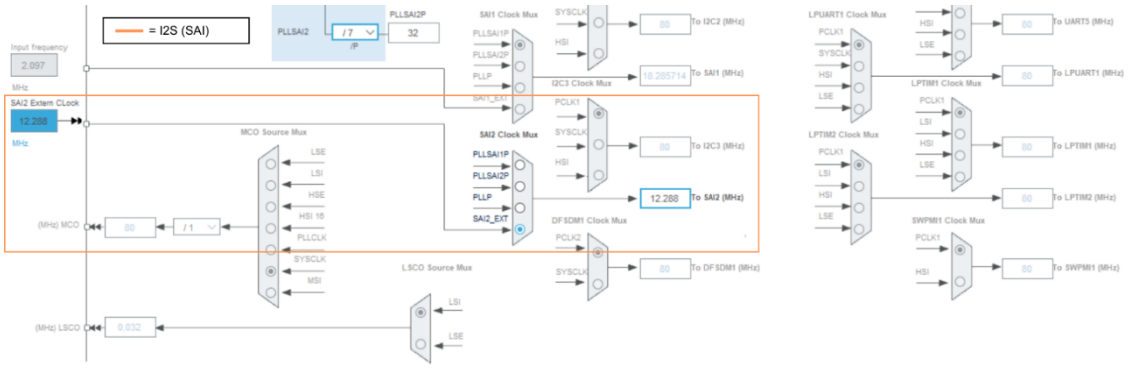

Figure 4.21. Clock configuration interface on STM32CubeMX (I^2S)

Figure 4.21 shows configuration interface where audio communication clock (I2S) can be graphically defined.

4.2.2 Code of system state machine

The state machine designed and showed in Figure 3.25 was implemented inside an infinite *while* loop ensuring continuous execution throughout the device operation. Additionally, a dedicated function called *FSM_SetNextState()* was created to handle state transitions from other source files. This approach improves code clarity, modularity, and safety by centralizing state management and preventing uncontrolled state modifications.

INIT STATE

Upon power-up the first state executed is INIT which is responsible for system initialization. The corresponding code is shown in Figure 4.22.

```
case INIT:

    // ------ LED Init -----
    init_led_I2C_Com_Status();

    // ------ I2C communication Init MAX9867 CODEC -------
    if (MAX9867_i2c_Check() != HAL_OK){     // Check i2c communication
        FSM_SetNextState(INIT);             // Retry
        break;
    }
    if (MAX9867_init() != HAL_OK){          // Initialize MAX9867 CODEC
        FSM_SetNextState(INIT);             // Retry
        break;
    }

    // ------ DMA Init -------
    for(int i=0; i<AUDIO_BUF_SIZE; i++) audio_tx[i] = 0;                                        // Audio Buffer Cleaning
    if (HAL_SAI_Transmit_DMA(&hsai_BlockA2, (uint8_t *)audio_tx, AUDIO_BUF_SIZE) != HAL_OK) {   // Start DMA
        Error_Handler();
    }

    // ------ Fade parameter Init -------
    FadeHann_Init();

    // ----- Initialization successful -----
    FSM_SetNextState(IDLE);
    break;
```

Figure 4.22.Code of INIT state in the main state machine

This state includes initialisation of I^2C peripheral (Figure 4.32), Initialization of DMA and Initialization of fade (Figure 4.29).

IDLE STATE

When all initialization procedures are successfully completed the system transitions to the IDLE state. In this state device waits for external commands. In this state is monitoring connection between MCU and CODEC. The connection is verified periodically and a blink status led on the MCU board. If the connection is lost or an error is detected system enters an error condition and wait until the connection is successfully re-established. All this part will be explained in section 4.2.5.

Finally, the background processing of Bluetooth communication, explain in section 4.2.6, is called. In the Figure 4.23 there is code to implement IDLE state.

```
case IDLE:

    MAX9867_I2C_Task();             // Periodical i2C communication check
    task_led_I2C_Com_Status();      // LED blinking if there is i2C communication
    MX_BlueNRG_MS_Process();        // Background process of Bluetooth

    break;
```

Figure 4.23. Code of IDLE state in the main state machine

SIGNAL GENERATOR STATE

To exit the IDLE state system requires an external trigger. In this implementation the command management is handled by a smartphone application communicating via Bluetooth (section 3.6.5, Figure 4.44). The instructions sent by the application are received by the system and handled as interrupts. Figure 4.24 shows code of SIGNAL_GENERATOR state, which is entered when the system receives the start command.

```
case SIGNAL_GENERATOR:

    MAX9867_Ready();
    Trigger_Generator();

    FSM_SetNextState(IDLE);
    break;
```

Figure 4.24. Code of SIGNAL_GENERATOR state in the main state machine

First call the MAX9867_Ready() function (section 3.6.3, Figure 4.33) which sets the CODEC to Ready state via I^2C communication. Finally, it calls the function *Trigger_generator()* show in Figure 4.25.

```
void Trigger_Generator(void) {
    // Only allow starting if we are currently doing nothing
    if (current_state == GEN_IDLE)
    {
        SignalGen_PlayLogSweep();
    }
    else{
        return;
    }
}
```

Figure 4.25. Code of Trigger generator function to activate the signal generation

This function subsequently calls *SignalGen_InitSineSweep() function,* shown in Figure 4.26, that initialize all sweep parameters (start and stop frequencies, duration, amplitude, and internal counters required for block-based generation) and updates generator state to GEN_PLAYING_SWEEP.

```
void SignalGen_InitSineSweep(void)
{
    s = 0.0f;                                                          //Signal Phase update
    sweep_n = 0;                                                       //Counter
    sweep_N = (uint32_t)(FS * sweep_duration);                         //Total number of Signal Sample
    sweep_sin_phi = 0.0f;                                              //Signal sine
    sweep_cos_phi = 1.0f;                                              //Signal cosine
    freq_instant = sweep_f_start;                                      //Frequency update
    freq_multiplier = powf(sweep_f_stop / sweep_f_start, 1.0f / (float)sweep_N);   //Frequency increase from F_start to F_stop
    fade_in_samples  = (uint32_t)(FS * FADE_IN_DURATION);              //Total number of fade_in Sample
    fade_out_samples = (uint32_t)(FS * FADE_OUT_DURATION);             //Total number of fade_out Sample
    amplitude = sweep_amplitude;                                       //Amplitude of the signal

    current_state = GEN_PLAYING_SWEEP;                                 //Update state
}
```

Figure 4.26. Code of *SignalGen_InitSineSweep()* function that initialize sine sweep parameters

OTHER STATE ONLY INITLISED

Finally, in Figure 4.27 the other states are shown. They are only initialized and they are ready for future implementation.

```
case AUDIO_ACQUISITION:
{
    FSM_SetNextState(INVERSE_FILTER_GENERATOR);
    break;
}

case INVERSE_FILTER_GENERATOR:
{
    FSM_SetNextState(DIGITAL_SIGNAL_PROCESSING);
    break;
}

case DIGITAL_SIGNAL_PROCESSING:
{
    FSM_SetNextState(RESULT_DISPLAY);
    break;
}

case RESULT_DISPLAY:
{
    FSM_SetNextState(IDLE);
    break;
}
```

Figure 4.27. Code of other state initialized of the main state machine

4.2.3 Sine-Sweep test signal generator code

The signal generator designed in the section 3.6.1 is now implemented. The generator is activated by the function *SignalGen_InitSineSweep()* shown in section 4.2.2. The function *Signal_Generator()* does not need to be explicitly called, as it is executed in a loop by the DMA callback, which will be presented in the following section 4.2.4.

```
void Signals_Generator(int16_t *pBuffer, uint16_t size_elements){

    switch (current_state) {

        case GEN_IDLE:

            memset(pBuffer, 0, size_elements * sizeof(uint16_t));   //Generate a buffer with 0 (silence)
            break;

        case GEN_PLAYING_SWEEP:

            for (uint16_t i = 0; i < size_elements; i += 2)
            {
                if (sweep_n >= sweep_N)                                    SIGNAL GENERATION IS FINISH
                {
                    current_state = GEN_STOP;                      //Go to GEN_STOP state
                    for (; i < size_elements; i++)                 //when signal generation is finish
                        pBuffer[i] = 0;
                    return;
                }
                // -------  Sine Sweep Logic --------                      SINE SWEEP GENERATOR
                float dphi = 6.283185307179586f * freq_instant / FS;
                float sin_d = sinf(dphi);
                float cos_d = cosf(dphi);

                float sin_new = sweep_sin_phi * cos_d + sweep_cos_phi * sin_d;  //Generate new phase
                float cos_new = sweep_cos_phi * cos_d - sweep_sin_phi * sin_d;

                sweep_sin_phi = sin_new;
                sweep_cos_phi = cos_new;
                freq_instant *= freq_multiplier;

                s = sweep_sin_phi;                                      //Update Phase

                // ---- fade in ----                                    FADE-IN / FADE-OUT
                if (sweep_n < fade_in_samples)
                {
                    uint32_t idx = (sweep_n * FADE_LUT_SIZE) / fade_in_samples;
                    if (idx >= FADE_LUT_SIZE) idx = FADE_LUT_SIZE - 1;
                    s *= fade_hann_lut[idx];                                //Apply fade-in on signal
                }
                // ---- fade out ----
                else if (sweep_n > (sweep_N - fade_out_samples))
                {
                    uint32_t n = sweep_n - (sweep_N - fade_out_samples);
                    uint32_t idx = (n * FADE_LUT_SIZE) / fade_out_samples;
                    if (idx >= FADE_LUT_SIZE) idx = FADE_LUT_SIZE - 1;
                    s *= fade_hann_lut[FADE_LUT_SIZE - 1 - idx];            //Apply fade-out on signal
                }

                // ---- Output Stereo Buffer ----                                BUFFER FILL
                int16_t out = (int16_t)(s * amplitude);                 //Buffer fill
                pBuffer[i]     = out;                                   //Left
                pBuffer[i + 1] = 0;                                     //Right

                sweep_n++;                                              //Update counter
            }
            break;

        case GEN_STOP:

            memset(pBuffer, 0, size_elements * sizeof(uint16_t));   //Reset signal
            current_state = GEN_IDLE;                               //Return to IDLE state
            break;
    }
}
```

Figure 4.28. Code of signal generator function

Let's now move on to the illustration of the code present in Figure 4.28. The states of the function are contained in the red rectangles; in the blue rectangles we find the code that generate the Sine-Sweep.

The initial state is GEN_IDLE, in which a buffer filled with zeros is produced, corresponding to silence at the codec output. In this condition, the system waits for an external trigger to start the test signal generation.

When the trigger command is received and processed by the main state machine, the generator state transitions to GEN_PLAYING_SWEEP. In this phase, the exponential Sine-Sweep samples are computed block by block and streamed to the codec through the previously described, in the section 3.6.4, double-buffering DMA mechanism.

Once the entire sweep has been generated and transmitted, the function transitions to the GEN_STOP state, which handles the completion of the playback process. Afterward, the state machine returns to GEN_IDLE, ready for a new trigger event.

This architecture ensures modularity, non-blocking execution, and seamless integration with the DMA-based real-time streaming system.

```
void FadeHann_Init(void)
{
    for (uint32_t i = 0; i < FADE_LUT_SIZE; i++) {                    //Generate Fade LUT
        float x = (float)i / (float)(FADE_LUT_SIZE - 1);
        fade_hann_lut[i] = 0.5f * (1.0f - cosf((float)M_PI * x));
    }
}
```

Figure 4.29. Code of *FadeHann_Init()* function that initialise fade parameters

Figure 4.29 shows the function *FadeHann_Init()* that calls in INIT state of main state machine shown in Figure 4.22. This function generates a lookup table (LUT) containing precomputed Hann window coefficients these values are later applied to the generated signal to implement fade-in and fade-out envelopes. The Hann fade was designed and explained in paragraph 3.6.1.

The adoption of a LUT significantly reduces the computational load on the CPU, since the window coefficients are calculated once during initialization and stored in an array, rather than being recomputed in real time for each sample.

This approach does not compromise audio quality. The portions of the signal affected by the fade-in and fade-out are not considered in the frequency response estimation; their sole purpose is to guarantee a smooth start and end of the excitation signal, preventing spectral leakage and unwanted transients.

4.2.4 Code of memory management (DMA)

As mentioned in the previous paragraph the transmission of the generated signal is handled by the DMA callback mechanism. Specifically, each time a DMA transfer is completed (half-transfer or full-transfer event) the callback function sends buffer to CODEC.

```
// ----- DMA SAI Half Transmission Callback -----
void HAL_SAI_TxHalfCpltCallback(SAI_HandleTypeDef *hsai)
{
    if (hsai->Instance == SAI2_Block_A){
        Signals_Generator(audio_tx, AUDIO_BUF_SIZE / 2);
    }
}
// ----- DMA SAI End Transmission Callback -----
void HAL_SAI_TxCpltCallback(SAI_HandleTypeDef *hsai)
{
    if (hsai->Instance == SAI2_Block_A){
        Signals_Generator(&audio_tx [AUDIO_BUF_SIZE / 2], AUDIO_BUF_SIZE / 2);
    }
}
```

Figure 4.30. Code of DMA Callback to implement data transmission to CODEC via SAI peripheral

In Figure 4.30 it is possible to observe the two DMA callback (TX Half, TX Complete) that transmit the audio buffer to the codec via the SAI peripheral. Furthermore, as explained above, we see the call of the Signal generator function containing the machine states.

4.2.5 Code of communication with CODEC peripheral

Communication between microcontroller and audio CODEC MAX9867 was implemented using the I²C bus through the HAL drivers provided by STMicroelectronics. The firmware architecture was designed to ensure reliability and maintainability by separating the register configuration from the low-level communication functions.

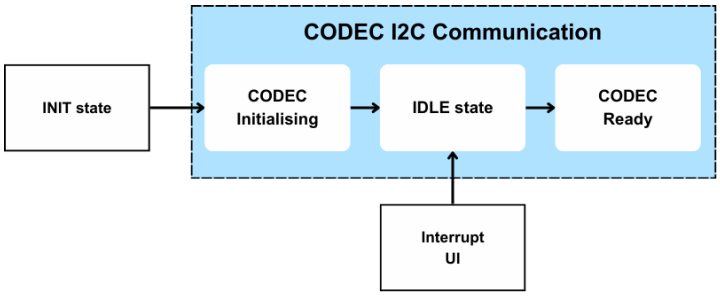

Figure 4.31. Block diagram with the firmware architecture of communication with CODEC peripheral.

