Acoustic data transmission for embedded software platforms: an empirical study

Master of Science Thesis
University of Turku
Department of Future Technologies
Embedded Systems Laboratory
2019
Henrique Gonçalves Nogueira

Supervisors:
Msc. Jorge Peña Queralta
Assoc. Prof. Tomi Westerlund
Tommi Otsavaara

UNIVERSITY OF TURKU
Department of Future Technologies

HENRIQUE GONÇALVES NOGUEIRA: Acoustic data transmission for embedded software platforms: an empirical study

Master of Science Thesis, 70 p.
Embedded Systems Laboratory
August 2019

As microcontrollers become increasingly powerful at a lower cost, they continue to expand to new fields of applications, in particular those under the process of a digital transformation. These systems are often packed with a broad array of complementary subsystems, that can be selectively enabled to further facilitate their integration in larger designs. Due to this immense malleability, they often enable creative problem-solving approaches that not only serve to improve the product's overall functionality, but may also help to drive down costs even further.
This thesis is based on the design and implementation of an embedded software modem system, consisting of a non hardware-native communication interface. The interface is based on the transmission of audio signals and can thus be often implemented with little to no additional hardware costs by utilizing the preexisting functionality of the platform's features. Under the constraints of the limited computational capabilities of embedded processors, the system works as an efficient communication layer that can be easily integrated into broader software systems concurrently running on these devices.
In contrast with signal propagation of wired interfaces, the wireless transmission of acoustic signals brings forth a new set of challenges, which are tackled using sensible strategies based on well-established telecommunication's theory. Nevertheless, the design approach is largely platform independent, with configurable performance parameters that can be adapted to the available computational resources and system specifications. The proposed architecture is based on the OFDM signalling scheme with QAM-16 carrier modulation and the implementation results show that the system can reliably support up to 32kb/s message transmission speeds for an average interface setup.

Keywords: DoS, OFDM, IoT, DSP

Contents

List of Figures

List of Tables

List Of Acronyms

AES	Advanced Encryption Standard
ADC	Analog to Digital Converter
AES	Advanced encryption standard
ASK	Amplitude Shift Keying
AWGN	Additive White Gaussian Noise
ADT	Acoustic-data-transmission
B2B	Business-to-business
CP	Cyclic prefix
CPU	Central Processing Unit
CRC	Cyclic Redundant Check
DAC	Digital to Analog Converter
DoS	Data-over-sound
DFT	Direct Fourier Transform
DOS	Data-over-sound
DSP	Digital Signal Processing
DMA	Direct Memory Access
FFT	Fast Fourier Transform
FIR	Finite impulse response
GPIO	General purpose input/output pins
IIoT	Industrial Internet-of-things

IoT	Internet-of-things
ISI	Inter Symbol Interference
LTI	Linear-time-invariant
MCU	Microcontroller Unit
PAPR	Peak-to-average-power-ratio
PCM	Pulse Code Modulation
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
M-QAM	M-ary Quadrature Amplitude Modulation
RF	Radio-frequency
RIFF	Resource Interchange File Format
OFDM	Orthogonal Frequency Division Multiplexing
SAI	Serial Audio Interface
SDK	Software development kit
SNR	Signal to noise ratio
SOC	System on a chip
SOF	Start of Frame
S&C	Schmidl and Cox
USB	Universal serial bus
UWCN	Underwater wireless communication networks
ZCS	Zadoff-Chu Sequence

1 Introduction

The Internet of Things (IoT) is built around connected devices, which monitor data or bring intelligence to multiple domains [1]. In most applications, data transmission systems for embedded devices are implemented using electric signals, transmitted under controlled physical media such as copper traces and wires, or wirelessly with the propagation of radio waves [2]. These methods support very fast data transfers due to their large signal bandwidth and are preferred over alternatives that rely on mechanical processes, such as the propagation of sound. However, while the number of connected devices is reaching the order of tens of billions [3], multiple products and applications involving embedded devices lack the connectivity required to join the IoT [4]. The focus of this project is to design and document a software communication system intended to run entirely on an embedded microprocessor. It uses a novel wireless communication solution based on acoustic data transmission that can be applied to extend IoT connectivity of musical instruments, while exploiting the existing hardware.

Data-over-sound (DoS) is the concept of transmission of digital information through audio, which, unlike common speech-based audio communication, relies on symbolic data, typically encoded in binary format. When compared with other channel media that make use of electric or electromagnetic signals, acoustic propagation uses a much narrower bandwidth at a lower frequency, however, also due to its slower data-rate nature, no specialized hardware is necessarily required in order to implement capable modem systems that can efficiently utilize this spectral band. Acoustic signals can be read as a continuous string of audio samples, and if the processing system has enough memory and is fast enough, it is possible to process these samples at a rate that facilitates real-time data transmission. So, unlike radio-based communication, DoS systems can theoretically be efficiently implemented on any existing device that already integrates the basic hardware support to support such an interface.

Often times, channel characteristics other than bandwidth are also important to consider. These include aspects such as whether the communication is being performed out of line-of-sight, system hardware limitations or whether the environment restricts RF propagation. These factors may define the need for the application of non-conventional communication channels. For example, if we consider the field of underwater wireless communication, due to an unusual ineffectiveness of underwater electromagnetic propagation, acoustic propagation is the most popular, efficient and proven method for underwater wireless communications [5].

It is also important to consider that, due to the very high bandwidth and processing speed requirements of typical communication systems, the modems typically exist in the form of hardware-based implementations. These devices are composed of dedicated hardware modules that can perform the required real-time signal processing operations at a very fast pace, however, their functionality is limited in the sense that they are explicitly designed for that single purpose. In contrast, software-based platforms such as microcontrollers serve as highly flexible, and often cheap, laboratories onto which we can deploy a plethora of systems. In summary, this thesis provides a set of design strategies for the software implementation of a sound-based data transmission system, requiring little to no additional hardware support for most commercially available embedded platforms. For this purpose, acoustic-based channels provide an attractive solution, not only due to the uncomplicated integration efforts but also because it facilitates a signal path for wireless data transmission.

1.1 Motivation and system objectives

This thesis project has been developed at *Darkglass Electronics Oy*, a Helsinki based company specializing in premium bass gear, in particular, bass distortion pedals. Essentially, these systems are audio signal processors, and thus serve to modify certain characteristics of the bass audio signal, both in analog and digital domains. These may include a microcontroller unit (MCU), not only responsible for controlling some of the system's various analog modules but also as an engine to run digital signal processing (DSP) algorithms.

These systems are highly configurable, facilitated by the fact that they integrate an MCU that executes firmware responsible for controlling the remaining device modules. In essence, the software can be rewritten/reconfigured to completely modify the user interface behaviour as well as the operation of some of the analog subsystems and the DSP algorithms. As such, the company has developed a computer application, the Darkglass Suite, that allows for each user to configure the system according to her/his needs within the bounds of the relevant implemented product features. By default, the application interfaces the products over USB and is thus the configuration accessibility is somewhat constrained by this factor.

The project idea consists of extending the user connectivity, with an alternative channel to the USB interface already implemented, allowing users to configure the devices on-the-go. Since these systems are designed to process audio signals they already possess the necessary analog conversion hardware, and thus the acoustic signalling path is already well established. This type of interface has the great benefit that it can be applied across the whole Darkglass product line, including already deployed production devices which can be easily updated via field firmware updates.

The system is originally implemented on Microchip's ATSAM4S4A MCU, currently deployed on all the relevant devices, running at 120 MHz frequency with an ARM Cortex-M4 architecture. Most of the interface messages are relatively small in size, typically under 1kB, with a few exceptions: impulse response filter coefficients start at 2kB in size but are expected to increase for products further down the line, and firmware update images, which, for now, can theoretically go up to 120kB in size. Since, ideally, the longest transmission duration should be under around one minute, for the sake of the quality of user experience, the targeted system transmission rate can be defined to be around 2kB/s.

The physical interface can be routed in two different ways. The first and simplest connection is achieved by hooking up the device to the user's phone through a regular 2.5mm audio jack cable, connected to the *aux* port of the devices. Alternatively, as a second option, the user may use the phone's stereo system to provide the acoustic signal that can be picked up by the magnetic pickup sensor present in the bass guitar. Note however that this wireless transmission route is not truly acoustic in nature as the audio signal is converted into an electric signal due to vibrations in the stereo actuator that are picked up by the bass pickup-sensor as changes in the magnetic field, and not due to the mechanical vibrations of the air. This works similarly to how the bass string vibrations are converted into electric signals and as such presents slightly different characteristics when compared to a typical speaker-microphone pair setup. Nonetheless, the methods that have been developed can be extended to pure acoustic transmission over the air or other mediums. For longer distance transmissions, multipath propagation needs to be taken into account.

The wireless transmission route option is naturally much more sensitive to specific characteristics of the phone's stereo system and the bass guitar's pickup sensor but importantly provides a practical way to configure the device on-the-go. While these communication channels are inherently different, the system design is tailored towards the characteristics of the speaker-pickup pair communication channel due to the fact that it is naturally more restrictive across the board and presents much greater challenges, meaning that it has a narrower bandwidth, generates channel distortion effects, is more susceptible to inter-symbol interference (ISI) and shows a worse SNR performance, etc.

Table 1.1: Synthesis of the system's specifications

	Specification
Direction	Simplex (user out, device in)
Data integrity	Lossless data transmission
Security	Unencrypted / encrypted with preset key
Data rate	2 kB/s
Route	Wired or wireless(stereo-pickup pair)
Message length	> 64 kB
Sample rate	$\simeq$48 kHz

The communication channel is also of type simplex, meaning that data can only travel in one direction: from the user to the device. This poses an additional challenge at a protocol level since the transmitter does not know when or if the receiver has received the complete message. Given that transmission should be loss-less at all times, independently of the form of the message, strategies need to be in place guaranteeing that the receiver may always eventually receive a complete message. The system specifications are summarized in Table 1.1.

1.2 Related works

The proposed system implementation presents several challenges. Nevertheless, arguably, the fundamental difficulty is related to the implementation of real-time digital signal processing of wireless communication algorithms on the limited computational capabilities of embedded software platforms. In recent years, solutions relying on data-over-sound transmissions have emerged to extend the possibilities of connectivity in the IoT. However, most recent advances have happened within the industrial sector, and the solutions are not open-sourced. In this section, we provide an overview of the two most widely used solutions for data-over-sound from *Mutable Instruments SARL* [6] and *Chrip.io* [7].

This project presents many development challenges, however, arguably the fundamental problem that is tackled here has to do with the implementation of real-time digital signal processing of wireless communication algorithms on the limited computational capabilities of embedded software platforms. While there are other products out there that offer a similar feature under equivalent hardware constraints, typically the implementation details are kept undisclosed to the public, and it is hard to speculate any performance characteristics without having the opportunity to experiment extensively with these devices.

The company *Mutable Instruments SARL* provides a promising case study for this field of research with the development of digital synthesizer modules supporting audio-based firmware update features. Their designs are publicly available as open-source [6]. For example, the *Elements* modal synthesizer runs on an STM32F405RG Microcontroller chip from *STMicroelectronics*, which coincidentally has very similar characteristics to the chip used for project development, having, in particular, the same Cortex-M4 ARM processor architecture. By consulting the firmware code available at their Github repository, we can infer that the firmware update system is using a single carrier quadrature phase-shift-keying (QPSK) as a signal modulation scheme. Notably, the software implementation is surprisingly short and is kept rather simple, however, the bandwidth usage cannot be very efficient, as it is using a signal correlation estimator based on the sign information alone. Furthermore, even though the firmware update procedure is based on signals produced in the acoustic band, it only supports a wired interface to the audio source, meaning that the implementation does not have to account for effects due to wireless propagation. In our work, we utilize OFDM modulation to provide higher bandwidth and configurable channel utilization.

A white paper published by *Disruptive Analysis Ltd.* provides an in-depth analysis of DoS as a technology and its current and potential application in different fields [7]. In particular, the concept is gaining traction over fields for the Industrial Internet-of-Things (IIoT). Multiple companies are already working on DoS as a cross-platform third party technology for the IIoT. One of the solutions with higher market penetration is *chirp.io*, a B2B service-based company developing a suite of software development kits (SDK) that can work under different platforms, including a plethora of operating systems or even support for C development of embedded bare-metal firmware. Their focus is on acoustic transmission through sound propagation using an M-ary frequency-shift-keying (FSK) modulation scheme due to its robustness in multipath-propagation environments, while also being adaptable to different data-rate profiles. There is a clear parallelism between this solution and our proposed system. In particular, the fact that both harness the capabilities of unspecified pre-existing hardware. Nevertheless, dealing with multi-platform development, decisively serves to amplify the challenges faced by the company, as the processes must not only be adapted to the device's computational abilities but also the audio-related hardware interfaces.

Table 1.2 provides an overview of the aforementioned systems and the system proposed in this thesis, labelled as "Ours". In this context, *Chirp.io* naturally offers the most mature solution, providing ample support for various types of applications, different platforms, high reliability and a broader set of features. On the other hand, the system developed for this thesis provides the basis for much higher data transmission rates than the remaining solutions. It is noteworthy to mention that, regarding the Mutable Instruments solution, all parameters are derived from the analysis of the open-source codebase alone, and should thus be taken with a grain of salt.

When discussing acoustic data transmission applications, arguably the most advanced application of such technology is in underwater wireless communication networks (UWCNs). Underwater wireless communication presents an extraordinary set of challenges, in particular, is the fact that electromagnetic propagation behaves very differently in this environment. For example, radio waves can only propagate over long distances when

Table 1.2: DoS systems comparison overview.

	Chirp.io	**Mutable**	**Ours**
Platform	OS/baremetal	Baremetal	Baremetal
Wireless	yes	no	yes
Encryption	optional	no	no (optional)
Data rate	Up to 1 kbps	N/A	Up to 32kbps
Distance	1 cm-100 m	N/A	1cm (extendable)
Spectral band	acoustic	N/A	acoustic
Direction	Simplex/Duplex	Simplex	Simplex
Signalling	M-ary FSK	QPSK	QAM16(OFDM)

transmitted at high power levels and at extremely low bandwidths (30-300Hz), rendering relatively low data rates [8]. While there exist underwater acoustic modem hardware solutions, which can be used to establish a communication link between two nodes, the embedded software platform alternatives may prove to be a more flexible and cost-effective alternative. Qiao et. al (2013) propose an acoustic modem, running on DSP software platforms, which, similarly to this project, utilizes orthogonal-frequency-division-multiplexing (OFDM) as the fundamental signalling scheme [9]. In essence, we see that the fundamental differences in acoustic-based communication schemes derive from characteristics of the channel, transducers and sensors and the subsequent complexity of the algorithms needed to tackle those differences.

1.3 Structure

The purpose of this document is to detail the actual implementation of a communication system, in a way that it is both instructive and tries to respect the actual progression of the project development. It aims at explaining the basic theory behind the design of each module, and it details the system design approach at an algorithmic and software implementation level, all while being conscious of embedded-related constraints. The core structure of this document is split into three main parts:

- Chapter 2 covers some of the abstract basic theory of digital communication systems in general. Namely, it narrows down on the OFDM signalling scheme as this is the central mechanism for digital data transmission.
- In chapter 3, we take a deeper look at the system design from an algorithmic point of view. While this chapter tries to be as agnostic as possible to specific platform constraints, the design approach is always conducted in a way that is conscious of typical embedded-processor limitations.
- Finally, chapter 4 mentions some platform-specific details and techniques as they relate to the software implementation of this system. Here, we also take a look at some of the most relevant experimental results.

2 Data communication systems

The fundamental principle of digital data communication is the exchange of information, once described in the form of discrete symbols. In every information exchange, a signal must be involved so that either of the relevant actors may sense changes in their perceived environment, and thus extract information from these events,i.e. the propagation of signals, the information is transmitted over a given communication channel. Equally as important, the transmitters and receivers must share a common language so that the exchanged information is meaningful, otherwise, the signal is just plain noise.

Signals can be extremely simple, for the most basic exchanges of information, or they can be complex to a point where they become imperceptible to normal humans, as the information becomes embedded in more subtle ways and in greater density. For example, while I am writing this paragraph, I am waiting for the bell to ring so that I can pick up a mail package being delivered. It can be thought of as a binary signal, where it is either active, while the bell rings, or inactive, while it is silent. As soon as it rings, I can only extract the following information: someone is at the door. This protocol is very simple, and given the bandwidth and message requirements, it works just fine. However, if I would also like to know more information about who exactly is at the door, I would have to resort to some other protocol, or perhaps a different system altogether.

It is important to emphasize this aspect to convey a sense of what is involved in the development of a data transmission system from "scratch". The options are practically endless and, with smart design choices, the system can be implemented with varying degrees of optimality. The challenges of this project consist of applying existing communication theory in ways that are suitable to the platform at hand, and according to the specs described in 1.1. To meet these requirements, we are required to carefully apply the telecommunication's principles that serve as a foundation for modern communication networks. The theoretical background on digital telecommunication systems is vast and complex, and so, the purpose of this chapter is to lay out the basic theory which pertains directly to the project's implementation, serving as the support for the most fundamental system design decisions.

The information compiled in this chapter is mostly inspired by the works of [10] and it is focused on topics which are directly related to the physical aspects of communication systems, often called the physical layer. The physical layer focuses on the generation and interpretation of analog signals as carriers of information and the principal challenge of this project is in designing an efficient scheme that can be translated into software. Section 2.1 mentions some basic concepts regarding data signalling techniques, then, section 2.2 introduces the main fundamental concepts for real-world applications. Finally, section 2.3 takes a deeper look into the signaling scheme that will serve as the basis for system design later on.

2.1 Data signalling basics

The functioning principle of data-communication systems boils down to the generation and processing of information-carrying signals. For the context of this thesis, signals are always bi-dimensional and can be conceptualized as voltages changing over time, as a result of the excitation of the pickup sensor. This section aims at explaining how symbolic information can be embedded into continuous signals.

2.1.1 Time and frequency domains

Typically, a one-dimensional signal is represented as a value varying over time. However, sometimes this approach is limiting in the sense that certain signal characteristics are not visually apparent by looking at its shape alone. The truth is that the shape, or form, of a signal, typically is not directly related with the information it contains. For example, when considering signals that we process naturally as humans, such as audio and images, the content can be more easily described by its frequency profile rather than the shape of the curve as the values change over time. When something looks red, it is because of the emission of a light signal with a wavelength close to 700nm, and when a sound feels dissonant, it is because the frequencies of the composing sounds don not fit our natural harmonic rules.

It quickly becomes apparent, once signals increase in complexity, it is important to look at the same signal from the perspective of two vastly different domains. The frequency domain lays out information about the signal across its frequency spectrum. Essentially, it conveys how the signal can be decomposed into a sum of sinusoidal waves for every frequency value, whereby for each frequency there is an associated component value represented in complex number notation, $C_f = Ae^{\Phi}$, where A defines the amplitude and Φ defines the phase of the component at a given frequency.

The Fourier transform operation $\mathcal{F}$ converts a time-based signal into its frequency domain equivalent and conversely, the inverse Fourier transform operation $\mathcal{F}^{-1}$ does the opposite. In digital systems, however, information is presented in a discretized fashion, i.e. time-based signals are usually stored as sequences of discrete samples, and it turns out that this characteristic imposes some interesting behavior when applying the Discrete Fourier Transform operation (DFT). The DFT, as the name implies, performs the equivalent of a Fourier transform over sets of digitally sampled signals, and while its intrinsic properties and constraints go beyond the scope of this document, there are a few key points to notice before moving forward with the digital communication theory basics.