Figure 4.31 shows block diagram of the firmware module responsible for managing the communication with the codec. The CODEC configuration is implemented through register configuration tables, where each entry specifies the register address and the corresponding value to be written. Two main configurations are defined: one for the initialization phase, which sets the clock parameters, the digital audio interface, and safe gain levels, and another for the ready state, used when the system starts audio transmission.

Each configuration is implemented as an array of structures *codec_reg_t*, enabling the firmware to sequentially write all the required registers without explicitly coding each individual transaction. This approach simplifies the configuration procedure, improves code readability, and facilitates future modifications or extensions of the CODEC configuration.

Figure 4.32 shows registers table to initialise CODEC before start audio transmission.

```
static const codec_reg_t codec_init_cfg[] =
{
    {MAX9867_PWRMAN, 0x00},         // Shutdown mode
    {MAX9867_INTEN, 0x00},

    {MAX9867_SYSCLK, (MAX9867_PSCLK_10_20 << MAX9867_PSCLK_SHIFT)}, // External MCLK 10-20 MHz
    {MAX9867_AUDIOCLKHIGH, 0x60},
    {MAX9867_AUDIOCLKLOW, 0x00},

    {MAX9867_IFC1A, 0x10},          // I2S communication
    {MAX9867_IFC1B, 0x00},

    {MAX9867_CODECFLTR, 0x91},      // FIR (Audio Filter) enable

    {MAX9867_SIDETONE, 0x00},       // No sidetone

    {MAX9867_DACLEVEL, 0x40},       // DAC Mute
    {MAX9867_ADCLEVEL, 0x33},       // ADC 0dB

    {MAX9867_LEFTLINELVL, 0x0f},    //L Line input Mute -6dB
    {MAX9867_RIGHTLINELVL, 0x0f},   //R Line input Mute -6dB

    {MAX9867_LEFTVOL, 0x56},        //L Line output Mute -20dB
    {MAX9867_RIGHTVOL, 0x56},       //R Line output Mute -20dB

    {MAX9867_LEFTMICGAIN, 0x14},    //L Mic. input 0dB Amp. Disabled
    {MAX9867_RIGHTMICGAIN, 0x14},   //R Mic. input 0dB Amp. Disabled

    {MAX9867_INPUTCONFIG, 0x00},    // No input is selected
    {MAX9867_MICCONFIG, 0x00},      // Base config.
    {MAX9867_MODECONFIG, 0x00},     // Base config.

    {MAX9867_PWRMAN, 0x80}          // Exit from Shutdown mode
};
```

Figure 4.32. Code of registers table to set CODEC in init status

Figure 4.33 shows registers table call to set CODEC in ready state before start audio transmission.

```
static const codec_reg_t codec_ready_cfg[] =
{
    {MAX9867_PWRMAN, 0x00},         // Shutdown mode

    {MAX9867_MODECONFIG, 0x04},     // Output set to Stereo single-ended (clickless)

    {MAX9867_DACLEVEL, 0x00},       // 0dB

    {MAX9867_LEFTVOL, 0x11},        // -10dB
    {MAX9867_RIGHTVOL, 0x11},       // -10dB

    {MAX9867_PWRMAN, 0x88}          // Enable Left DAC and Exit from Shutdown mode
};
```

Figure 4.33. Code of registers table to set CODEC in ready status

The functions that allow to write in to CODEC registers are show in Figure 4.34. Through the function *MAX9867_WriteRegisterList()* registers are updated with a list of values, such as the one shown in Figure 4.33. To increase communication reliability, a retry mechanism, implemented with the *MAX9867_WriteRetryt()* function, it allows the firmware to attempt the write operation multiple times before returning an error.

```
// Function to Write RegisterList
HAL_StatusTypeDef MAX9867_WriteRegisterList(const codec_reg_t *cfg, uint16_t size)
{
    HAL_StatusTypeDef status;
    for (uint16_t i = 0; i < size; i++)
    {
        status = MAX9867_WriteRetry(cfg[i].reg, cfg[i].val);
        if (status != HAL_OK)
            return HAL_ERROR;
    }
    return HAL_OK;
}

// ----- CODEC MAX9867 I2C Write Register Function Retry-----
HAL_StatusTypeDef MAX9867_WriteRetry(uint8_t RegAddr, uint8_t RegValue)
{
    HAL_StatusTypeDef status;

    for (uint8_t i = 0; i < MAX9867_I2C_RETRIES; i++)
    {
        status = MAX9867_i2c_Write_Register(RegAddr, RegValue);

        if (status == HAL_OK)
            return HAL_OK;
    }

    return HAL_ERROR;
}

// ----- CODEC MAX9867 I2C Write Register Function -----
HAL_StatusTypeDef MAX9867_i2c_Write_Register(uint8_t RegAddr, uint8_t RegValue)
{
    return HAL_I2C_Mem_Write(&hi2c1,
                             MAX9867_I2C_ADDR,
                             RegAddr,
                             I2C_MEMADD_SIZE_8BIT,
                             &RegValue,
                             1,
                             5);
}
```

Figure 4.34. Code of functions that allow to modify CODEC register

In addition to the register configuration procedure, firmware implements a connection monitoring mechanism to verify that the CODEC is correctly responding on the I²C bus. The function *MAX9867_i2c_Check()* shown in Figure 4.35 performs a read operation on the codec revision register. If the communication is successful and the returned value corresponds to the expected revision identifier (0x42), a status flag *i2c_codec_connected* is set, indicating that the CODEC is correctly connected to the system. Otherwise, the flag is cleared, and an error status is returned.

```
// ----- CODEC MAX9867 I2C communication check -----
HAL_StatusTypeDef MAX9867_i2c_Check(void)
{
    uint8_t revision;
    // Read revision register from MAX9867
    HAL_StatusTypeDef status = HAL_I2C_Mem_Read(&hi2c1,
                                                MAX9867_I2C_ADDR,
                                                MAX9867_REVISION,
                                                I2C_MEMADD_SIZE_8BIT,
                                                &revision,
                                                1,
                                                5);

    if ((status == HAL_OK) && (revision == 0x42))
    {
        i2c_codec_connected = 1;
        return HAL_OK;
    }

    i2c_codec_connected = 0;
    return HAL_ERROR;
}
```

Figure 4.35. Code of I²C communication check

The connection status can be accessed by other firmware modules through the function *MAX9867_i2c_IsConnected()* shown in Figure 4.36, which returns the current value of the connection flag. This abstraction allows the rest of the firmware to verify the CODEC availability without directly interacting with the low-level I²C driver.

```
// ----- CODEC MAX9867 I2C communication periodic Check -----
uint8_t MAX9867_i2c_IsConnected(void)
{
    return i2c_codec_connected;
}
```

Figure 4.36. Code of *MAX9867_i2c_IsConnected()* function that returns current value of connection flag

To avoid blocking the firmware execution, the connection verification is executed periodically by the function *MAX9867_I2C_Task()* shown in Figure 4.37. This function uses the system tick counter to perform the check every 500 ms, thus implementing a lightweight background task that continuously supervises the communication with the codec. This function is called by the IDLE state of the main state machine shown in section 4.2.2.

```
void MAX9867_I2C_Task(void)
{
    static uint32_t tick = 0;

    if (HAL_GetTick() - tick < 500)
        return;
    tick = HAL_GetTick();

    MAX9867_i2c_Check();
}
```

Figure 4.37. Code of background task that continuously supervises the communication with the CODEC.

This implementation separates the CODEC configuration logic, the I²C communication layer, and the connection monitoring mechanism, resulting in a modular and maintainable driver architecture. The use of configuration tables simplifies the register programming process, while the periodic connection check increases the robustness of the system by enabling the firmware to detect possible communication failures during operation.

4.2.6 Code of Bluetooth communication (BLE)

The firmware integrates BlueNRG BLE stack through the Host Controller Interface (HCI) and implements a custom GATT service used to exchange commands with the Android application. The firmware architecture follows an event-driven model, where BLE stack processes asynchronous events while main firmware periodically executes the BlueNRG background task.

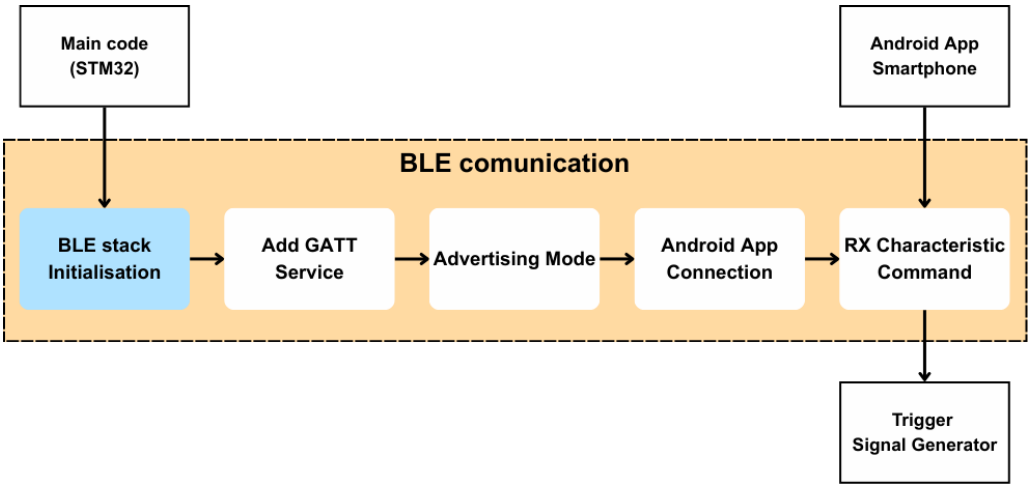

Figure 4.38. Block diagram of BLE communication firmware

In the current implementation, shown in Figure 4.38, BLE interface provides basic functionality required to establish a connection with an external device and to receive a command used to trigger the start of test signal generation. This configuration allows the embedded device to be controlled remotely through a smartphone application and represents the fundamental communication mechanism between the user interface and the measurement system. Further extensions of the communication protocol, including additional configuration parameters like set CODEC parameters shown in section 4.2.5 and data exchange functionalities like test results.

BLE STACK INITIALISATION

Initialization of the BLE subsystem is performed inside the function *MX_BlueNRG_MS_Init()*, which configures the BlueNRG device, initializes BLE stack and registers the custom GATT services. In the Initialization sequence, shown in Figure 4.39, is possible see the standard BlueNRG startup procedure. Instead in Figure 4.40 the code implemented is shown.

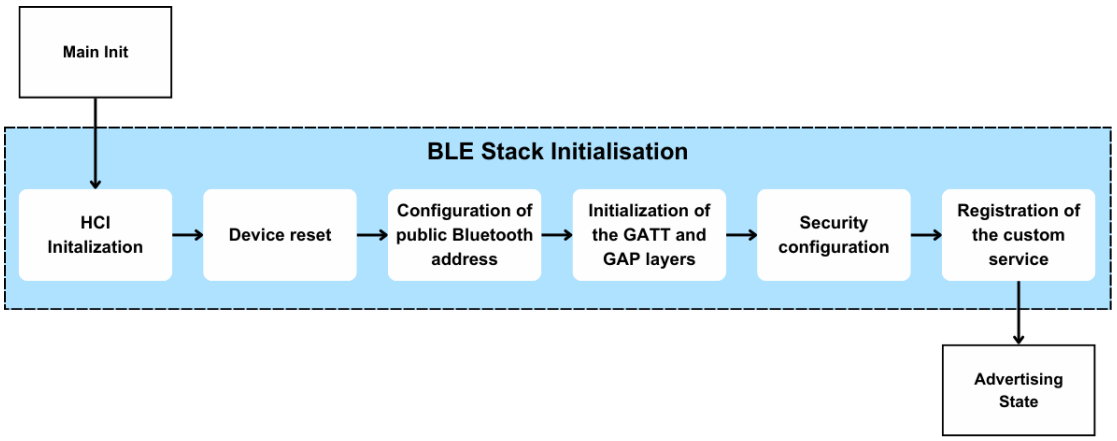

Figure 4.39. Block diagram of BLE stack initialisation firmware

```
void MX_BlueNRG_MS_Init(void)
{
  uint8_t SERVER_BDADDR[] = {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};          // Static BLE device address
  uint8_t bdaddr[BDADDR_SIZE];                                             // Buffer used to configure the BD address
  uint16_t service_handle, dev_name_char_handle, appearance_char_handle;   // Handles returned by GAP initialization
  uint8_t  hwVersion;                                                      // BlueNRG hardware version
  uint16_t fwVersion;                                                      // BlueNRG firmware version
  int ret;                                                                 // Return value for BLE API calls

// ******* User and HCI Initialization ************************************************************************//
  User_Init();                                                            // Initialize user peripherals (e.g. UART)
  hci_init(user_notify, NULL);                                            // Initialize HCI

// ******* BlueNRG Version Detection **************************************************************************//
  getBlueNRGVersion(&hwVersion, &fwVersion);                              // Read hardware and firmware versions
  printf("HWver %d, FWver %d\n", hwVersion, fwVersion);                   // Print version for debugging

// ******* BlueNRG Reset **************************************************************************************//
  hci_reset();                                                            // Reset the BlueNRG device
  HAL_Delay(100);                                                         // Wait for the device to complete the reset

// ******* Bluetooth Address Configuration ********************************************************************//
  BLUENRG_memcpy(bdaddr, SERVER_BDADDR, sizeof(SERVER_BDADDR));           // Copy predefined BD address
  ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,             // Configure public Bluetooth address
                                  CONFIG_DATA_PUBADDR_LEN,
                                  bdaddr);
  if (ret) {
    printf("Setting BD_ADDR failed 0x%02x.\n", ret);                       // Print error if configuration fails
  }

// ******* GATT Layer Initialization **************************************************************************//
  ret = aci_gatt_init();                                                  // Initialize GATT layer
  if (ret) {
    printf("GATT_Init failed.\n");                                        // Print error if initialization fails
  }

// ******* GAP Layer Initialization ***************************************************************************//
  ret = aci_gap_init_IDB05A1(GAP_PERIPHERAL_ROLE_IDB05A1,                 // Configure device as BLE Peripheral
                             0,
                             0x07,
                             &service_handle,
                             &dev_name_char_handle,
                             &appearance_char_handle);

  if (ret != BLE_STATUS_SUCCESS) {
    printf("GAP_Init failed.\n");                                         // Print error if GAP initialization fails
  }

// ******* Security Configuration *****************************************************************************//
  ret = aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED,            // Enable MITM protection
                                     OOB_AUTH_DATA_ABSENT,
                                     NULL,
                                     7,                                   // Minimum encryption key size
                                     16,                                  // Maximum encryption key size
                                     USE_FIXED_PIN_FOR_PAIRING,           // Use fixed PIN for pairing
                                     123456,                              // Fixed pairing PIN
                                     BONDING);                            // Enable bonding
  if (ret == BLE_STATUS_SUCCESS) {
    printf("BLE Stack Initialized.\n");                                   // Confirm BLE stack initialization
  }
  printf("SERVER: BLE Stack Initialized\n");

// ******* Custom Service Registration ************************************************************************//
  ret = Add_Sample_Service();                                             // Add custom GATT service used by the application

  if (ret == BLE_STATUS_SUCCESS)
  printf("Service added successfully.\n");                                // Service correctly registered
  else
  printf("Error while adding service.\n");                                // Error during service creation

// ******* Radio Transmission Power Configuration *************************************************************//
  ret = aci_hal_set_tx_power_level(1,4);                                  // Configure BLE transmission power level
}
```

Figure 4.40. Code to initialise Bluetooth BLE

An important part of initialization to enable the communication with the Android application is the custom service registration. The service contains two characteristics:

- TX characteristic → used to transmit notifications from the embedded device to the smartphone
- RX characteristic → used to receive commands from the mobile application

```
tBleStatus Add_Sample_Service(void)
{
  tBleStatus ret;                                         // Return status of BLE API calls
// UUID Definition
  const uint8_t service_uuid[16] = {0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe0,0xf2,0x73,0xd9};
  const uint8_t charUuidTX  [16] = {0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe1,0xf2,0x73,0xd9};
  const uint8_t charUuidRX  [16] = {0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe2,0xf2,0x73,0xd9};

  // ******* Custom Service Creation ***************************************************************************//
  ret = aci_gatt_add_serv(UUID_TYPE_128,                  // Service UUID type
                          service_uuid,                   // Custom service UUID
                          PRIMARY_SERVICE,                // Declare as primary GATT service
                          7,                              // Maximum number of attribute records
                          &sampleServHandle);             // Output service handle

  if (ret != BLE_STATUS_SUCCESS) goto fail;               // Abort if service creation fails

  // ******* TX Characteristic Creation ************************************************************************//
  ret =  aci_gatt_add_char(sampleServHandle,              // Parent service handle
                           UUID_TYPE_128,                 // UUID type
                           charUuidTX,                    // TX characteristic UUID
                           20,                            // Maximum payload size (bytes)
                           CHAR_PROP_NOTIFY,              // Notification property enabled
                           ATTR_PERMISSION_NONE,          // No access restriction
                           0, 16, 1,
                           &TXCharHandle);                // Output characteristic handle

  if (ret != BLE_STATUS_SUCCESS) goto fail;               // Abort if characteristic creation fails

  // ******* RX Characteristic Creation ************************************************************************//
  ret =  aci_gatt_add_char(sampleServHandle,              // Parent service handle
                           UUID_TYPE_128,                 // UUID type
                           charUuidRX,                    // RX characteristic UUID
                           20,                            // Maximum payload size (bytes)
                           CHAR_PROP_WRITE | CHAR_PROP_WRITE_WITHOUT_RESP, // Allow write operations from client
                           ATTR_PERMISSION_NONE,          // No access restriction
                           GATT_NOTIFY_ATTRIBUTE_WRITE,   // Generate event when attribute is written
                           16, 1,
                           &RXCharHandle);                // Output characteristic handle

  if (ret != BLE_STATUS_SUCCESS) goto fail;               // Abort if characteristic creation fails

  // ******* Successful Service Registration *******************************************************************//
  printf("Sample Service added.\nTX Char Handle %04X, RX Char Handle %04X\n",
         TXCharHandle, RXCharHandle);                     // Print assigned handles for debugging
  return BLE_STATUS_SUCCESS;                              // Service correctly added
  // ******* Error Handling ************************************************************************************//
fail:
  printf("Error while adding Sample Service.\n");         // Error message if any step fails
  return BLE_STATUS_ERROR;
}
```

Figure 4.41. Code to generate Custom BLE Service

Service is created using the function *Add_Sample_Service()* shown in Figure 4.41. This function is called in the communication initialization shown in Figure 4.40.

The TX characteristic uses the notification mechanism to transmit measurement data asynchronously, while the RX characteristic allows the smartphone to send control commands without requiring acknowledgment. This configuration enables low-latency bidirectional communication, which is particularly suitable for measurement control applications.

ADVERTISING AND DEVICE CONNECTION

Once the BLE stack is initialized, the device enters advertising mode to allow external devices to establish a connection. This procedure is implemented in the function *Make_Connection()* calls by *MX_BlueNRG_MS_Process()* running in the background.

```
void Make_Connection(void)
{
  tBleStatus ret;                                         // Return status of BLE API calls

  // ******* Device Name Definition ********************************************************************************//
  const char local_name[] = {AD_TYPE_COMPLETE_LOCAL_NAME,          // Advertising data type: complete device name
                             'E','S','S',' ','D','E','V','I','C','E'}; // BLE device name shown to the client

  // ******* Scan Response Configuration ***************************************************************************//
  hci_le_set_scan_resp_data(0,NULL);                       // Disable scan response data

  // ******* Advertising Configuration *****************************************************************************//
  printf("General Discoverable Mode ");
  ret = aci_gap_set_discoverable(ADV_DATA_TYPE,                    // Advertising type
                                 ADV_INTERV_MIN,                   // Minimum advertising interval
                                 ADV_INTERV_MAX,                   // Maximum advertising interval
                                 PUBLIC_ADDR,                      // Use public device address
                                 NO_WHITE_LIST_USE,                // Accept connection from any device
                                 13,                               // Length of advertising data
                                 local_name,                       // Device name included in advertising packet
                                 0,                                // No service UUID included
                                 NULL,
                                 0, 0);                            // No slave connection interval

  printf("%d\n",ret);                                      // Print return status for debugging
}
```

Figure 4.42. Code to configure the BLE advertising parameters and makes the device discoverable

The function *Make_Connection()* shown in Figure 4.42 configures advertising parameters and makes the device discoverable, allowing an Android application to establish a connection.

EVENT-DRIVEN COMMUNICATION MANAGEMENT

BLE communication, like connection to android app, is managed through asynchronous events generated by the BlueNRG stack. These events are processed inside the callback function *user_notify().*

The function parses the received HCI packets and dispatches them to the appropriate handlers.

This code shown in Figure 4.43 allows the firmware to react to events such as:

- Connection establishment
- Device disconnection
- Characteristic write operations
- BLE notifications

```
void user_notify(void * pData)
{
    // ******* HCI Packet Extraction ********************************************************************************//
    hci_uart_pckt *hci_pckt = pData;                                  // Pointer to the received HCI UART packet
    hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;     // Extract HCI event packet from UART frame

    // ******* Packet Type Verification *****************************************************************************//
    if(hci_pckt->type != HCI_EVENT_PKT)                               // Process only HCI event packets
    return;


    // ******* HCI Event Dispatcher *********************************************************************************//
    switch(event_pckt->evt){

        // ******* Disconnection Event ******************************************************************************//
        case EVT_DISCONN_COMPLETE:
            {
              GAP_DisconnectionComplete_CB();                             // Handle BLE disconnection event
            }
            break;

        // ******* LE Meta Events ***********************************************************************************//
        case EVT_LE_META_EVENT:
            {
              evt_le_meta_event *evt = (void *)event_pckt->data;          // Extract LE meta event structure

              switch(evt->subevent){

                  // ******* Connection Complete Event ******************************************************************//
                  case EVT_LE_CONN_COMPLETE:
                    {
                      evt_le_connection_complete *cc = (void *)evt->data;        // Extract connection parameters
                      GAP_ConnectionComplete_CB(cc->peer_bdaddr, cc->handle);    // Call connection callback
                    }
                    break;
              }
            }
        break;

        // ******* Specific Events **********************************************************************************//
        case EVT_VENDOR:
            {
              evt_blue_aci *blue_evt = (void*)event_pckt->data;           // BlueNRG specific event

              switch(blue_evt->ecode){

                  // ******* GATT Attribute Modified Event **************************************************************//
                  case EVT_BLUE_GATT_ATTRIBUTE_MODIFIED:
                    {
                        evt_gatt_attr_modified_IDB05A1 *evt =
                        (evt_gatt_attr_modified_IDB05A1*)blue_evt->data;  // Extract attribute modification event
                        Attribute_Modified_CB(evt->attr_handle,           // Handle write operation on characteristic
                                              evt->data_length,
                                              evt->att_data);
                    }

                    // ******* GATT Notification Event ********************************************************************//
                  case EVT_BLUE_GATT_NOTIFICATION:
                    {
                      evt_gatt_attr_notification *evt =
                      (evt_gatt_attr_notification*)blue_evt->data;              // Extract notification event
                      GATT_Notification_CB(evt->attr_handle,                   // Forward notification to handler
                                           evt->event_data_length - 2,
                                           evt->attr_value);
                    }
                    break;
              }
            }
            break;
    }
}
```

Figure 4.43. Code to build the event-driven communication management BLE

COMMAND HANDLING FROM THE MOBILE APPLICATION

Commands sent by the Android application are received through the RX characteristic and processed inside the function *receiveData()*. This function connects the Bluetooth communication interface with the firmware responsible for the sound generation and the configuration of the CODEC peripheral parameters. However, the latter functionality has not been implemented in the current version of the system and is left for future developments.

```
void receiveData(uint8_t* data_buffer, uint8_t Nb_bytes)
{
  if (Nb_bytes > 0) {
    if (data_buffer[0] == 0x01) {
        FSM_SetNextState(SIGNAL_GENERATOR);  //Start signal generator
    }
    else {
      printf("Unrecognized command: 0x%02X\n", data_buffer[0]);
    }
  }
}
```

Figure 4.44. Code to manage receive data from smartphone application

In Figure 4.44 it is possible to see the *receiveData()* function, which receiving the byte 0x01 from the smartphone application corresponds to a transition to the SIGNAL_GENERATOR state, thereby activating the signal generation process shows in section 4.2.3.

BLE BACKGROUND PROCESSING

The BLE stack requires periodic processing of pending events to correctly manage advertising, connections, and data exchange. For this reason, the function *MX_BlueNRG_MS_Process()*, shown in Figure 4.45, is executed continuously within the main firmware loop (IDEL state, Figure 4.23) described in section 4.2.2.

Inside this routine, the *User_Process()* function calls *Make_Connection()*, shown in Figure 4.42, which manages the device visibility and the establishment of BLE connections. By executing this background processing within the main loop, the device remains discoverable and can handle connection requests and data transfers with external devices.

```
void MX_BlueNRG_MS_Process(void)
{
  User_Process();           //Function that call make connection()
  hci_user_evt_proc();      //Function that process HCI events
}
```

Figure 4.45. Code of BLE background processing function

4.3 3D Modelling and final assembling

Finally, a custom enclosure was designed and prototyped to house all the system components, serving both functional and demonstrative purposes. The mechanical design was developed using SolidWorks, and the enclosure was fabricated in PLA using a Bambu Lab 3D printer. Figure 4.46 shows the rendered views of the enclosure and its cover obtained from the CAD model.

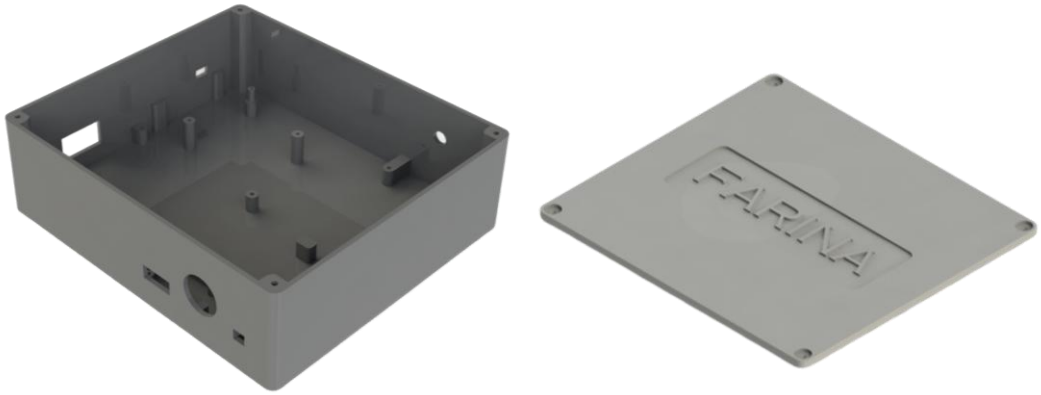

Figure 4.46. 3D modelling of case and cover designed with SolidWorks

Figure 4.47 shows the assembled device with the printed parts. The device has been named after Angelo Farina in recognition of his fundamental contributions to acoustic measurements based on the ESS (Exponential Sine-Sweep) method. Without these studies, the development of this device would not have been possible.

Figure 4.47. Picture of embedded device “FARINA”

Figure 4.48, Figure 4.49 and Figure 4.50 show images of the inside of the device from different angles and Input/Output connectors.