As the sampled signal represents can only ever be represented by a finite number of values over a given time-span, so to can the frequency spectrum only represent a finite number of frequency components. For a signal composed of a total of N samples, sampled at a fixed rate f_s, the frequency resolution of the resulting DFT spectrum is calculated as $\Delta_f = \frac{f_s}{N}$, meaning that the resulting spectrum does not directly represent frequency components that do not fall exactly at those specific frequency coordinates. Moving forward, this factor will be particularly important when trying to efficiently and explicitly calculate the key frequency components of a given signal.

The conversion between time and frequency domains are central operations in the realization of digital communication systems, and, as it pertains to the implementation of this projects, it is also important to note that all DFT operations are actually performed with the Fast Fourier Transform (FFT) algorithm. Multiple variations of this specific algorithm exist, towards varying degrees of speed and precision, but notably, one of the main aspects is that it operates most efficiently in data sets where the number of samples is a power of two, for example, 32, 64, 128, 256 and so forth. This is because the basic construction of the algorithm revolves around a divide-and-conquer approach, where the data-sets are operated on while recursively broken down in half. In essence, the takeaway is that this operation offers an improved computational complexity $\mathcal{O}(N\log(N))$, when compared to more straightforward DFT implementations of the order $\mathcal{O}(N^2)$.

2.1.2 Pulse Code Modulation

For binary data, the simplest signal construction is the direct representation of successive bits over time, where each bit value is mapped to a specific voltage level, for example, +1V for the bit value 1, and -1V for the bit value 0. This stream of bits would be transmitted at a pre-defined rate R_S, corresponding to the rate at which the receptor will sample the transmitted signal: $R_d = R_s$.

To try and improve the transmission speed, assuming that the maximum sampling rate is fixed at a specific rate, multiple bits can be packed into the same symbol and, again, mapped to specific ranges of voltage levels, and finally, transmitted at a fixed rate. In this case, the data transmission speed depends on the number of bits per symbol and the system's sampling rate. If L defines the number of discrete voltage levels, then the data rate can derived from: $R_d = R_s \lfloor \log_2 L \rfloor$.

This modulation technique is called Pulse Code Modulation (PCM), where symbolic codes are transmitted as pulses at fixed sampling rates. Its appeal comes from the fact that it is very simple to implement when compared to other alternatives: just sample the voltages at periodic moments in time and compare the readings with a predefined value table. However, there are some key limitations in the implementation of a PCM based system when considering the speaker-pickup communication channel:

- **Channel attenuation:** There would need to be some strategy, such as compensation factor, to calibrate the values to the voltage level table. Furthermore, making matters worse, given that the speaker-pickup pair are not in fixed positions during the transmission, the attenuation values fluctuate over time.

- **Effective bandwidth utilization:** While PCM frequency domain profiles depend on the data being transmitted, in general, the signal is largely spread across the spectrum. For example, with a transmission consisting of consecutive alternate bit values with binary coding, the resulting wave would be a square wave with frequency $F_{sq} = R_d/2$. For this particular signal, the spectrum decays linearly with the frequency, and the fundamental frequency component corresponds to only about 81% of the entire signal power. As explained later on, the bandwidth of the speaker-pickup pair channel is very limited, and so, ideally, the power density of the transmission signal should be strong.

- **Phase response linearity:** A linear-phase system performs in such a way that the output signal maintains the same shape as the output signal, i.e. if a square signal is transmitted, the received signals continues to be a square signal as well. However, in general, the speaker-pickup pair does not show a linear-phase behavior. This is further exacerbated by the fact that the channel is limited in the frequency band, and so, the signal is further deformed. In such cases, modulation techniques that depend on the signal shape, such as PCM, are not optimal.

2.1.3 Passband signaling techniques

In the previous modulation technique, PCM, the signal is in its most basic form: sequences of symbols are directly translated into voltage levels varying over time. This is an example of baseband transmission, where the original signal is transmitted directly without any modulation effects. The bandwidth of the transmitted signal can thus be defined as its highest relevant frequency component. Also, as mentioned earlier, the spectrum of the signal depends, to a large extent, on what data is being transmitted and basic system specifications such as the data rate.

In general, communication channels have a limited available bandwidth. For example, the human auditory system can capture and process signals inside the 20Hz to 20kHz band. This means that, acoustic signals that have a lower bandwidth, can not be "fully" heard and understood, as is the case with frequencies that exceed the upper limit. A typical approach to tackle this issue is by applying carrier wave signaling techniques, whereby the signal information is somehow embedded in a sinusoidal wave oscillating at a desired frequency.

Digital signaling schemes, can be roughly separated into three categories: Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK). These techniques consist of embedding information on time-varying parameters of a sinusoidal wave, respectively its amplitude, frequency and phase. Simply speaking, it means that, given a certain baseband signal $p(t)$ containing data, this signal will be present as either the amplitude-over-time, frequency-over-time or phase-over-time values of the emitted signal. Essentially, instead of encoding data as voltage levels over time, these symbols are "keyed" into other signal characteristics like its amplitude, frequency and phase, respectively for ASK, FSK and PSK. Also, it is interesting to notice that, given that these characteristics are effectively independent, these techniques can be combined to improve the signal's information density. For the purpose of a software based PHY encoder/decoder, ASF and PSK are the more interesting techniques when compared to FSK. This will become more clear over the following sections, but essentially, these signalling schemes produce more predictable spectrum profile, allowing for the implementation of more efficient techniques, such as orthogonal frequency division multiplexing (OFDM).

2.1.4 M-ary Quadrature Amplitude Modulation

The process of converting symbols into an ASK or PSK signal can be simple. Symbols are mapped to certain amplitude or phase values, and these values are used to modulate the carrier wave, generating the symbol's corresponding signature signal. For example, given a carrier wave oscillating at a fixed frequency F_c with phase Φ_0 and amplitude A_0, and a symbol k, mapped by the phase value Φ_k, the respective PSK signal can be written as:

$$psk_k(t) = A_0 \sin(2\pi ft + \Phi_0 + \Phi_k) \tag{2.1}$$

Similarly, the ASK signal of the same symbol mapped to an amplitude value A_k can be written as:

$$ask_k(t) = A_0 A_k \sin(2\pi ft + \Phi_0) \tag{2.2}$$

One important aspect to note from equations 2.1 and 2.2 is that the phases and amplitudes of the resulting signal always appear in relation with the phase and amplitude of the carrier wave. This means that, for the receptor to extract accurate symbolic information from the received signal, it must know what are the values of the base amplitude and phase values, A_0 and Φ_0 respectively, of the carrier wave. This is a matter of coherence, whereby the transmitter and receiver must be synchronized to be able of communicating with each other. Furthermore, other channel effects such as attenuation, delay and phase response are going to affect the perceived amplitude and phase of the received signal, and will become a compound problem when trying to achieve coherence. Achieving proper timing synchronization and amplitude compensation is one of the major challenges in the implementation of PSK and ASK systems. Section 3.2.1 offers an explanation of the strategies applied.

While theoretically speaking, there is no upper limit on the symbolic complexity, or the number of bits per symbol, that fit into the same signature signals, in reality, as the symbol constellation becomes more dense, i.e. the distance between different symbols, the bit error rate increases. This is because a received signal can not be perfectly remapped into the original constellation: the more noisy the channel, the less precise the resulting map will be. For this reason, one heuristic to have in mind when arranging the symbol mapping, is that the constellation points should be as distant from each other as possible.

By combining PSK and ASK signaling techniques, the information density of each of the signature signals can be further improved. This means that more symbols can be embedded into the same waveform, while at the same time maintaining a reasonable distance between the constellation points. The idea is straightforward: each symbol is mapped in both phase and amplitude "coordinates" so that the composition of both coordinates provides more leeway in terms of constellation density. Figure 2.1 shows a symbol constellation map illustrating this composition, where each cross corresponds to a different symbol defined as specific complex number.

This signaling modulation scheme is called M-ary Quadrature Amplitude Modulation (M-QAM) because the signal can be decomposed into two quadrature carrier waves, with an arbitrary number M of constellation points. However it is important to consider that while there is no absolute upper or lower limit in the mapped phase values, special consid-

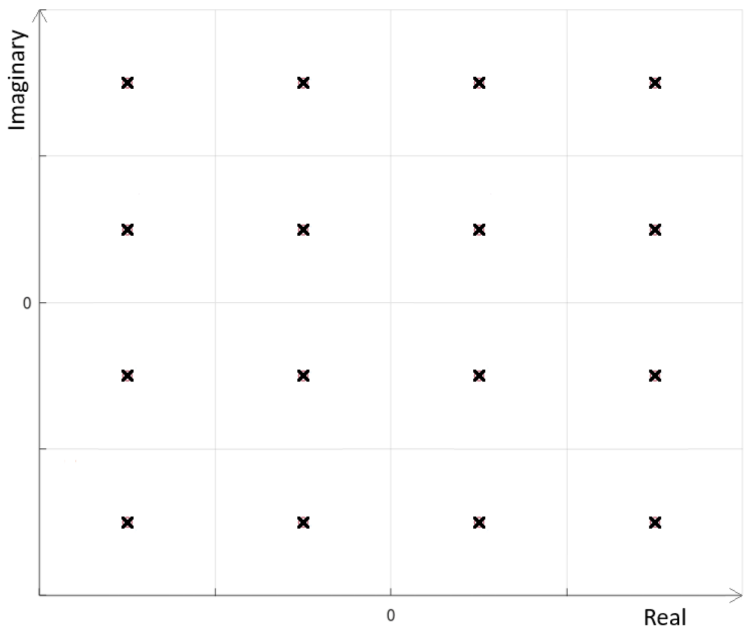

Figure 2.1: Generic illustration of a QAM-16 constellation map.

eration must be taken for the amplitude ranges. The symbol's amplitudes are, at the very least, limited by the the dynamic range of the transmitter, or else non-linear behaviour due to saturation effects will become prevalent.