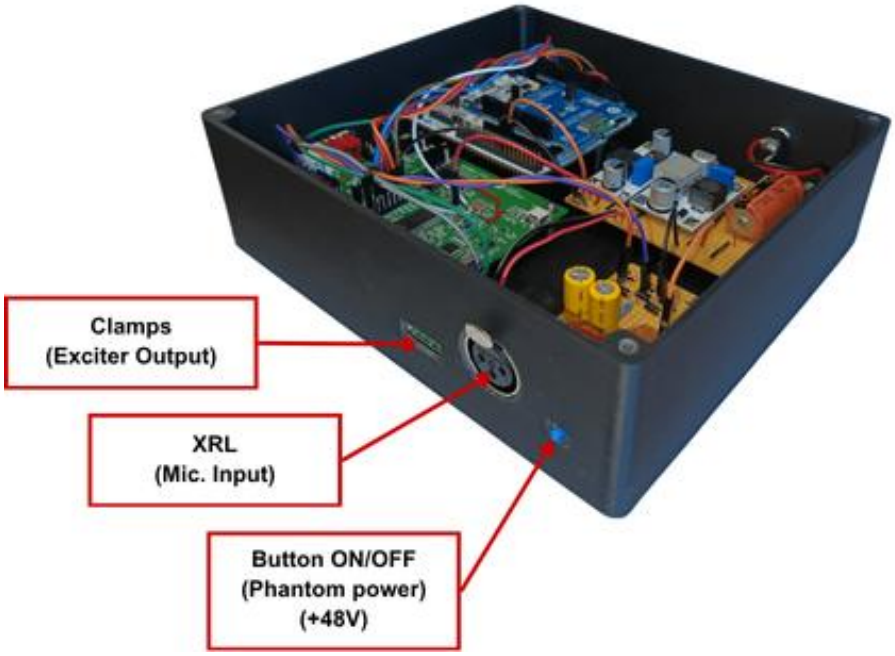

Figure 4.48. Interior picture in the left view of the embedded device

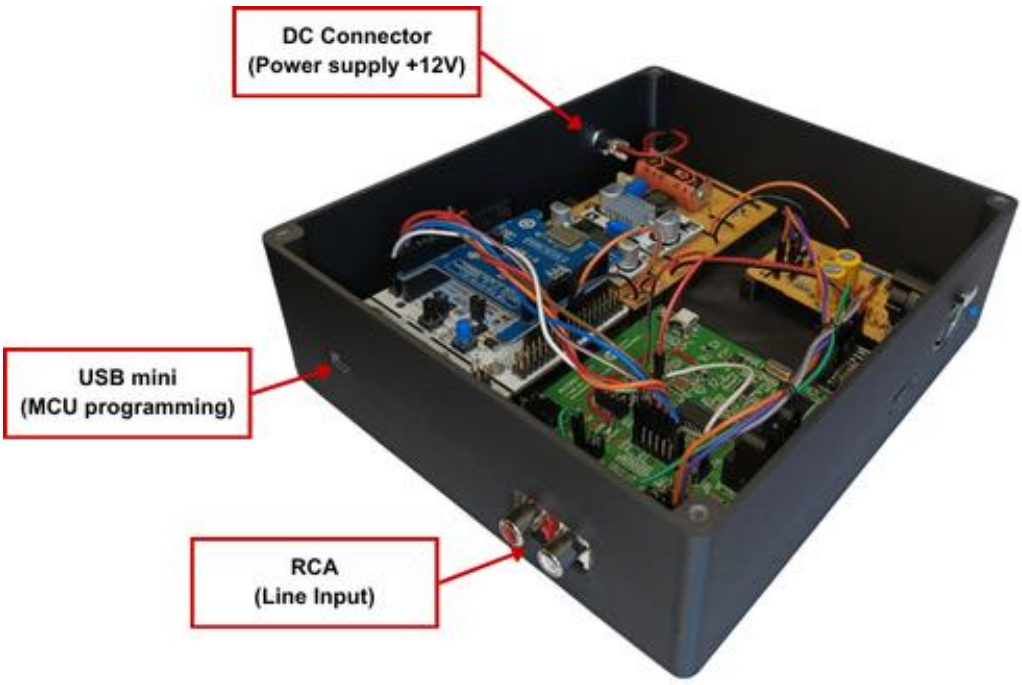

Figure 4.49. Interior picture in the right view of the embedded device

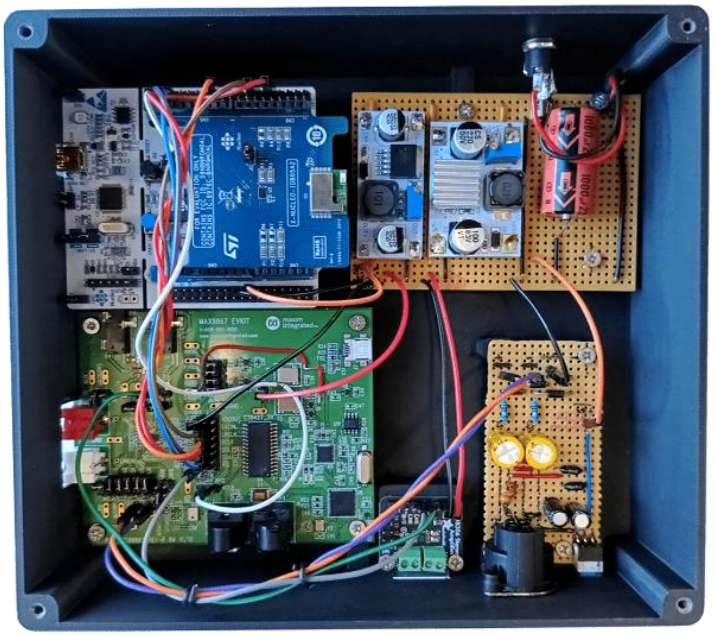

Figure 4.50. Interior picture in the up view of the embedded device

5 VALIDATION AND TEST

This chapter presents the tests carried out to evaluate the performance and quality of the measurements that can be performed with the developed device. Obtained results are then compared with target performances defined in Table 3.3 and measurements performed with the current method.

Four different tests were carried out. The first test aims to evaluate performance of the developed device in terms of signal-to-noise ratio (SNR) and harmonic distortion (THD) by means of the SINAD method.

The second test verifies the quality of the signal used for the measurements, the Sine-Sweep signal. The test is performed by comparing signal generated using the standard method with the one generated by the developed device, analysing their respective frequency responses.

The third test focuses on the analysis of the power supply. In this case, an oscilloscope is used to evaluate the ripple present in the different power supply stages.

The fourth and final test consists of measuring the frequency response of a musical instrument, comparing the results obtained with the developed device with those obtained using the current reference method. This last test provides a comprehensive and accurate evaluation of efficiency of the implemented device.

5.1 Embedded device performance tests

This paragraph presents tests carried out to evaluate the quality of developed device in accordance with AES17. Tests will be performed to evaluate the noise signal ratio (SNR), the total harmonic distortion (THD) and the Jitter. These results are compared with those obtained using the device currently employed, to provide a direct assessment of the device's performance. For evaluation of SINAD and THD, a single-tone sinusoidal signal at 1 kHz will be used, while for the assessment of jitter, a single-tone sinusoidal signal at 12 kHz will be employed.

5.1.1 Tests setup

A fundamental aspect to consider before performing the tests is the correct set-up of instrumentation. This includes a correct set of gain levels of signal to obtain reliable results and define connection between devices.

The first setup shown in Figure 5.1, concerns performance tests on the developed device, where test signal was generated directly by the Farina device. The signal is recorded using Adobe Audition through an Audient EVO 16 audio interface (Line Input) and saved as a .wav file. The recorded signal was then processed in MATLAB, where the mathematical routine required to compute the SINAD [34], THD and Jitter [34] values were implemented.

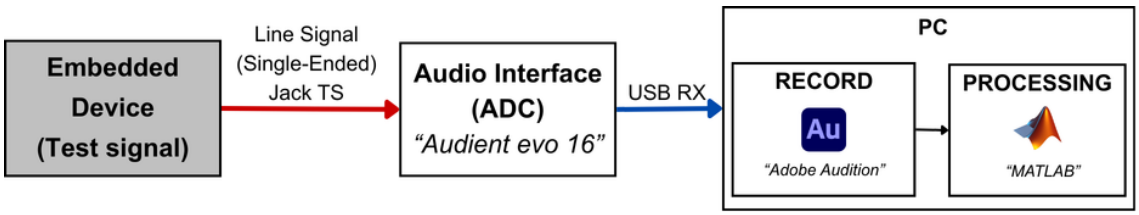

Figure 5.1. Block diagram of the setup used to run the performance tests on the developed device

As shown in Figure 5.1, test signals are generated by the microcontroller. Figure 5.2 shows the code developed to implement signal generation. This code has been implemented within the signal generator function presented in Figure 4.28.

```
case GEN_PLAYING_TONE:

    // --------------- Tone Signal Generation --------------//
    for (uint16_t i = 0; i < size_elements; i += 2 ){


        if (tone_n >= tone_N){                  //Stop generator after duration
            current_state = GEN_STOP;
            for (; i < size_elements; i++)      //Fill the remaining part of the buffer with zeros
                pBuffer[i] = 0;
            return;
        }

        tone_phase += phase_inc;                //Increment the phase of the sinusoidal oscillator.

        if (tone_phase >= 2.0f * M_PI){         // Phase wrapping to keep the phase within [0 , 2π]
            tone_phase -= 2.0f * M_PI;          // This prevents numerical overflow during long executions.
        }
        s = sinf(tone_phase);                   //Compute the sine value corresponding to the current phase

        int16_t out = (int16_t)(s * amplitude); // Scale the normalized sine value by the desired amplitude

        //----- Write the generated sample into the output buffer -----//
        pBuffer[i] = out;       // Left channel receives the tone
        pBuffer[i+1] = 0;       // Right channel is set to zero (mono tone on left channel)

        tone_n++;                               // Increment the generated sample counter
    }
    break;
```

Figure 5.2. Code to generate 1 kHz sinusoidal tone on STM32 MCU

Figure 5.3 shows the function used to initialize the parameters of the 1 kHz sinusoidal tone and 12 kHz sinusoidal tone. This function is implemented within the signal generator file, which also contains the other initialization routines described in section 4.2.3 of Chapter 4 implementation.

```
void SignalGen_PlayTone(void)
{
    s = 0.0f;                                   //Initialize the instantaneous signal value
    tone_n = 0;                                 //Reset the generated sample counter
    tone_N = (uint32_t)(FS * tone_duration);    //Compute the total number of samples that will compose the tone
    tone_phase = 0.0f;                          //Initialize the oscillator phase
    phase_inc = 2.0f * M_PI * tone_f / FS;      // Compute the phase increment for each sample.
    amplitude = tone_amplitude;                 // Set the tone amplitude
    current_state = GEN_PLAYING_TONE;           // Set the generator state to start tone playback
}
```

Figure 5.3. Code of function that initialise parameters of 1 kHz sinusoidal tone on STM32 MCU

About signal level, MCU generate signal at the maximum possible amplitude for a 16-Bit full-scale signal (32767), corresponding to 0 dB_{FS}. This choice ensures the best possible SNR.

Codec plus Power amplifier, with gain set to zero, produces a line-level output measured with a tester of $\sim 1V_{RMS}$, which corresponds ~ 2.2 dBu, as shows in equation (17).

$$Line\ out\ (Device) = 20log_{10}\left(\frac{V_{RMS}}{0.775}\right) = 20log_{10}\left(\frac{1}{0.775}\right) = 2.2\ \text{dBu} \qquad (19)$$

This value is consistent with devices operating at 0 dB_{FS} output level. When acquired through the audio interface with the gain set to 0 dB, the signal exhibits an amplitude of approximately $\sim -17\ \mathrm{dB}_{FS}$.

Considering a headroom of -6 dB_{FS} to avoid clipping during recording, the gain set on the audio interface is:

$$Gain\ (Audio\ interface) = 17\ dB - 6\ dB = 11\ dB \tag{20}$$

Second set-up shows in Figure 5.4, concerns performance tests on the currently used equipment, where signals test were generated using Adobe Audition and sent to the audio interface (Line Output). The signals were then looped back, with jack TRS cable, to the input of the interface (Line Input), recorded again, and processed within the same software environment. In this way, the quality of the DAC of the audio interface could be evaluated and subsequently compared with that of the developed device.

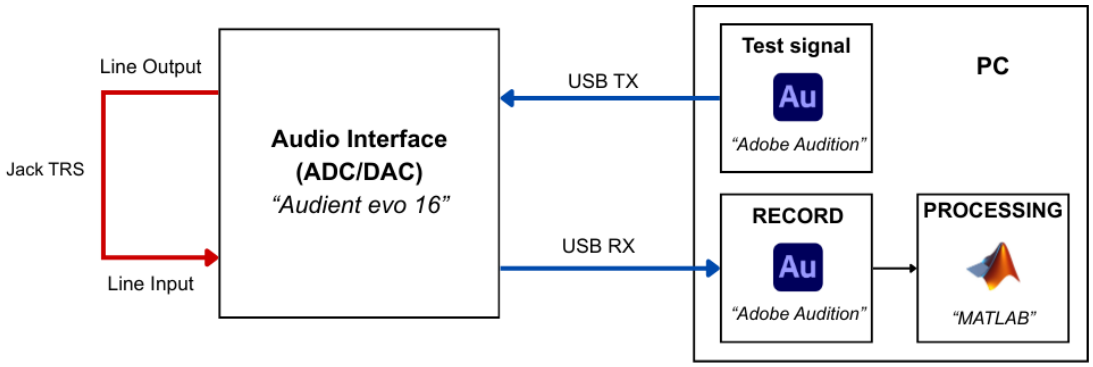

Figure 5.4. Block diagram of the set-up used for performing the comparison of performance tests on the currently used equipment

About signal level, it has a level of -6 dB_{FS}. Audio interface gain is set to 0 dB. Figure 5.5 shows audition plug-in for 1kHz sinusoidal tone signal generation with set parameters.

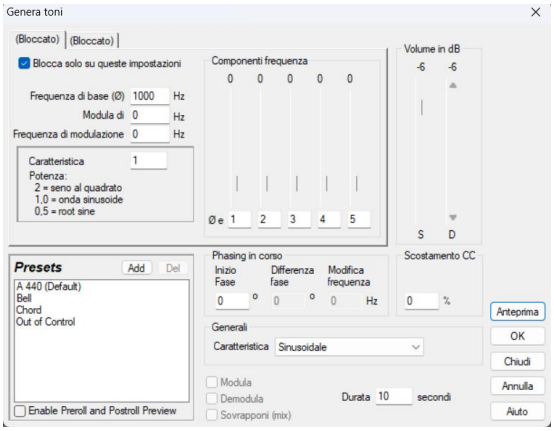

Figure 5.5. Audition plug-in to generate a 1 kHz sinusoidal tone

Figure 5.6 shows MATLAB function developed to implement SINAD and THD tests on computer. The algorithm operates on a recorded signal and performs a spectral analysis using a configurable FFT size, like professional measurement software such as audio analysers or acoustic measurement tools.

```
function results = SINAD(device,fs)
device = device(:);

% Take 8 central seconds
window_sec = 8;
Nwin = round(window_sec*fs);
Ntot = length(device);

if Ntot < Nwin
    error("signal too short")
end

center = round(Ntot/2);
start_idx = center - round(Nwin/2);
end_idx   = center + round(Nwin/2) - 1;

device = device(start_idx:end_idx);
N = length(device);

% Remove DC
device = device - mean(device);

% Hann window
w = hann(N);
xw = device .* w;

% FFT
X = fft(xw);
f = (0:N-1)*(fs/N);
halfN = floor(N/2);

% Find fundamental frquency
[~,idx] = max(abs(X(2:halfN)));
idx = idx + 1;
f0 = f(idx);

% NOTCH FILTER
bw = round(5*N/fs); % NOTCH width
X_notch = X;
i1 = max(1, idx - bw);
i2 = min(N, idx + bw);
X_notch(i1:i2) = 0;  % fondamental remove

% Residual power calculation
residual_power = sum(abs(X_notch(1:halfN)).^2);
signal_power = abs(X(idx))^2;

%% ----- SINAD -----
SINAD = 10*log10(signal_power / residual_power);

%% ----- THD ------
harmonics = 8;
harm_freq = zeros(harmonics,1);
harm_level = zeros(harmonics,1);
thd_power = 0;
for k = 1:harmonics
    fk = k*f0;
    if fk > fs/2
        break
    end
    idx_h = round(fk*N/fs);
    harm_freq(k) = fk;
    harm_level(k) = 20*log10(abs(X(idx_h))/abs(X(idx)));

    if k > 1
        thd_power = thd_power + abs(X(idx_h))^2;
    end
end
THD = 10*log10(thd_power/abs(X(idx))^2);
```

Figure 5.6. Code of MATLAB function to implement SINAD and THD tests

A central signal portion of 8 seconds are selected. The power of the fundamental is compared to the residual power (noise + distortion) to compute SINAD. In addition, THD of the signal was also calculated.

```
function jitter_struct = Jitter(file_path, Nfft, sideband_range)
[x, fs] = audioread(file_path);

% Select central segment
Ntot = length(x);
if Ntot < Nfft
    error('Signal shorter than FFT size');
end
center = round(Ntot/2);
start_idx = center - floor(Nfft/2);
end_idx   = center + floor(Nfft/2) - 1;
x_seg = x(start_idx:end_idx);


% Remove DC and apply Hann window
x_seg = x_seg - mean(x_seg);
w = hann(Nfft);
xw = x_seg .* w;

% FF
X = fft(xw, Nfft);
f = (0:Nfft-1)*(fs/Nfft);
halfN = floor(Nfft/2);
magX = abs(X(1:halfN));


% Find fundamental
[~, idx] = max(magX(2:end));  % ignore DC
idx = idx + 1;
f0 = f(idx);
A_fund = magX(idx);

% Find maximum sidebands in range
bin_range = round(sideband_range * Nfft / fs);

% Positive sideband
i1 = idx+1;
i2 = min(idx+bin_range, halfN);
A_sb_pos = max(magX(i1:i2));

% Negative sideband
i1 = max(idx-bin_range,1);
i2 = idx-1;
A_sb_neg = max(magX(i1:i2));

% RMS jitter (seconds)
t_jitter_pos = A_sb_pos / (2*pi*f0);
t_jitter_neg = A_sb_neg / (2*pi*f0);
t_jitter_avg = mean([t_jitter_pos, t_jitter_neg]);

% Convert to picoseconds
jitter_ps = t_jitter_avg * 1e12;
```

Figure 5.7. Code of MATLAB function to implement Jitter test

Figure 5.7 shows MATLAB function developed to implement Jitter test on computer.

5.1.2 Tests results

Table 5.1 shows results of the tests performed on embedded device.

Characteristic	Value	Unity
Fundamental	1000	Hz
SINAD	63	dB
THD	0.02	%
Jitter	100	pSrms

Table 5.1. Performance tests results of embedded device (SINAD, THD and Jitter)

Table 5.1 Shows results of the tests performed on reference Audient Evo.

Characteristic	Value	Unity
Fundamental	1000	Hz
SINAD	96	dB
THD	0.0003	%
Jitter	33	pSrms

Table 5.2. Performance tests results of currently used device (SINAD, THD and Jitter)

Figure 5.8 shows plot of frequency responses (SINAD 1 kHz sinusoidal tone) of the two analysed devices, developed embedded system and reference device. Graph is presented to allow a visual comparison of their performance and highlighting the differences between them.

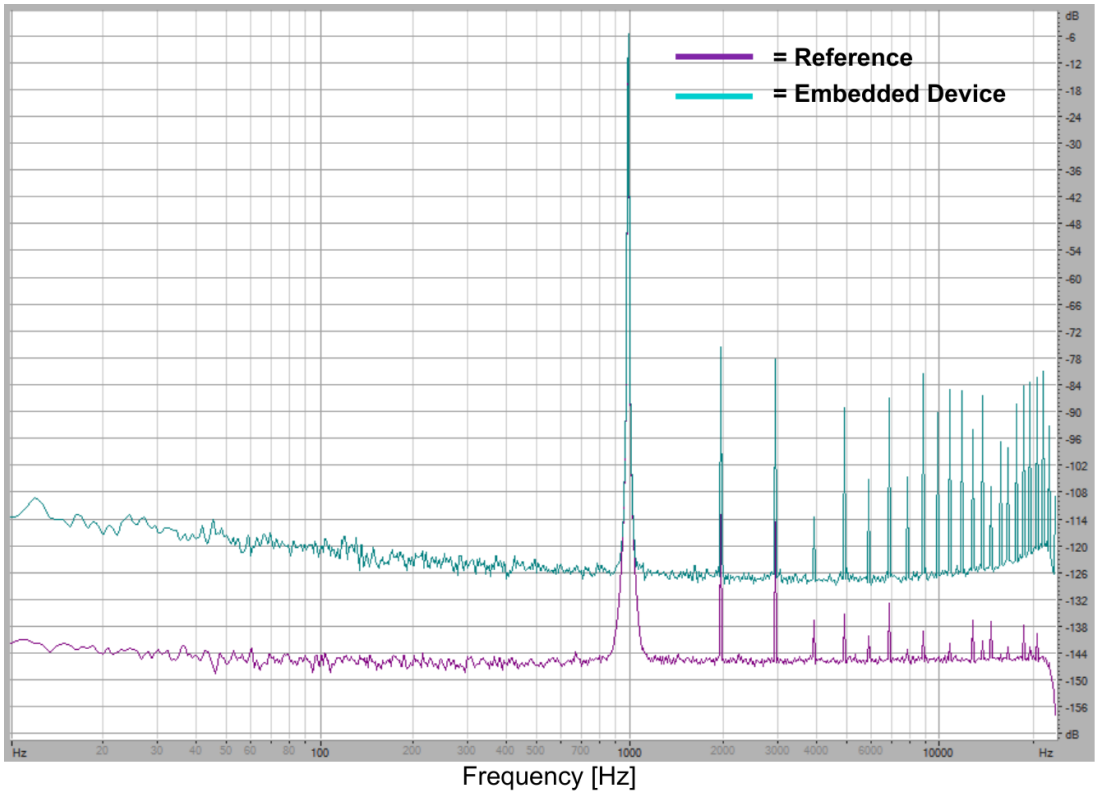

Figure 5.8. Plot of frequency responses of 1 kHz tone compared between embedded device “FARINA” (blue) and reference Audient Evo audio interface(purple)

As can be observed from the comparison of the results, the developed device does not reach the performance levels of currently available commercial devices. This outcome is consistent with the design choices adopted during the development and implementation phases of the system.

These results will be particularly valuable for future developments, as they provide a starting point for further improvements of the overall system. Considering the prototype nature of the device, the obtained results can be regarded as acceptable and demonstrate that the system can perform the basic functionality for which it was designed.

5.2 Quality of the generated Sine Sweep signal

This section describes test performed to validate quality of the Exponential Sine-Sweep signal generated by the developed device by comparing it with signal produced by Aurora plug-in.

The experimental setup remains the same as in the previous test, with only difference being the test signal used. In this case, an Exponential Sine-Sweep is employed.

5.2.1 Test set-up

About connections and operating logic refer to Figure 5.1 and Figure 5.4. The signal generated by the embedded device corresponds to the one described in section 4.2.3 of Chapter 4 implementation, with signal levels consistent with those used in the SINAD experiment presented in the previous section 5.1.

The reference signal was generated using the Aurora plug-in within Adobe Audition. Regarding the signal level, Audition generates a signal with an amplitude of 16384 that correspond at -6 dB_{FS}. Audio interface gain is set to 0 dB. Plug-in window with set parameters is show in Figure 5.9.

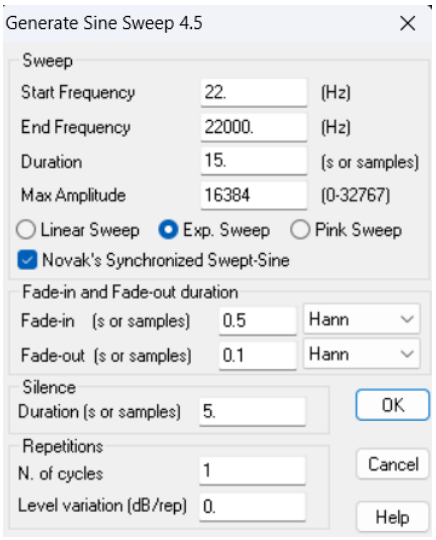

Figure 5.9. Aurora plug-in window on Adobe Audition to generate Sine-Sweep

Both signals are characterized by an initial frequency of 22 Hz, a final frequency of 22000 Hz and a duration of 15 s. With the addition of a fade in fade out as previously shown in section 4.2.3.

5.2.2 Test results

Figure 5.10 shows plot of the frequency response generated on Sine-Sweep signal output from the two different devices.

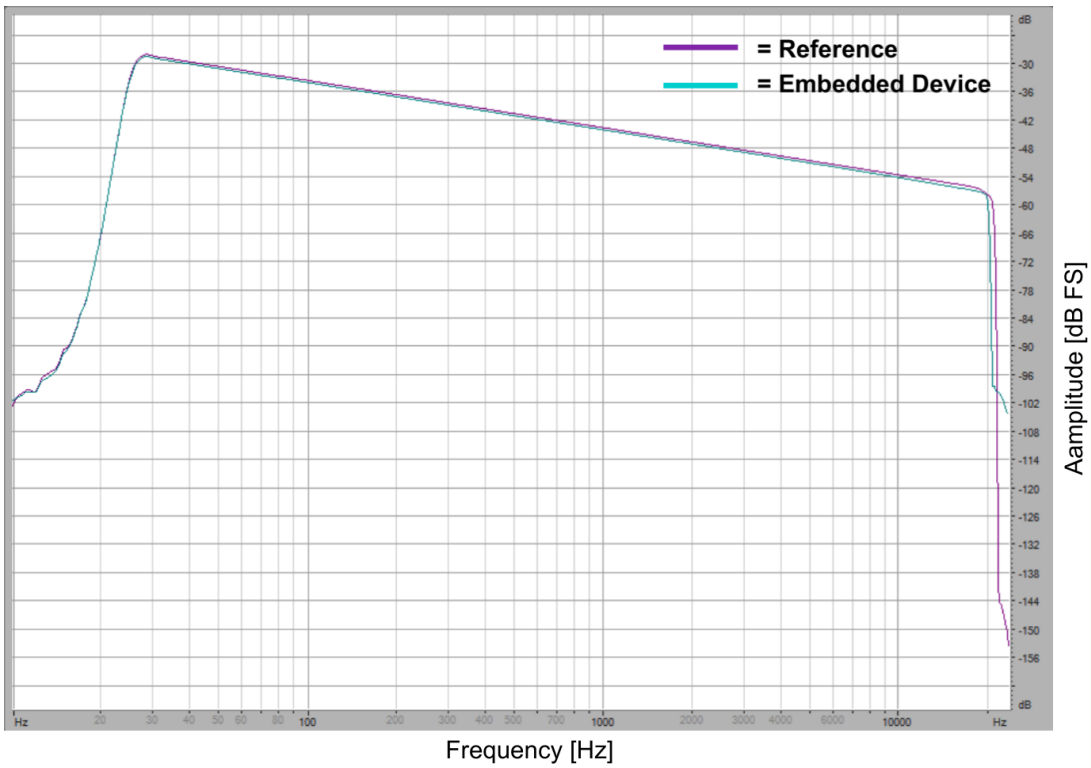

Figure 5.10. Comparative plot of frequency responses of Exponential Sine Sweep signal generates by reference Audient Evo audio interface (purple) and embedded device “FARINA” (blue)

Figure 5.11 shows a zoomed view of the upper part of the frequency response presented in Figure 5.10, where the actual difference between the two signals can be better observed. It can be noted that the fade-in and fade-out profiles (section 3.6.1) are almost identical and closely match those of the signal generated by Aurora plug-in.

The main difference that can be observed is an amplitude offset between the two curves. This aspect has already been discussed in the previous section 5.4.

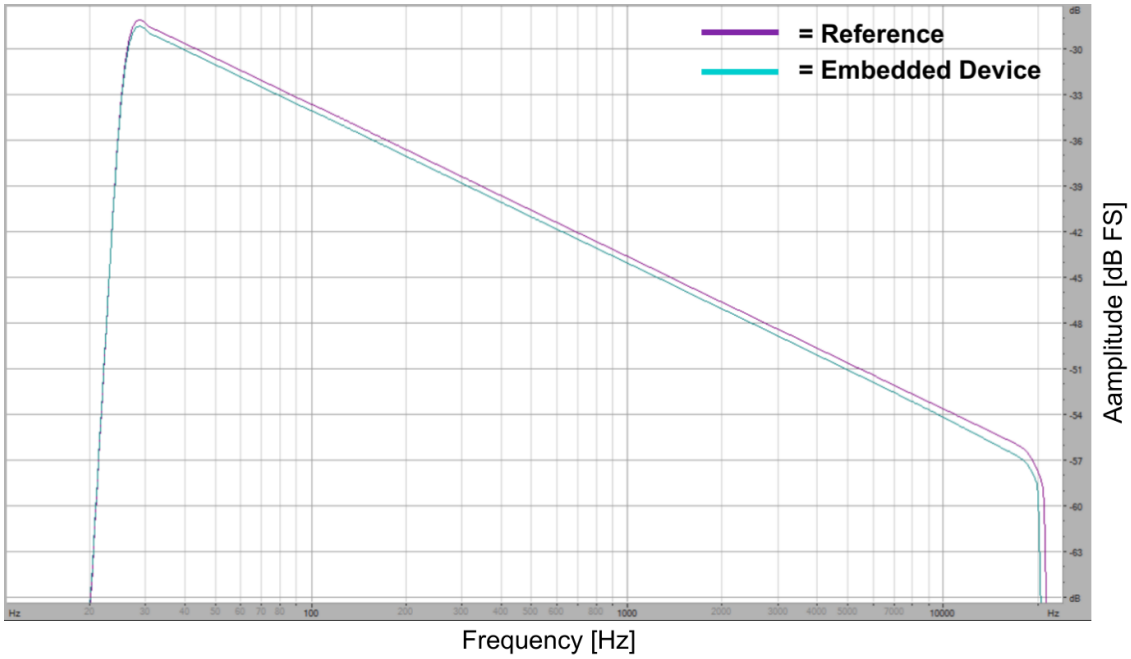

Figure 5.11. Comparative plot Zoom of frequency responses of Exponential Sine-Sweep signal generates by reference system and embedded device

5.3 Power Supply Performance Test

To evaluate the quality of the device power supply implemented in Figure 4.4, the ripple present on each supply rail was measured using an oscilloscope in AC coupling mode. This configuration allows the DC component of the supply voltage to be removed, making it possible to observe only the AC fluctuations associated with the ripple.