2.2 Physical constraints in real-world applications

Before moving on towards the description of more specific data communication schemes, it is important to consider what are some of the real-world effects that have to be taken into account when designing the system. In general, the degree to which we must consider such effects depends on the "harshness" of the system's conditions. For example, there's little adaptation efforts when transmitting low frequency electric signals over short copper wiring: the channel bandwidth is comparatively huge and there are barely any noisy disturbances present in the line. In these cases, PCM is often used as a modulation scheme and very simple hardware efforts are involved. However, when trying to use a given interface to its fullest potential, some physical barriers will become more apparent. This section goes over some important physical effects that are relevant for the implementation of the communication system over the speaker-pickup pair.

2.2.1 Noise

The concept of noise consists of the part of a signal that is chaotic and contains no relevant meaning. Generally, noise is unwanted as it interferes with the original information contained in the signal and, in many cases, may entirely corrupt it. One way to think of noise in this context is to imagine a pristine painting laid out on the floor, and as sand starts pouring slowly onto the canvas the ratio of "sand to painting" increases. With time, the picture becomes less and less discernible, until eventually it turns entirely indistinguishable .

Noise is caused by many sources and can behave very differently depending on the physical processes involved. For example, thermal noise is commonly mentioned as it occurs in every single component of an electronic circuit, arising from the random movement of electrons due to the temperature of materials. This type of noise can be modelled as being additive, i.e. summed with the original signal, and white, i.e. spread uniformly across the spectrum.

Another interesting example to consider is the noise generated from converting digital signals into analog signals, e.g. when generating an audio signal with a phone's speaker, and conversely, the noise generated when converting analog signals into digital signals, e.g. when recording the guitar's pickup sensor signal with an analog-to-digital converter. This is denominated as quantization noise and it arises from the fact that digital signals can ever only be stored, recorded and reproduced with discrete signal values, in the form of digital data. Interestingly, this type of noise is more affected on the shape of the original signal: if the signal weak, its voltage levels are described with a lower number of significant bits and thus the digital quantization effect has a greater impact.

When thinking about the speaker-pickup pair, there is one more relevant source of noise that comes to mind: with the pickup sensor located just under the guitar strings, any disturbances in the strings will be propagated as a signal. As it turns out, from empirical trials, it becomes evident that this noise does have a significant impact on system performance, and depending on the setup, the user may be required to help dampen the vibration of the strings during transmission. Even though random string vibration can still be modelled as an additive noise source, its spectrum is not uniform as it is more prevalent near the natural frequencies of the strings.

Despite there existing vastly different models for noise generation, for the purposes of analysis of this project all noise considered will be idealized as Additive White Gaussian Noise (AWGN). This simplification distills all noise to a single noise signal that: is added to the original signal, has a uniformly distributed power spectrum and it follows a Gaussian distribution with an average value of zero. As such, for a given system setup, the noise floor defines the average value of noise, in terms power per Hertz.

The signal-to-noise ratio, typically witten in dB units, represents the ratio between the power of the signal over the power of the noise: $SNR = \frac{P_s}{P_n}$. The system can be more or less resistant to noise, depending on how good is the data accuracy despite lower SNR levels. Typically, a better SNR performance is inversely correlated to higher data-rates. This is because a signal that packs more information with the same amount of transmitted energy will likely be more susceptible to noise disturbances.

2.2.2 Channel response in LTI systems

When travelling through the channel interfacing a transmitter and receiver pair, the signal may suffer some additional transformations that are not accounted by noise alone. While some of this behaviour is hard to model, in reality, most communication channels can be accurately modeled as linear time-invariant (LTI) systems, which are systems that apply predictable linear transformations and whose behaviour does not vary with time. As it so happens, the speaker-pickup pair can be approximately modelled as an LTI system, as long as its behaviour is isolated in smaller chunks of time. The main reason for this distinction is because, since both the speaker and the pickup are usually hand-held, the geometry of the pair may suffer changes over time. Nevertheless, since these variations occur at relatively low frequencies, it turns out that the speed at which the channel behaviour changes is slow when compared to the signal frequency band, and so, it turns out that the channel can still be modelled as an LTI system.

LTI systems can be described by a transfer function that establishes how the signal is exactly transformed. Typically, the transfer function is written in the frequency domain, often called impulse response, and the spectrum of the resulting signal, $Y(f)$, can be calculated from the application of a complex scalar multiplication of the transfer function, $H(f)$, and the input spectrum, $X(f)$. This operation represented in eq. 2.3, and it showcases how simple it is to predict the output behaviour of a given system once its transfer function is characterized.

$$Y(f) = X(f) \times H(f) \tag{2.3}$$

As seen in eq. 2.3, any given frequency component of the output signal, depends exclusively on its respective input and the system's transfer function. In a passive channel, where no signal energy is introduced during propagation, the signal will some suffer attenuation, corresponding to the reduction in power. As the attenuation levels increase, the SNR levels will typically decrease, and so, the passband of a given channel is defined as the frequency-band in which the attenuation has not exceeded a certain threshold. This concept is particularly important as it defines the working band of the transmitted signals and it can have a large influence on the upper limit of the data rates. Ideally, a channel should have a large passband, which means more bandwidth, located at higher frequencies, so that the baseband signal can have a relatively small bandwidth when compared to the frequency of the carrier wave. This will become more evident as the concept of frequency multiplexing is introduced in section 2.3.

When a channel is limited in terms of its "available" band, in particular, when its cutoff frequency is relatively close to the that of the carrier wave, the signal shape changes in such a way that it is spread over time. Essentially, successive signals windowed in time lose their ability to transition immediately, and so, parts of the signal remain and continue propagating until, eventually, they die out. This factor is responsible for inter-symbol interference (ISI), whereby signals related to different symbols interfere with each other, with an increasing degree the closer in time they are located to each other.

Finally, as mentioned earlier, the input's signal phase is also affected when crossing the communication channel. While this factor does not influence the signal's energy or bandwidth it is also important to consider, especially when information is embedded in phase information as is the case of PSK signalling. In such cases, strategies must be set in place to compensate for any arbitrary phase deformations that took place, so that the original symbols may be extracted from the resulting signal.

2.2.3 Audio to digital conversion

So far, all signals have been though of as continuous waveforms that can be described in both time and frequency domains, however, in most modern systems, which work in the digital domain, signals often need to be translated into a discrete form. This is because symbolic data can only ever represent a finite amount of information, and so, signals need to be discretized if they are to be generated, processed and stored on digital platforms. For audio-based systems, the central components that is responsible for such a conversion is the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC), and the quality of these components will have a large impact on the quality of the link between transmitter and receiver.

In section 2.2.1, this topic is briefly mentioned in the context of the noise introduced from digital discretization, meaning inaccuracy in the signal due to the quantization of continuous values. However, one aspect that has not been touched upon is that of the finite sampling rate of any given digital conversion systems. In audio systems the ADC and DAC components sample the signal at a fixed rate denominated as sampling rate and, during the sampling process, some of the continuous signal information is lost due to the fact that signal records are discarded of the periods in between sampling instants.

The Nyquist theorem states that, if a continuous baseband signal has a bandwidth B and is perfectly sampled at a fixed rate f_s, then, the original signal can be accurately reproduced if and only if $2B \leq f_s$. Also, components of the original signal that correspond to higher frequencies, exceeding $\frac{f_s}{2}$, are mirrored back into the lower frequencies, i.e. they end up interfering with the lower frequencies. This means that, ideally, the continuous signal spectrum should avoid "spilling" over this frequency threshold. In essence, not only must the communication system be designed to operate at channel bandwidth, it is also constrained by the sampling rate of the receiver.

2.3 An effective signaling scheme for passband channels

In section 2.1.4, M-QAM was introduced as a signaling modulation scheme capable of encoding symbols of arbitrary complexity into a single carrier wave. This signal can be "positioned" at a desired frequency, depending on a channels frequency response characteristics, and the symbols can be extracted from amplitude and phase information alone. In this section, a frequency multiplexing scheme is introduced as a way to improve data rates and the system's overall performance, taking into account the real-world physical effects discussed in section 2.2.

If the spectral distance between two passband signals is large enough, it is possible to isolate each of these signals, assuming the the bandwidths don not overlap. Analog radio transmission systems work under this principle, since there is no baseband signal constraints, other than limits in signal bandwidth. However, in the case of the speaker-pickup pair interface, it is assumed that the speaker has the entire control over the available bandwidth. This fact can advantageously used to improve the transmission signal's spectral efficiency, which can be described as $\rho = \frac{R_b}{B}[bit/s/Hz]$, where R_b represents its bit rate, and B represents its bandwidth.

2.3.1 Orthogonal frequency division multiplexing

Orthogonal frequency division multiplexing (OFDM) is a frequency multiplexing technique that allows for the spectral overlapping of multiple carrier signals in such a way that, theoretically speaking, no information is lost. By packing the frequency components tightly together does the spectral density improve dramatically and more carriers can be used. Furthermore there is also tremendous advantages related to the signal processing to be performed on software-based platforms, which will become more apparent in chapter 3.

The OFDM scheme works under the principle of orthogonality of time-windowed sinusoidal signals, whereby the central spectral component of two signals does not suffer from the presence of the other, even though the spectrum of the signals may overlap. Essentially, given two sinusoidal signals $x_A(t)$ and $x_B(t)$, of frequency f_a and f_b, windowed in time by a rectangular pulse of duration T_D and separated in frequency by a distance of $\Delta_f = k/T_D$, ($k = 1, 2, 3, ...$), then, the frequency components at those frequencies are independent of the presence of the complementary signal: $X_A(f_B) = 0$ and $X_B(f_A) = 0$, where $X_A(f) = \mathcal{F}(x_A(t))$ and $X_B(f) = \mathcal{F}(x_B(t))$. Figure 2.2 illustrates the spectrum of a generic OFDM signal consisting of two neighbouring carriers, where we can clearly see that the respective amplitude peak of one corresponds to one of the null-points of the other.