Figure 5.12 shows measurement points on each rail of the power supply to assess the noise level present on the different supply lines.

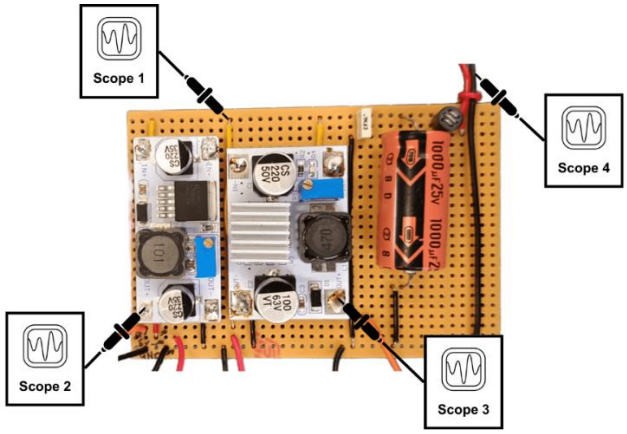

Figure 5.12. Illustrative diagram with the locations where measurements were performed with the oscilloscope

Figure 5.13 shows the waveform measured on the +12 V supply rail. The signal is displayed in AC coupling mode, highlighting the ripple component present on the supply line. A peak-to-peak ripple voltage of 68.8 mV can be observed.

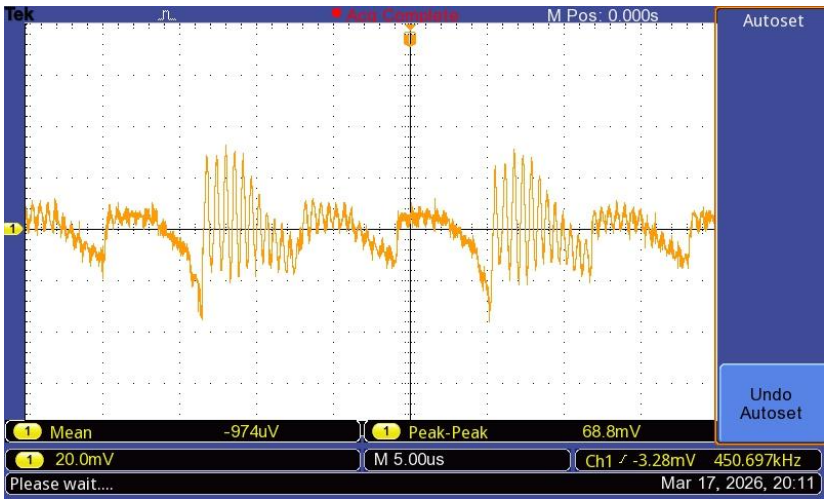

Figure 5.13. Scope 1, AC fluctuations of +12V supply rail

Figure 5.14Figure 5.13 shows the waveform measured on the +5 V supply rail. The signal is displayed in AC coupling mode, highlighting the ripple component present on the supply line. A peak-to-peak ripple voltage of 47.2 mV can be observed.

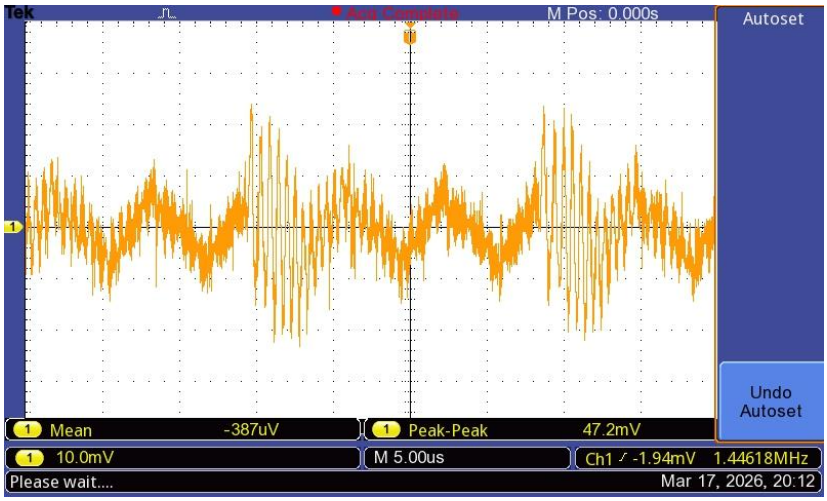

Figure 5.14. Scope 2, AC fluctuations of +5V supply rail

Figure 5.15 shows the waveform measured on the +48 V supply rail. The signal is displayed in AC coupling mode, highlighting the ripple component present on the supply line. A peak-to-peak ripple voltage of 80 mV can be observed.

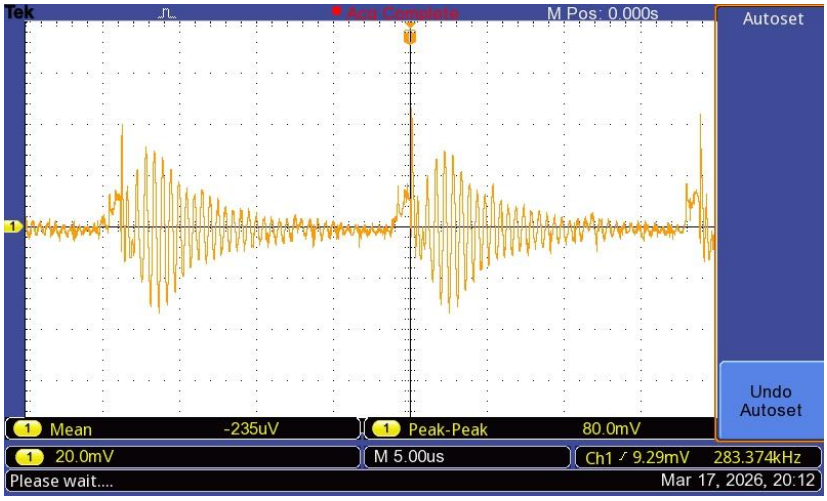

Figure 5.15. Scope 3, AC fluctuations of +48V supply rail

Finally, Figure 5.16 shows the waveform measured at the input of the power supply. The ripple voltage is approximately twice the value measured downstream of the EMI filter Figure 5.13, with a peak-to-peak amplitude of 128 mV.

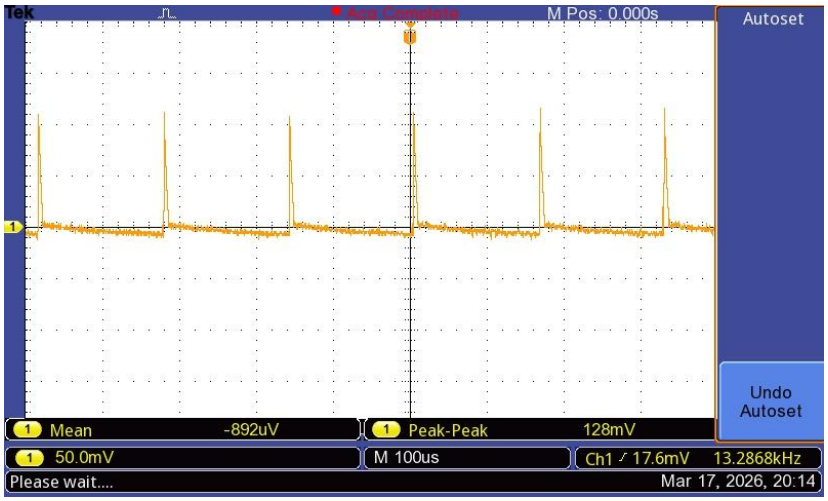

Figure 5.16. Scope 4, AC fluctuations of Power supply input +12V

5.4 Analysing the frequency response of a guitar measured with the embedded device

This paragraph presents the most important test performed to evaluate the capability of the proposed device to measure the frequency response of a musical instrument. Since it was not possible to fully develop the acquisition stage of the system, and only the signal reproduction chain was implemented, this test focuses exclusively on the evaluation of the output stage.

To perform the measurement, it was necessary to use the DAW Adobe Audition, which was used to acquire the generated signal, generate inverse filter, carry out the signal processing and the visualization of the results. Once the complete device, including the acquisition stage, is fully implemented, the use of external software will no longer be required, as all the measurement and processing operations will be performed directly by the embedded system.

The measurement performed with the embedded device will be compared with the measurement obtained using the current method, described in section 2.4 of literature review Chapter.

5.4.1 Test set-up

In Figure 5.17 a block diagram illustrating the set-up used to perform measurements with embedded device is shown. As can be observed, the acquisition chain is the same as the one used in the current measurement method.

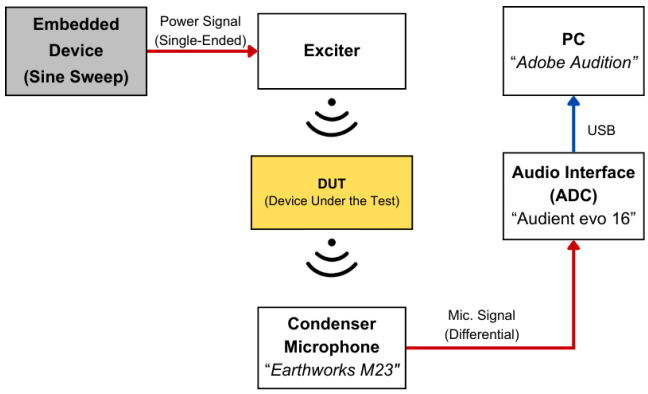

Figure 5.17. Block diagram of the set-up used for performing the measurement of frequency respond of a DUT with the implemented embedded device

For type of signal, signal volume settings, and gain configuration of the audio interface, refer to the section 5.2.1. In Figure 5.18 it is possible to see picture of set-up based on block diagram shows in Figure 5.17.

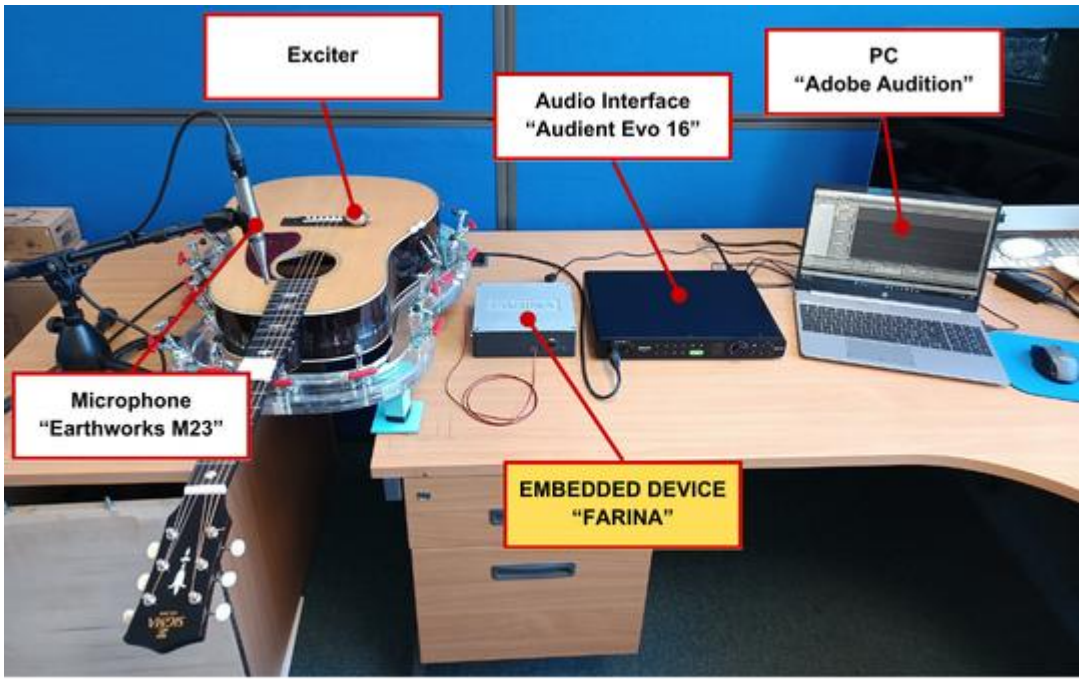

Figure 5.18. Picture of the set-up used for performing the measurement of frequency respond of a DUT with the implemented embedded device

5.4.2 Test results

Figure 5.19 shows plot of frequency response obtained by exciting with a sine sweep signal an acoustic guitar. The purple curve shows the result of the test performed with the current method the blue one the result with the implemented device. Figure 5.20 shows a zoom in the most significative part of a guitar response (80-8000 Hz).

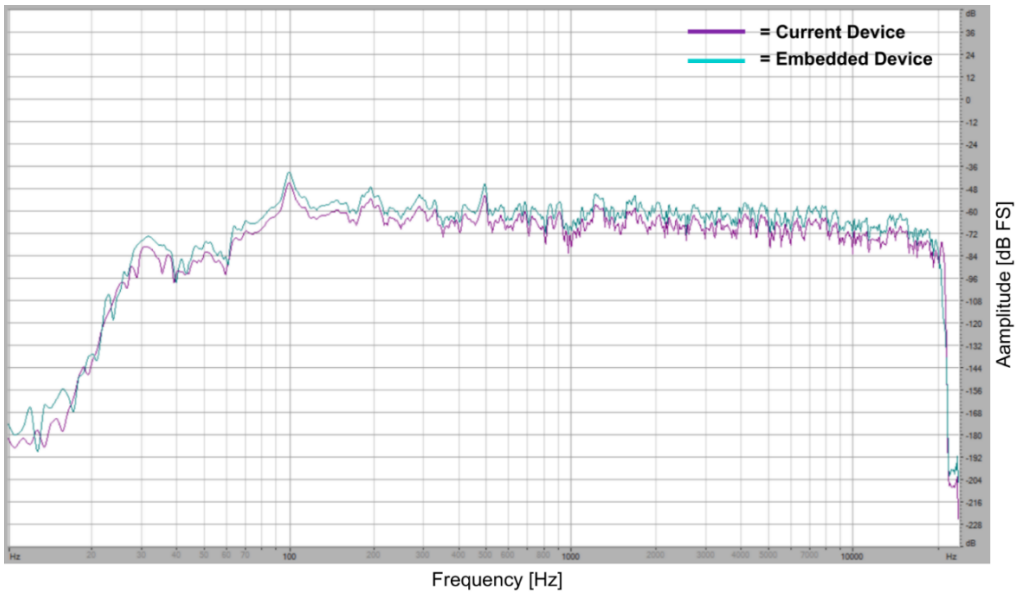

Figure 5.19. Plot of the frequency responses obtained by the two different devices of an acoustic guitar, with exciter positioned on the bridge

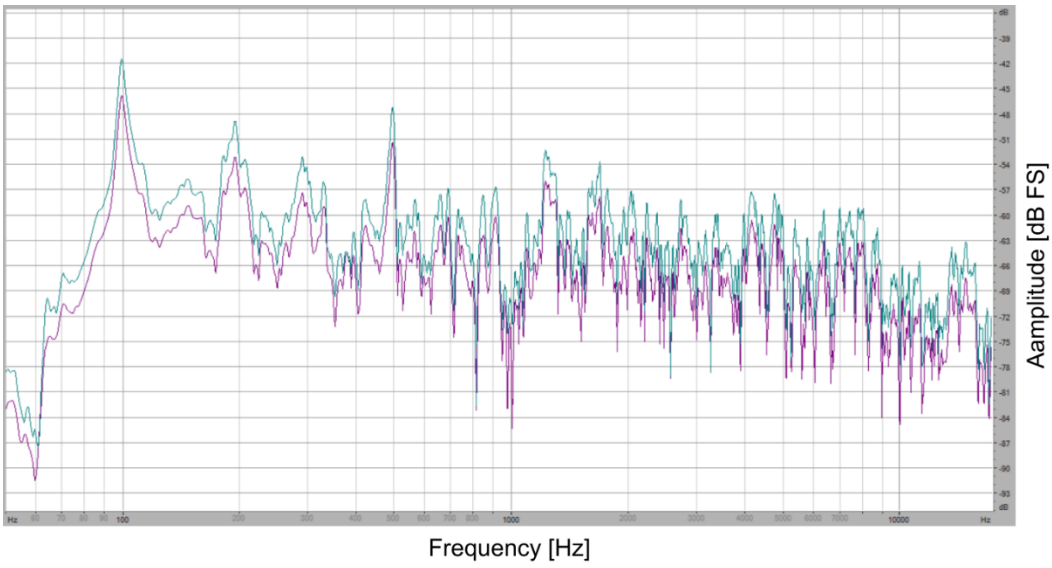

Figure 5.20. Zoom Plot of the frequency responses obtained by the two different devices of an acoustic guitar, with exciter positioned on the bridge

The frequency responses obtained, using the Current system where Sine-Sweep is generated by Aurora plug-in and the embedded device where Sine-Sweep is generated by microcontroller, show a similar spectral behaviour with only a constant magnitude offset.

During the deconvolution process, the Aurora plug-in applies an automatic rescaling factor to normalize the impulse response as shows in Figure 5.21. The applied rescaling was −48.2 dB for the standard Aurora sweep and −42.6 dB for the embedded sweep, resulting in an offset of approximately 5.6 dB between the two responses consistent with the behaviour shown in Figure 5.19. This indicates that the two signals have slightly different energy levels, although their spectral content is equivalent considering the disturbances present in the developed embedded device.

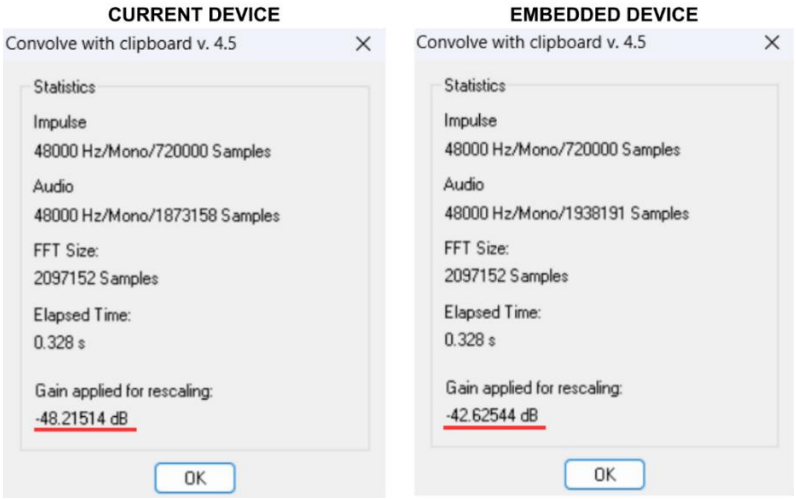

Figure 5.21. Audition windows of deconvolution with inverse filter of the two different tests

To obtain a more complete and accurate comparison, the impulse responses obtained through the deconvolution of the recorded signals from the two tests can also be analysed. These impulse responses were used to compute the frequency responses shown in Figure 5.19.

Figure 5.22 shows plots of the recorded signal and the deconvolved signal used to obtain the impulse response in the time domain for the measurement performed with the current method, which is taken as the reference.

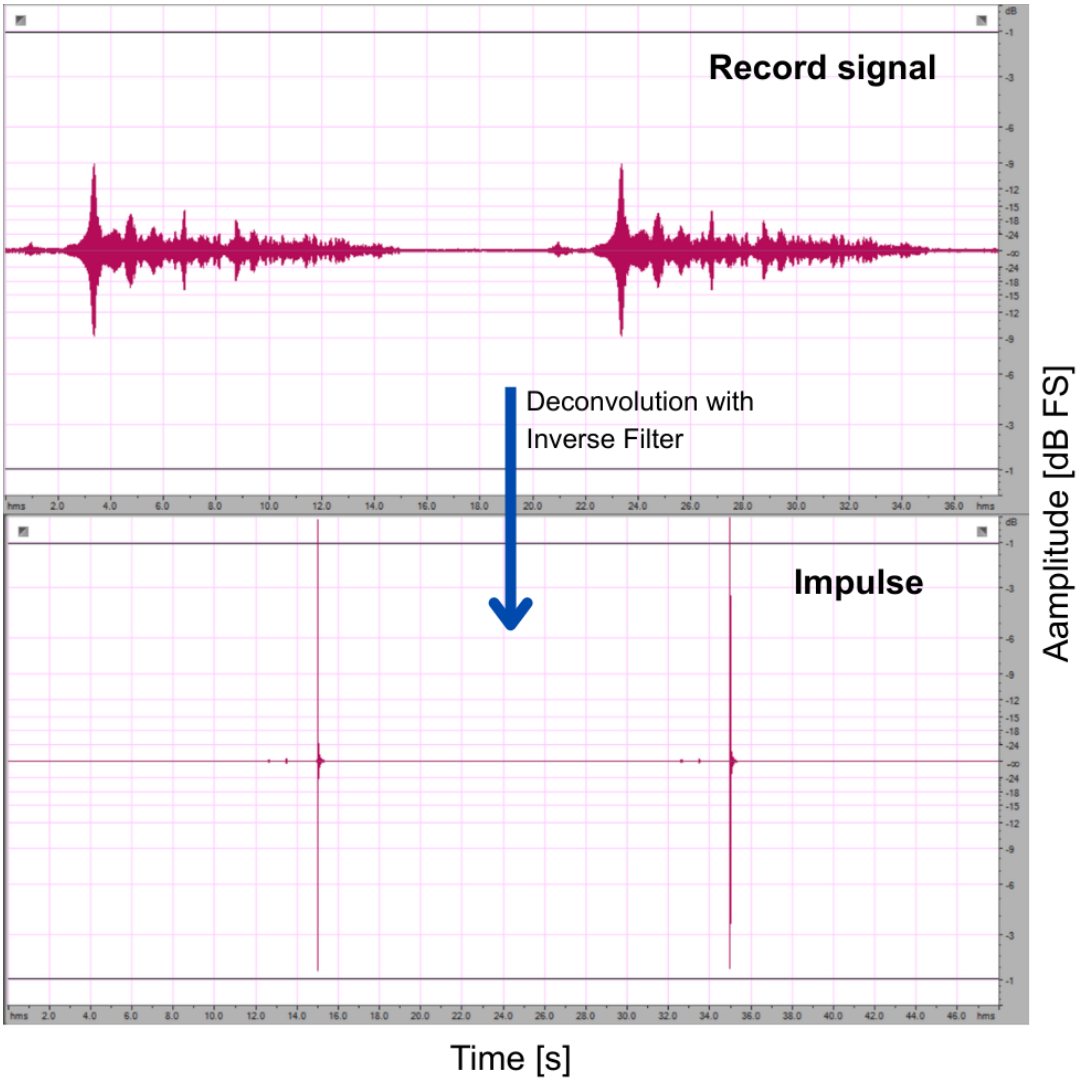

Figure 5.22. Time-domain plots of the recorded signal containing the guitar response and the impulse response obtained through deconvolution of the recorded signal using the current device

Instead, Figure 5.23 shows plots of the recorded signal and the deconvolved signal used to obtain the impulse response in the time domain when using the embedded device.

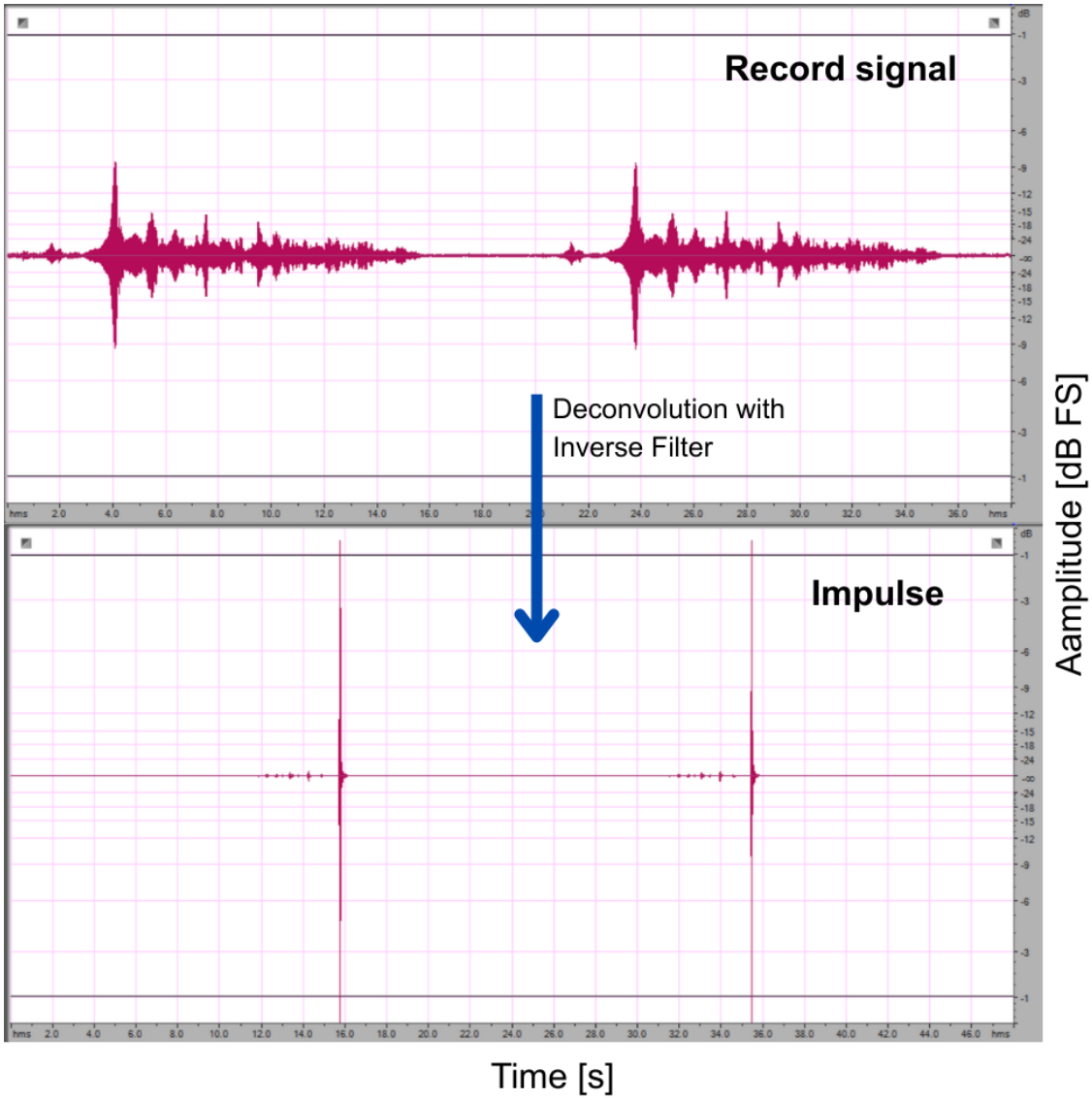

Figure 5.23. Time-domain plots of the recorded signal containing the guitar response and the impulse response obtained through deconvolution of the recorded signal using the embedded device

It can be observed that the embedded device can generate a clean impulse response that is comparable to that obtained with the reference system, although with a different SNR, as discussed in the section 5.1. This result is fundamental to ensure reliable measurements. Another aspect that confirms the considerations previously made regarding the different energy levels of the two signals is that, in Figure 5.23 representing the measurement performed using the embedded system, the impulse amplitude is higher than the one shown in Figure 5.22 obtained with the current method.

By zooming into the signal recorded with embedded device, as shown in Figure 5.24, it is possible to better distinguish the harmonic components of the system, which are separated from the impulse response through the ESS (Exponential Sine-Sweep) method [7].