As it so happens, the M-QAM signature waveforms (modulated carrier waves) can easily satisfy the properties described above. For example, if the N carriers are located at distance of Δ_f apart from each other, then the spectrum obtained from the sum of the

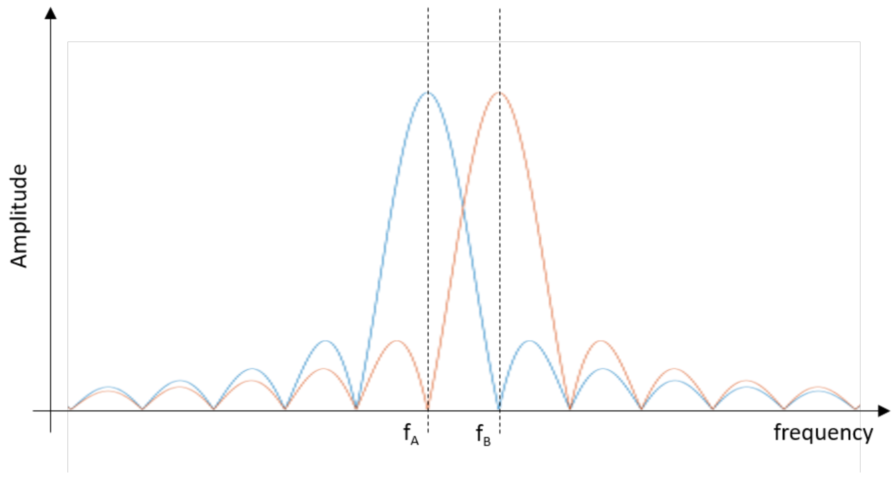

Figure 2.2: Spectral representation of two carriers using the OFDM scheme.

carriers during the windowing pulse duration will show overlapping spectra and the symbol information can be extracted from the amplitude and phase values of the frequency components of each respective carrier.

The parameter T_D is defined as the symbol's duration, and defines how fast the signal is switching between consecutive symbols, transmitted as OFDM signals. With a longer symbol duration, the minimum Δ_f value decreases and so the carrier waves can be more tightly packed together, and so, the data rate does not directly depend on the symbol duration/rate, but rather only on the available bandwidth B and the number of bits transmitted per carrier M. Assuming no signal overhead, the theoretical data rate limit R_{max} can be derived from the Nyquist rate theorem.

$$R_{max} = B\log_2(M) \tag{2.4}$$

3 Design overview

The theoretical fundamentals laid out in chapter 2 presented a set of models and techniques that can be used as the building blocks for a working communication system, however, their application in a real-world implementation brings forth a few challenges. This chapter aims to describe a scheme in which these principles are applied towards a functional design that can be viably implemented on an embedded processor. Section 3.1 provides an overview system's sub-modules for a practical application, and section 3.2 offers an in-depth view of the signal processing algorithm and while taking a deeper look into the construction of a single data packet signal.

3.1 System design considerations

In most cases, it is hard to isolate the design choices of the various system parameters and basic components. These are often interdependent, and so, some of the outcomes are the result of theoretical estimates, simulations, and practical prototyping. Here, we discuss some considerations that are not explicitly mentioned in chapter 2 but represent nonetheless factors that had an important influence in the implementation of the final design.

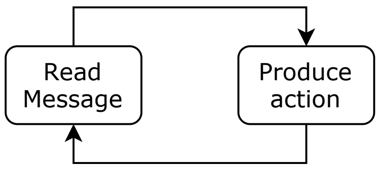

Figure 3.1: Simplified representation of the application model.

3.1.1 Application message formatting

Throughout this project, from an application point of view, messages are seen as blobs of information that can be translated into independent commands transmitted from a host device. Figure 3.1 represents a simplified model of the device operation, where the emphasis here is that a single message should be able to contain all the relevant information that is required to performs a certain action, an also, since the communication channel is unidirectional, messages should not really depend on any specific state or previous transmissions. At this level, messages should be encoded in a format that best suit the data structures being transmitted and the system's requirements.

This communication interface is used to deliver device settings for the Darkglass product line, allowing the device to reconfigure itself during operation. For this purpose, the message formatting scheme is designed around the specifications defined by the Resource Interchange File Format (RIFF), originally developed by Microsoft and IBM [11]. The RIFF is a specification designed for multimedia files, which can be used as the message structure, providing clear and simple guidelines for generating files as a collection of chunks. Since all chunks are identified, only the very basic structure of the format components have to be statically assigned, in terms of memory location and value. The construction of a Darkglass RIFF is simple: all files are generated as a set of an arbitrary number of consecutive chunks of varying size. Even though all chunks need to be designed according to the needs and features of the application, the actual construction of the RIFF is flexible. adhering to the following rules:

- All files start with the *RIFF* chunk, containing information about the whole file. It contains three fields: the *chunkID* (4 bytes) depicting the ASCII string "RIFF" itself, the *chunkSize* (4 bytes) which contains information on the size of the whole file and the *Format* field (4 bytes), defining the type of file at hand. The formatting rules of this chunk are shared by all file formats, so it is recommended that new file implementations respect the construction of this chunk as well.

- Every chunk following the first chunk, also called subchunks, start with a header region which is similar in construction as the *RIFF* itself. It contains two fields: the *subchunkID* (4 bytes), identifying the subchunk type which will depend on its desired function, and the *subchunkSize* (4 bytes), containing information on the size of the subchunk. The content of the subchunk itself is located immediately following the subchunk header. For example, the content of a *data* subchunk simply consists of byte array data, while other commands have statically assigned fields containing all relevant information.

Due to the flexibility offered in the construction of the RIFF, the software complexity is also kept simple. As the information required to translate the RIFF is embedded in chunks of the RIFF itself, the translation algorithm is not directly tied to the content type and does not need to be rewritten for every new file or subchunk type that is added. Also, application messages can be efficiently encoded and, from the designer's perspective, they are simple to construct in support of new device features.

3.1.2 Data integrity

According to the specifications defined in table 1.1, the communication system must yield lossless data transmission, which essentially means that no message should be accepted until its integrity can be verified to a reasonable degree. The simplest way to achieve this is by applying a hash function that generates a signature of a given data set. The hash function should be such that, if two data sets are different, then, there is an extremely high probability that the hashes generated from each of the data sets will differ as well. So, the integrity check process simply consists of recalculating the hash of the received data-set and comparing it with the hash already present.

Even though it was not designed explicitly for this purpose, the cyclic redundant check algorithm (CRC) can be used as a hash function. It not only works as a hashing function, but it can be used as a rudimentary error correction code, meaning, the CRC code can be used to correct corrupt data in the received data-set. While some alternative hash algorithms can be more appropriately used for data integrity checks alone, the CRC is ubiquitous and is often present in embedded platforms as dedicated hardware accelerators. This latter aspect can turn out to be quite relevant as it helps decrease the computational expense of this process.

3.1.3 Splitting messages into packets

From a transactional point of view, the communication protocol is designed towards the transmission of independent messages, made up of a variable number of fixed-size packets. This approach simplifies the system design, in the sense that no additional signalling is required for dictating the packet decoding control-flow, and at the same time, the packets efficiently tailored towards the specific information density of the OFDM signals. Packets should be large enough so that the fixed packet overhead information, i.e. packet header and data integrity codes, is small when compared to the total packet size, and they should be adequately small to improve granularity. This latter aspect is particularly important when considering lossless transmission over a noisy channel since smaller packets are less likely to have contained corrupted bits.

The packet size can be chosen to be such that it complies with a whole number of symbols transmitted by the OFDM signals. For example, if a packet is transmitted over L OFDM signals, then the size of the packet can be directly derived from the number of

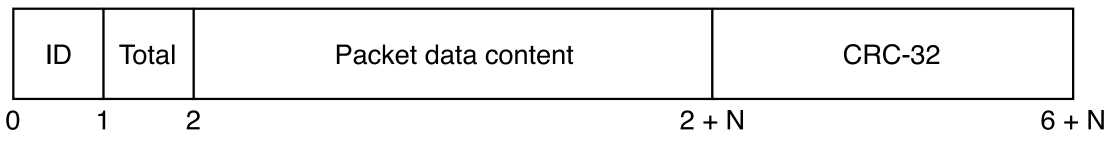

Figure 3.2: Representation of the packet structure.

bits transmitted by this set of signals. So, if each OFDM signal contains M' bits, then the number of content bits N per packet can be calculated as: $N = M'L - 6$. Such an approach ensures that no signal is effectively "wasted", while simultaneously maintaining complete independence between separate packets.

The packet structures represent the atomic piece of data, and since the communication system is unidirectional by nature, with the possibility of losing packets mid-transmission unbeknownst to the transmitter, the packet must include some additional information regarding its place in the overall message. For this project, we have opted for the most straightforward strategy consisting of identifying each packet two additional byte numbers, the first one corresponds to an identification byte for identifying its slot within the message, and the second one represents the total number of slots. Thus, the receiver can simply derive the starting position of the packet content bytes by multiplying the identification byte with the total length of bytes per packet.

The diagram presented in figure 3.2 is a byte-mapped representation of the packet structure, where the "identification" byte and "total size" bytes can be seen in index 0 and 1 respectively. The data content segment is of variable size and depends on the number of payloads transmitted per packet, which will be further discussed in 3.2.3. Finally, each packet is terminated with a CRC-32 hash with the intent of verifying the validity of the decoded information. Naturally, invalid packets are not counted towards the completion of a message.

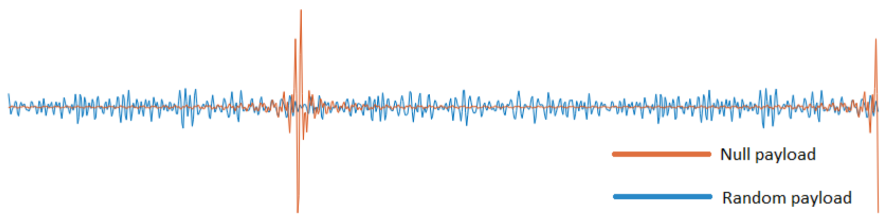

Figure 3.3: Illustration of the data dependent peak-power-to-average-power-ratio.