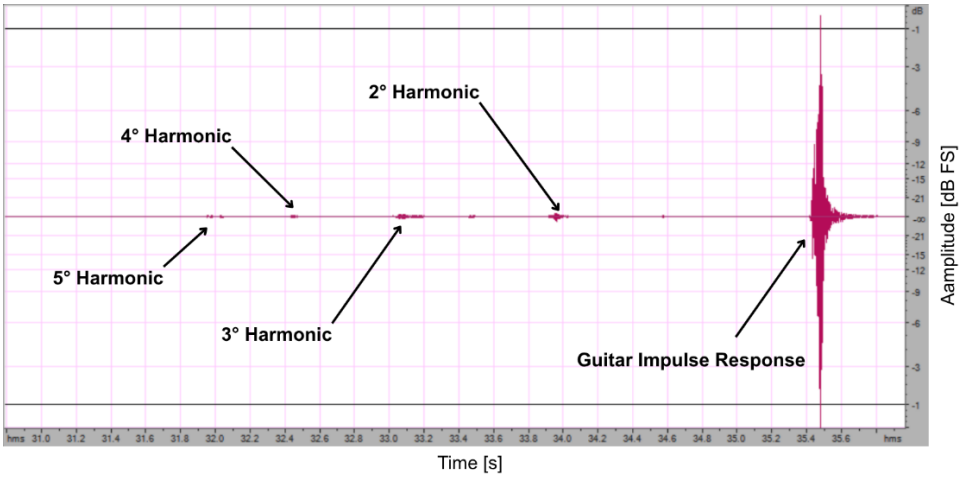

Figure 5.24. Zoom time-domain plot of the guitar impulse response and harmonics recorded with embedded device

Figure 5.25 shows the frequency-domain plot where the frequency response of each individual harmonic is reported and compared with the impulse response of the guitar (20 - 20000 Hz).

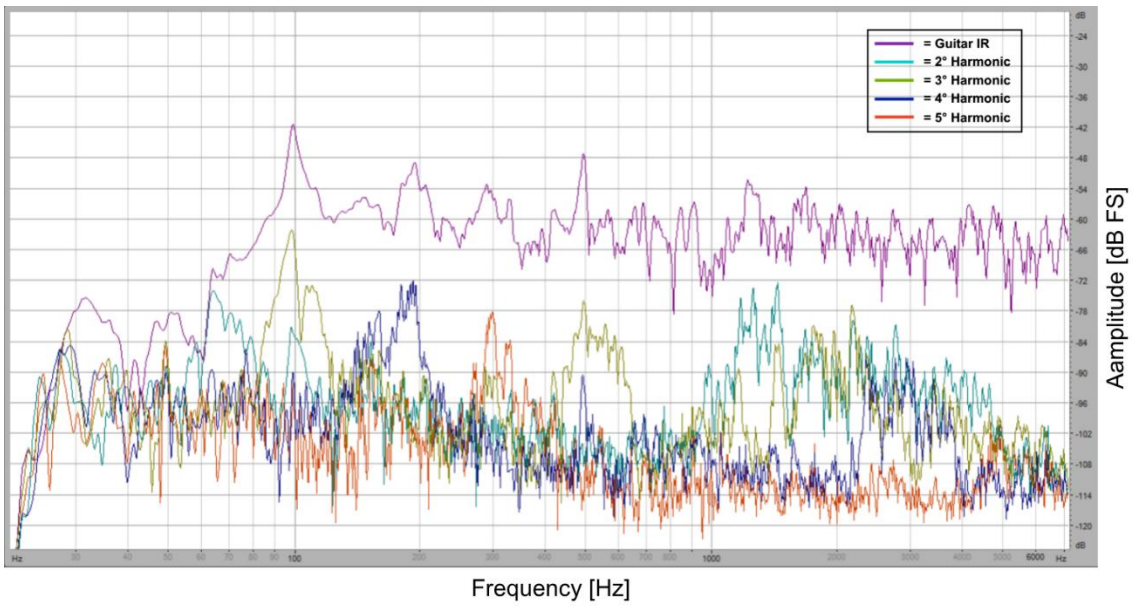

Figure 5.25. Plot in the frequency domain of harmonics and guitar impulse response recorded with embedded device

5.5 Discussion of results

Based on the performed tests and the design choices, an overall overview of the device's performance can be reconstructed, as shown in Table 5.3. The measured values are compared with the system requirements defined in paragraph 3.1.4.

Features	Target Values	Obtained Values
Frequency response	20 - 20 000 Hz	20 - 20 000 Hz
CPU Load	≤ 70 %	≤ 70 %
Storage Size	≥ 8 MB	128 KB
Digital Gain control	≤ 2 dB	1 dB
Differential I/O Audio	1 Vrms	≈ 1 Vrms
Resolution	≥ 16 Bit	16 Bit
Sample rate	≥ 48 kHz	48 kHz
Dynamic Range	≤ -90 dB	-69 dB
THD	≤ 0,02 %	≈ 0,02 %
Jitter	≤ 10 pSrms	≈ 100 pSrms
SNR	≥ 90 dB	63 dB
Power Supply Ripple	≤ 40 mV	80 mV
PSRR	≤ -80 dB	-70 dB
Volume	≤ 8 dm^3	≈ 2 dm^3
Power consumption	≤ 24 W	≈ 23,3 W
Low cost	≤ 300 €	≈ 200 €

Table 5.3. Comparison table between target values and values obtained from tests of the implemented embedded device.

Based on Table 5.3 and the results obtained from the various tests described in the previous sections; it can be concluded that the developed embedded device can generate a test signal with sufficient quality to perform reliable measurements. Although

the performance does not fully match that of target reference systems, the results demonstrate that the device fulfils the fundamental requirements defined in the design phase. These findings provide a solid basis for further optimization and future developments, confirming that the prototype is functional and capable of supporting the intended measurement operations. Figure 5.26 illustrates how the implemented system modifies the measurement set-up.

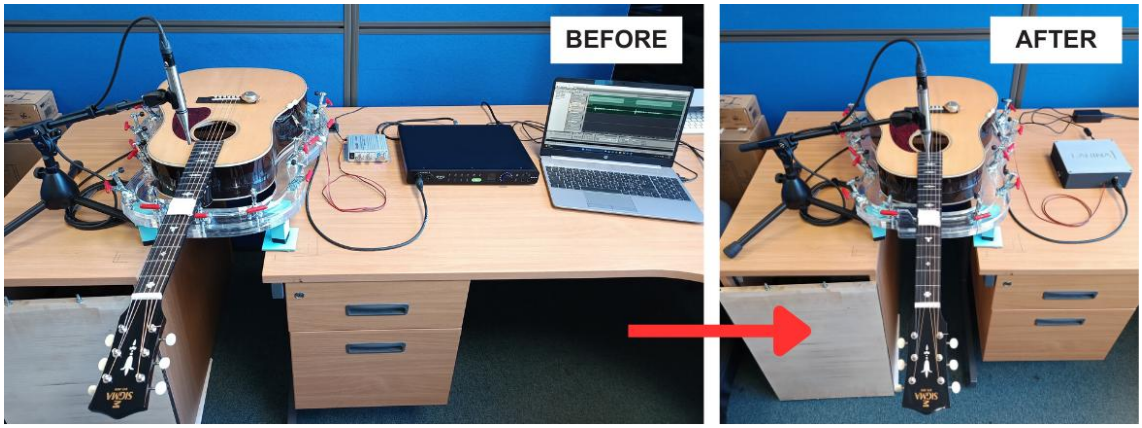

Figure 5.26. Picture with the old measurements set-up compared to the new set-up with the embedded device.

6 CONCLUSION

This thesis has presented the design, implementation, and validation of a stand-alone embedded system dedicated to the measurement of the frequency response of acoustic musical instruments using the Exponential Sine-Sweep (ESS) technique. The primary objective of the work was to overcome the limitations of traditional measurement setups based on personal computers, external audio interfaces, and multiple laboratory instruments, by integrating the essential functionalities of signal generation, control, and system management into a single compact, portable, and low-cost device.

The research activity was carried out following a structured engineering approach, starting from the definition of system requirements based on professional audio standards and progressing through the design of both hardware and firmware architectures. The system was developed around an STM32L476RG microcontroller, supported by dedicated peripherals for audio signal generation, amplification, and communication. Attention was given to the implementation of efficient real-time signal processing techniques, such as the use of DMA-based double buffering for continuous sine sweep generation, and to the integration of a Bluetooth Low Energy interface to enable user interaction through a smartphone application.

The experimental phase led to the realization of a functional prototype, which was validated through a series of tests aimed at assessing its performance in comparison with a professional reference system. The results demonstrated that the device can generate ESS signals with full audio bandwidth (20 Hz – 20 kHz) and a total harmonic distortion of approximately 0.02%, fully meeting the initial design targets. The spectral comparison with reference signals confirmed the accuracy of the implemented algorithms, particularly in terms of envelope shaping and signal consistency. Furthermore, the practical application to the characterization of an acoustic guitar showed that the proposed system can produce a frequency response comparable to that obtained with standard laboratory equipment, validating its effectiveness as an excitation source for vibro-acoustic measurements.

At the same time, the experimental results highlighted some limitations of the current prototype. In particular, the measured signal-to-noise ratio and jitter performance did not fully meet the initial targets, mainly due to the use of interconnected evaluation boards and non-optimized power supply stages. Residual ripple in the supply rails and the

absence of a fully integrated hardware design contributed to these performance constraints, indicating clear areas for improvement.

Despite these limitations, the work demonstrates that it is feasible to perform reliable acoustic measurements using a compact embedded system, significantly reducing the complexity and cost of traditional set-up. The developed prototype provides a solid foundation for future advancements, which will focus on the design of a custom printed circuit board to improve electromagnetic compatibility and signal integrity, the completion of the acquisition and processing chain directly on the microcontroller through the implementation of deconvolution and spectral analysis algorithms, and the optimization of the power management system to further reduce noise and enhance measurement accuracy.

In conclusion, this work highlights the potential of modern embedded technologies to simplify advanced measurement techniques in the field of musical acoustics, paving the way for the development of fully integrated, portable, and accessible instruments for both research and industrial applications.

REFERENCE

[1] T. D., Handbook of acoustic, Rossing (Springer), 2007.

[2] A. Farina, A. Langhoff and L. Tronchin, “Acoustic characterisation of “virtual” musical instruments: Using MLS technique on ancient violins,” *Journal of New Music Research,* vol. 27, no. 4, pp. 359-379, 1998.

[3] A. Farina, “Aurora Plug-ins,” [Online]. Available: https://www.aurora-plugins.com/index.htm.

[4] Fleched and Rossing, Physics of musical instruments, Springer, 1998.

[5] J. Vanderkooy, “Aspects of MLS measuring systems,” *JAES,* vol. vol. 42, no. n. 4, pp. pp. 219-231, 1994 April.

[6] Brown, Allemang and Phillips, “Forty years of use and abuse of impact testing: a practical guide to making good FRF measurements,” *Experimental Techniquest, Rotating Machinery, and Acoustic,* vol. vol. 8, pp. p. 221-241, 2015.

[7] A. Farina, “Simultaneous measurement of impulse response and distortion with a swept-sine technique,” Journal of the Audio Engineering Society. 108th AES Convention., 2000.

[8] A. Farina, “Advancements in impulse response measurements by sine sweeps,” Journal of the Audio Engineering Society. 122nd AES Convention paper, 2007.

[9] L. Ausiello and et al, “Guitar soundboard measurements for repeatable acoustic performance,” Reproduced sound, 2018.

[10] L. Ausiello, “Characterisation of a resonating system by mean of electrical impedance measurements,” 2025.

[11] D. Katz and R. Gentile, Embedded Media Processing, Newnes, 7 Sept. 2005.

[12] AES, “High-resolution audio,” *Journal of the audio engineering society,* vol. Volume 52, no. Number 3, 2004 March.

[13] AES, “High-Resolution Audio,” *Journal of the audio engineering society,* vol. Volume 67, p. Number 5, 2019 May.

[14] I. Dennis, Performance challenges in audio converter design, Cambridge, UK: Prism Sound group, 2011.

[15] “CMSIS DSP Software Library,” CMSIS, [Online]. Available: https://arm-software.github.io/CMSIS_6/latest/DSP/index.html.

[16] STMicroeletronics, *STM32_L476RG MCU Datasheet,* 2024.

[17] STMicroeletronics, *BlueNRG-M0 Datasheet,* 2025.

[18] Analog Device, *Ultra-Low Power Stereo Audio Codec_MAX9867 Datasheet,* 2021.

[19] Maxim Integrated, *Stereo 3.7W Class D Amplifier_MAX98306 Datasheet,* 2011.

[20] R. Bortoni and W. Kirkwood, “The 48 Volt Phantom Menace Returns,” Audio Engineering Society, New York, NY, USA, 2009 October 9–12.

[21] K. Gary, W. Hebert and Frank and T. , “The 48 Volt Phantom Menace,” Audio Engineering Society, Amsterdam, 2001 May 12–15.

[22] *Revisiting Phantom Power Fault Protection, Design Brief 201,* That Corporation.

[23] Texas Instrument, *LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator,* 2023.

[24] Texas Instrument, *LM1577/LM2577 SIMPLE SWITCHER® Step-Up Voltage Regulator,* 2013.

[25] STMicroeletronics, *500 mA low quiescent current and low noise voltage regulator_LD39050,* 2019.

[26] Linear Tecnology, *Low Dropout Linear Regulators 3μA IQ, 20mA, 45V_ LT3008 Series Datasheet,* 2007.

[27] STMmicroeletronics, *Description of STM32L4/L4+ HAL and low-layer drivers_UM1884,* 2021.

[28] STMicroeletronics, *Upgradable Bluetooth® low energy network processor_BlueNRG-MS,* 2019.

[29] STMicroeletronics, *STM NUCLEO_L474RG Evaluation board Datasheet,* 2024.

[30] STMicroelectronics, *X-NUCLEO-IDB05A2 BLE expansion board_UM2700 Datasheet,* 2020.

[31] Maxim Integrated, *MAX9867 Evaluation Kit_MAX9867 Datasheet,* 2016.

[32] STMicroeletronics, *STM32CubeIDE user guide_UM2609,* 2026.

[33] STMicroeletronics, *STM32CubeMX for STM32 configuration and initialization C code generation_UM1718,* 2026.

[34] Forum, “Audio science review_Understanding Digital Audio Measurements,” [Online]. Available: https://www.audiosciencereview.com/forum/index.php?threads/article-understanding-digital-audio-measurements.10523/.

[35] L. Ausiello and V. Hockey, “Quantitative measurements to enhance performance of acoustic musical instruments and improve manufacturing,” Acoustic Bulletin 47, 2, 2021.

[36] L. Ausiello, M. Ducceschi, S. Duran and B. Morrison, “Affordable wide-band measurement ecosystem for musical acoustics based on electro-dynamic transducers,” Acta Acustica, 8(53), 2024.