3.1.4 Reducing the peak-power-to-average-power-ratio

One key challenge with OFDM systems has to do with the containment of the peak-power-to-average-power-ratio (PAPR). Essentially, the OFDM signal carrier waves may line up in such a way that the resulting wave concentrates a big portion of its energy in a short time-span, leading to quick bursts of energy. These bursts should be avoided as they may result in the breaking of the peak power limitations or dynamic range of the emitter/receptor, leading to signal distortion and, consequentially, loss of signal information. The graph shown in figure 3.3 demonstrates the behaviour of the automatically generated OFDM signal for two opposite situations: the red curve is generated with an all-zero data content showing mostly a concentration of the signal's power near the central peak, and the blue curve was generated with random data and clearly shows a much lower peak-to-average-power value.

While there are exist promising strategies to tackle this issue [12], the simplest way to avoid this situation is by generating data entropy, i.e. reducing the likelihood of non-optimal phase alignments in the signal due to its apparent randomness. For this purpose, all outgoing data is "scrambled" by XORing the data with a predefined array, immediately before translating the data into the transmitted signal, and the same operation is performed by the receiver for all incoming data, immediately after extracting the symbolic data from the transmitted signal.

3.1.5 ISI and the guard interval

As mentioned in section 2.2.2, a signal travelling through a band-limited channel will be spread over time even shortly after the ending of its transmission, similarly to the perceived effect of sound echoing. In the case of OFDM signals, this phenomena causes the occurrence of ISI due to how past symbols negatively affect future symbols, acting similarly to noisy channel effects. To decrease the impact of ISI, the most straightforward and effective strategy is to introduce guard intervals in between succeeding symbols. The purpose of these guard intervals is to allow the remnants of previous symbols to "die out" before the transmission of a new symbol can start. The greater the length of the guard interval, the less ISI will occur, however, the data rates naturally decrease due to increased waiting periods in between symbols. So, the interval should be chosen in a way that is adapted to the characteristics of the channel, in particular, the *channel length*.

3.1.6 Timing synchronization challenges and the cyclic prefix

The OFDM scheme can only be implemented in coherent communication systems, i.e. systems in which the transmitter and receiver are synchronized in time. This is particularly important due to the fact that each carrier wave contains information embedded in its relative phase and any timing offset Δ_t results in phase shifts proportional to the frequency of each component f_k, with the relation $\Delta_\varphi = 2\pi f_k \Delta_t$.

Consider the scatter graph that is shown in figure 3.4, where all the carrier components are represented in the complex plane and the default QAM-16 constellation map are marked with black crosses. The remaining dots show the resulting image with slight component phase variations of φ, where we can see that variations in phase will eventually lead to wrong conclusions about the symbolic value of a carrier. If we consider a 48kHz sampling rate at the receiver, a timing estimation that is wrong by exactly one sample will result in a ± 1.3 radians phase shift at frequency components near the 10kHz mark, and thus it becomes clear that without any additional phase compensation algorithm, this ap-

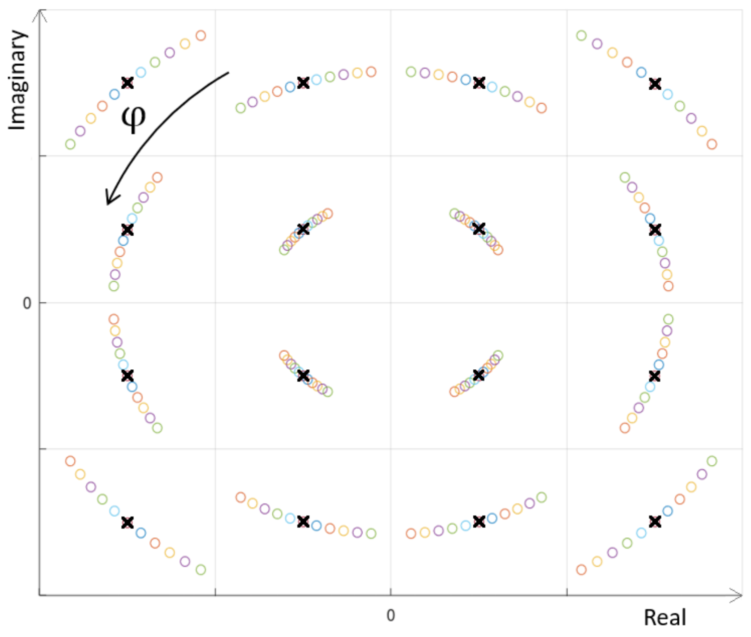

Figure 3.4: QAM constellation map and the resulting rotations due to phase offsets φ.

proach will most likely always yield wrong results. So, no matter how well implemented the system synchronization techniques are, the timing can only ever be precise to a certain degree, always leading to warped components, when looking at the scatter graph. Moreover, with higher precision requirements naturally follows an increase of complexity and overhead in the synchronization components.

The cyclic prefix (CP) is a signal which is prepended to the original signal, consisting of a replica of its ending. It is an essential concept for OFDM system as it allows reduced precision requirements on the timing synchronization techniques. It is easy to understand the idea by imagining an ever-repeating OFDM signal: the resulting spectrum of the symbol, sampled at any point in time, will be the same, except for a linear phase change. This linear phase change can then be estimated with a higher degree of precision through a separate process explained in section 3.2.1.

The CP typically has a shorter duration than the symbol itself, but theoretically, it may have any arbitrary length. Given that the system is not constrained in terms of its energy efficiency, the CP can least occupy the duration of the guard interval. This way, as the timing synchronization instant approaches the start of the symbol, or equivalently the ending of the CP, the effective SNR of the sampled signal improves.

3.2 Signal processing algorithms

The overall communication system's signal processing algorithm can be fully comprehended by deconstructing the transmission signal of a single packet. So far, the focus has been on how to encode data in a passband signal through the OFDM signalling scheme, however, two other packet sub-signals are critical for the construction of the packet signal in its final form: the mode-switch signal and the preamble. These, bring forth the remaining information required for the receiver to "know" how to extract the relevant data, and so this section looks at what is their purpose, how they are constructed, and finally, how they are inserted in the packet's final form.

3.2.1 The Preamble

As mentioned earlier, the OFDM signalling scheme relies on emitter-receiver timing coherency, meaning that the receiver should know exactly what period consider when decoding the physical signal. The intuition behind this requirement is quite straightforward: the OFDM payload signals are transmitted for a finite period in time, and even with the addition of the redundant CP, the receiver system must be able to identify these periods and sample the signal accordingly, or else some of the information may be lost. Simultaneously, the receiver should be able to identify the predictable transformations that the signal undergoes when propagated through the communication channel. Here, we first look at a timing synchronization signalling scheme that is embedded in the preamble and is used to provide a good estimation of the synchronization instant. Then, a method is presented to calibrate the system for the channel effects, reusing the same information

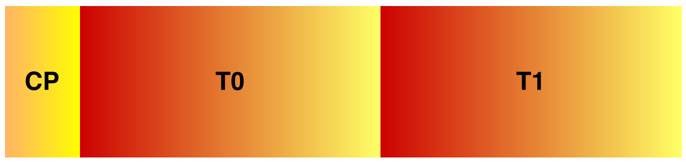

Figure 3.5: Representation of the preamble signal consisting of the CP and two consecutive training symbols: T0 and T1.

embedded in the preamble signal. Finally, we present a simple frequency offset compensation scheme that is used to adjust the system to small sampling frequency discrepancies.

Schmidl & Cox OFDM timing synchronization scheme

The timing synchronization scheme is adapted from the works published by Timothy Schmidl and Donald Cox (S&C), where they present an efficient system for calculating precise timing for the start of a frame, or packet, of a given signal [13]. The idea consists of the transmission of a preamble signal, at the beginning of the packet, whose purpose is to inform the receiver of a timing reference from which it can derive the sampling instants for all remaining OFDM signal payloads. The preamble signal consists of two consecutive training symbols, T0 and T1, which can be exact replicas of each other, preceded by the CP as depicted by figure 3.5. The authors go on to describe specifications for the design of these training symbols, however, for our purposes, they can simply remain equal to each other, which will offer additional advantages as explained in section 3.2.1.

The underlying idea behind the synchronization algorithm consists of calculating the instant at which a signal of fixed length consists of two equal parts or two equal training symbols. In reality, as mentioned earlier, the training symbols will never be perceived by the receiver as being exactly similar, so, the main challenge is in defining a method to reliably identify this similarity. One idea that naturally occurs would be to continuously calculate the discrete correlation between two consecutive signals, each composed of the same number of samples corresponding to the length of the training symbol. The algorithm proposed by (S&C) builds on this idea, providing a method to derive the timing synchronization instant from the calculations of three metrics that can be efficiently computed in real-time for a given sample with the index d: $P(d)$, $R(d)$ and $M(d)$.

The metric $P(d)$ is a measure of auto-correlation and intends to represent the degree of similarity between the training symbols. It measures the correlation between two subsets of a signal $x(d)$ spaced L samples apart and, importantly, can be calculated with an iterative approach as shown in eq. 3.1. The system simply needs to keep track of the last $2 \times L$ samples, and the point where $P(d)$ has the highest value should indicate the synchronization instant.

$$P(d+1) = P(d) + x(d)x(d-L) - x(d-L)x(d-2L) \tag{3.1}$$

This metric, however, only reveals half of the picture since the measure is not normalized for signal power, i.e. if the $x(d)$ is large in amplitude, the resulting $P(d)$ will be larger regardless of the actual correlation. For that reason, the authors included a metric $R(d)$, which measures signal energy for the duration of the last L samples, and can also be iteratively calculated according to eq. 3.2. Finally the last metric $M(d)$ can be calculated from eq. 3.3 and represents a measurement of the "L-distanced" autocorrelation, normalized to the signal's energy.

$$R(d+1) = R(d) + x(d)^2 - x(d-L)^2 \tag{3.2}$$

$$M(d) = \frac{|P(d)|^2}{|M(d)^2)|} \tag{3.3}$$

One important question to consider at this point is what type of signal to transmit as the training symbols. The only constraint naturally imposed by the S&C algorithm is that both signals be replicas of each other, however, there are a few more things to consider taking into account the real-world application. A signal coded with the Zadoff-Chu sequence (ZCS) is often used in timing synchronization schemes due to having good autocorrelation properties [14], meaning that the autocorrelation value has a maximum at the null offset and close to zero for every other point. Heuristically speaking, this provides a greater assurance the preamble signal will offer a better sensitivity when calculating the autocorrelation metric $P(t)$. Also, it can be verified that this signal provides a good peak-power-to-average-power-ratio and it has the added advantage of being evenly spread on the frequency spectrum.

Channel calibration

Even if the timing synchronization is accurate, the channel may change the signal behaviour to such an extent that it does not directly translate into the OFDM symbols expected at the receivers end. As covered in section 2.2.2, these transformations are predictable and tend to operate similarly if the channel conditions remain the same, meaning that if the receiver can extract a model of that behaviour once, it can repeatedly predict and compensate for that model. This is the process of channel calibration and, the easiest way to achieve it is to receive a fixed signal and determine what is the inverse channel transformation that generates the original signal. For all the future OFDM symbols, that same compensation transformation can then be applied to generate the intended signal.

The preamble adopted for the timing synchronization scheme can also be used to transmit fixed reference signals as training symbols, already known from the receivers end. Then, the complex channel compensation factor for each spectral component k, c_k, can be calculated from the values of the expected, e_k, and received, r_k, components: $c_k = \frac{e_k}{r_k}$. From then on, the receiver simply applies the compensation factor to the respective component of the following OFDM symbols, thus extracting a good estimation of its actual value. Notice that the spectral band of the training symbols should correspond to the same band as the one used by the OFDM symbols so that the signal energy is contained only inside the relevant frequencies.

It is also interesting to notice that this channel calibration scheme also compensates for slight offsets in the timing synchronization, as long as the synchronization instant is located somewhere within the limits of the CP. This is because any time delay, positive or negative, simply generates a linear phase transformation that can easily be accounted by the same calibration procedure, i.e. it is as if the delay was a channel effect as well. Naturally, though the synchronization instant should still be as accurate as possible to improve the perceived SNR of the following symbols.

As mentioned earlier, for the speaker-pickup pair, the channel behaviour depends on external factors that change over time, such as the distance between the emitter and receiver. If the packets are long, containing multiple OFDM symbols for each respective preamble, it may be the case that the initial compensation factors calculated during the channel calibration phase do not accurately model the channel behaviour in the later payloads. To improve on this design, one can use the OFDM symbol information to have the system adapt to the ever-changing channel over time, even during the transmission of the same packet. This is called a soft calibration, and the idea is to compare the received signal with the symbol's expected signal for each QAM symbol in a given OFDM payload. Essentially, if we assume that a considerable chunk of the signal's deviation is due to the channel behaviour, the component compensation factor can be iteratively updated by weighing in the new differences. Eq. 3.4 demonstrates one viable approach: for every new OFDM payload $i + 1$, the updated factor $c_{k,i+1}$ is calculated from a weighted sum of the previous factor $c_{k,i}$ and the ratio between the expected component $e_{k,i}$ and the received component $r_{k,i}$. Notice that, in this case, $c_{k,0}$ is the first compensation factor calculated during the channel calibration, and usually the weighting factors should be chosen such that $w_0 > w_1$.

$$c_{k,i+1} = c_{k,i} w_0 + \frac{e_{k,i}}{r_{k,i}} w_1 \tag{3.4}$$

Frequency offset compensation

So far, all design decisions have been made while considering perfect system oscillators, meaning that the transmitter and receiver sampling frequencies perfectly align with their theoretical values. Notice that not only are the absolute frequency values tremendously crucial for maintaining carrier orthogonality with the OFDM scheme but also, the precise timings of the packet signal depend on this factor as well. For this reason, a frequency offset compensation scheme is presented here that can be used to compensate for small sampling frequency discrepancies with little additional processing load.

The works presented by Sliskovic (2001) showcases a process for accurately estimating the frequency offset of an OFDM signal [15]. The idea is to transmit two consecutive training symbols that utilize the relevant frequency components and to compare the phase of the resulting spectrums. When assuming no noise, the frequency-offset Δ_f produces a frequency-dependent phase shift φ between both training symbols, according to a factor ε, as shown in eq. 3.5.

$$\varphi(f) = \varepsilon(\Delta_f)f \tag{3.5}$$

The correction methodology presented by the previous author assumes that the signals are sampled at the precise instants, however the frequency-offset also produces an ever-increasing timing misalignment that must be taken into consideration. Since all the essential signal timings are derived from counting the number of samples occurring since the synchronization instant, the frequency offset has the effect of changing the timing offset proportionally to this number. This timing offset ends up producing an effect very similar to the one described by eq. 3.5, by inducing a phase shift to all frequency components, while the value of this shift is proportional to the number of samples that span between both sampled signal.

To estimate and correct for this effect, we reuse the preamble's training symbols to derive the relevant phase shift values according to eq. 3.5, for each relevant component.

Through simulation results, we conclude that the bulk of the perceived distortion is actually due to timing misalignment induced by the frequency offset. As such, the corrective solution applied for our solution can be reduced to a timing offset compensation algorithm, that is described as follows:

- First, we separately apply the FFT over each training symbol t0 and t1, obtaining the real and imaginary values of each of the relevant frequency components. We expect that the resulting spectrum T0 and T1 be the same, except for the phase shifts produced by the effects of the frequency offset.
- Divide each component of the spectrum T1 by each component of T0 and derive each of the phases φ_k, which corresponds to the phase shift described in eq. 3.5.
- Do $\Delta_{\varphi_k} = \varphi_k / N$, where N represents the number of samples composing the training symbol. This factor represents the induced phase shift per sample for each frequency component.
- The value of Δ_{φ_k} can be used in different ways. One way is to calculate the correction phase shift at each relevant point throughout the packet signal by multiplying this factor with the number of samples it takes to reach that point. Alternatively, a more optimal approach is to calculate the phase correction factors that can be continuously re-applied to the channel compensation factors, in order to compensate for this behaviour over time. This approach assumes that future sampling instants are equally spaced apart, which happens to be the case for the signalling scheme used for this system.

3.2.2 A computationally efficient mode-switch signalling scheme

One of the key challenges in the system implementation phase consists of integrating the communication system in a way that it can sustainably cohabit with the remaining device systems. Theoretically speaking, the process mentioned above for finding the start of a frame/packet could be used as a way to detect the start of message transmission, however, while still relatively efficient, this approach would consistently require a still excessive amount of continuous computing power. Here is presented an efficient alternative algorithm for the continuous scanning of the start of a new packet.

The mode-switch detection algorithm consists of the identification of a pre-defined signal consisting of a specific sequence of successive sinusoidal tones. This frequency pattern defines a tone sequence, or *tone-key*, which the algorithm tries to match with its internal key value. The graph presented in figure 3.6 shows the time-based representation of a tone sequence, composed of four different and consecutive sinusoidal waveforms. This effectively corresponds to an FSK type, where symbols are defined in terms of frequencies and transmitted in succession.

There are two great advantages for using sequences of "purely" sinusoidal waveforms for transmitting the signature mode-switch signal. The first one has to do with the most basic behaviour of LTI channels: a waveform composed of a single sinusoidal component will always maintain its shape, i.e. the receiver will observe a sinusoidal signal of the same frequency, even if with a different phase. Secondly, for sinusoidal waves, its frequency can be estimated by calculating the number of zero crossings over a defined period. The algorithm is thus based on a zero-crossing counter, which tracks the number of zero-crossings during successive time slots of the recent signal. Given a pre-defined tone sequence, the mode-switch algorithm consists of the following:

- The process maintains an array that works as bin-counters, one for every number of tones present in the sequence and ordered accordingly. The purpose of this structure is to track how many zero-crossings are counted in the time-span corresponding to a given tone, relative to the last input audio sample.

- Simultaneously, it also maintains a cyclic structure called the *cross-map*, tracking the zero-crossing instants, in terms of distance to the most recently processed sample. The *cross-map* should include at least 1 bit per sample, indicating whether a given time instant corresponded to a zero-cross, and it should have a length equal to the length of the entire tone sequence. Each tone is then mapped to a region in this structure, or tone slot, which continuously shifts for every new sample that is computed.

- For every new sample value, it checks the value crosses the reference zero value, i.e. if the new sample has a different signal when compared to the previous sample. Note that it is usually necessary to include some hysteresis in this calculation to avoid false positives due to low magnitude noise. When a zero-crossing is detected, it is marked as a "1" in the most recent index of the *cross-map*.
- Then, the idea is to shift the *cross-map* regions progressively as the index increments and update the bin-counters with the number of zero crossings present in each region. For example, if a new zero crossing is detected, the bin-counter for the last tone will either increment or remain the same, depending on whether the updated *cross-map* region contains one more or the same number of zero crossings.
- The bin counters can be efficiently updated by propagating the information from the last bin counter, all the way back to the first bin. For example, if a *cross-map* region corresponding to a given tone slot "loses" one zero-crossing, then the region corresponding to the preceding tone will now include the zero-crossing that was yielded.
- Finally, every time the bin-counters are updated, they are values are compared with a reference pattern which holds the expected number of zero crossings per bin for the given system configuration. If the difference between the expected and calculated pattern is small enough, the mode-switch signal has been detected.

It is clear that the computational complexity per sample of the detection algorithm is very low, involving only fast memory accesses and some additional operations. The

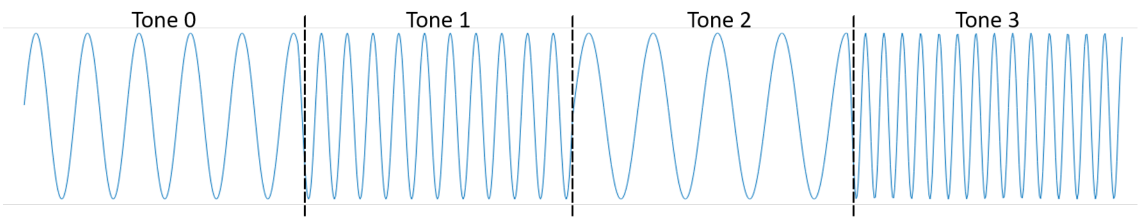

Figure 3.6: Illustrative example of a tone-sequence time signal.

selectivity of the algorithm can increase indefinitely with the size of the tone-sequence, at the expense of a slight linear increase in computational load. It is important to note however that, similarly to the ISI effects, the waveforms of immediately successive tones will exert influence on each other, leading to a different number of zero-crossings per tone on the receivers end. This can also be solved with a guard interval scheme, by, for example, ignoring the number of zero-crossings during the first half of the transmitted tone.

3.2.3 OFDM payloads

In section 2.3, OFDM is described as the underlying data signalling encoding scheme. With this method, data symbols are encoded in carrier waves transmitted simultaneously, placed in specific locations of the frequency spectrum in such a way that the resulting inter-carrier interference is kept to a minimum. Each OFDM signal, or OFDM payload, can transmit several symbols at a time, one per carrier wave and each symbol can itself contain information about a number of bits. The information density of a given OFDM signal is thus not only dependent on the number of carrier waves, which directly depends on the channel bandwidth, but also on the number of bits per carrier itself.

For this project, the carrier waves are designed as 16-QAM signals, which can express 4 bits per carrier, striking a good balance between data rates, i.e. constellation density, and reliability, i.e. inter-point distance. From eq. 2.4, and given the channel bandwidth B, we can calculate the maximum theoretical data rate with this signalling scheme to be

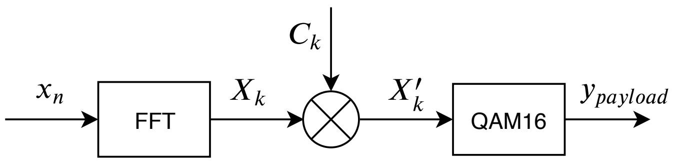

Figure 3.7: Representation of the OFDM payload processing algorithm.

$R_{max} = \log_2(16)B = 4B$.

The process for extracting the symbolic payload data is represented in figure 3.7, consisting of a total of three stages. First, the discrete time payload signal x_n is converted to its frequency domain equivalent, X_k, through the FFT operation. Then, the frequency spectrum is multiplied with the channel and frequency compensation coefficients, C_k, which are calculated during the timing estimation, channel calibration and frequency-offset estimation phase and updated for every new processed signal block. The resulting frequency-based signal X_k should contain an accurate representation of the OFDM payload spectrum, as generated by the transmitter, and so, each individual carrier wave can be decoded using the QAM-16 scheme in reverse. For that purpose, it suffices to check which QAM-16 constellation map point is nearest that of each component X'_k.

3.2.4 Signal construction

Having gone through all the separate signal components, it's time to put it all together and showcase the progressive packet decoding stages. The communication system processes the signal as it is being transmitted, and as such, the order of the several processing stages are located following the order of the respective sub-signals transmitted in sequence. Figure 3.8 presents an example of the first part of the time graph of a packet signal, for an arbitrary format configuration, where the emphasis is on the start of the transmission and the stage progression starting from the mode-switch signal, going to the preamble and finally onto the OFDM payload stages.

All packets start with the *Mode-switch* signal so that the receiver can react to the start

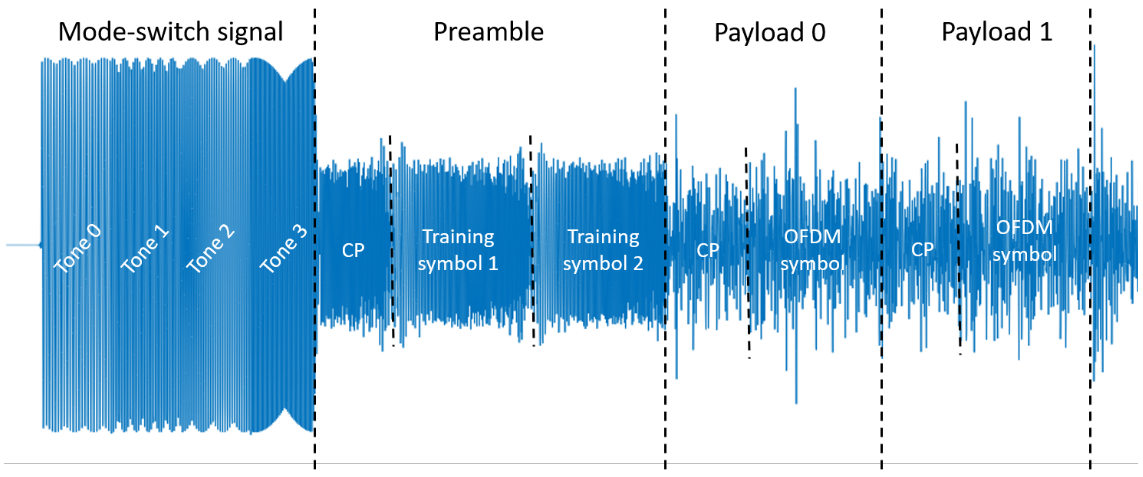

Figure 3.8: Time graph of the packet signal construction for an arbitrary format.

of the packet on time to start computing the next stage. As we can see, this sub-signal is much greater in amplitude than the remaining others, which is due to the fact this stage is much more resistant to signal saturation distortion effects, since the only relevant information is the points of zero-crossing. Also, it is important to note that all packets include this sub-signal so that the receiver can identify the message transmission from any arbitrary point in the message, which is cyclically transmitted.

Immediately following is the *Preamble signal*, which will serve to provide a good estimation of the synchronization instant and the frequency component compensation factors for channel calibration. Naturally, the stage precedes every the first payload stage because this information is crucial for decoding of the OFDM payloads. After this stage, the system can then infer all critical instants within the remaining signal.

Finally, several *Payload* signals are sent in succession for an arbitrary number of times. Each *Payload* is encoded using the OFDM signalling scheme and contains a subset of the entire packet data. As information is extracted from each payload, the data can be buffered into an internal packet buffer so that it can be processed later. Once the entire packet is decoded, the CRC value can be verified, and if the validity is confirmed, the packet content data can be copied to the appropriate slot of the final message buffer.

4 System implementation

The following chapter aims at clarifying the most relevant details regarding the software implementation of the communication system. First, some specifications regarding the target platform architecture are briefly mentioned in section 4.1, then, section 4.2 looks at some methods and schemes applied for the implementation of the system, and finally, section 4.3 serves to showcase the implementation outcomes and discuss the more relevant performance results metrics.

This system implementation has been designed for the ATSAM4S4A chip [16], from the ARM M4 processor family [17], nevertheless, the details described in this document can be directly applied to several other MCU architectures. The relevant hardware system supporting the microprocessor is straightforward in principle and can be entirely abstracted from. The fundamental peripheral hardware module consists of an interface for communicating the digital audio data, sampled from the output of the pick-sensor. This interface can either be in the form of a serial-audio-interface (SAI) connected to an external ADC/codec, or it can be more directly accomplished reading the signal directly from the internal ADC inputs, included in the SOC. More detailed specifications regarding the system setup and constraints have been discussed in section 1.1.

4.1 Platform Characteristics

This project runs on a 32 bit ARM M4 processor, clocked at 120MHz. The Cortex-M4 family comes equipped with digital signal processing (DSP) features that are incredibly useful for the relevant signal decoding algorithms, however, for this particular chip, the processor core does not offer a dedicated hardware floating-point unit (FPU). Even though the acoustic channel is of relatively low bandwidth, during the project development phase it became obvious that some of the implementation decisions were largely constrained by the available computational power, and that the system processor usage should be configurable to an extent.

Let us consider the following example. The system is running at a fixed sampling rate $f_s = 48$kHz, so, if the processor is to process data in real-time, the speed at which it can process each task set related to a block of samples should be greater than the speed at which new blocks of samples originate. This means that the average processing time per sample must be no greater than the sampling period, equating to around 21μs processing time per sample. In order to extract the symbols from each OFDM payload, the processor must at least apply an FFT operation over the block of samples buffered for that signal's length. Knowing that the FFT operation has a computational complexity of $O(N \log(N))$, it means that the greater the length of OFDM signal, the more time per sample it takes to process such block. For OFDM decoding, longer blocks are typically preferred so that the GI and CP lengths are kept relatively short, leading to a more efficient implementation. However, after benchmarking the FFT performance it became clear that these blocks should be no longer than 512 samples.

4.1.1 DSP Features

The ARM Cortex-M4 processor system packs several DSP features [18], which can be essentially described as internal processing hardware that facilitates and improves the performance of common data processing functions. Single-instruction-multiple-data type instructions (SIMD) are one prime example of how hardware can be modified to improve digital processing throughput. These special instructions can simultaneously operate over multiple data fields at once, and in the case of the Cortex-M4 SIMD instructions work with sets of 8-bit or 16-bit values at once with 32 bit wide registers.

Even though DSP instructions can only be directly accessed through lower-level software, typically in the form of C macros, the ARM CMSIS DSP libraries provide wrappers for more complex high-level functions such as FFT, or even simpler and common operations such as vector arithmetics. Also, since the library's API is abstracted to the actual hardware, the code can be migrated between different ARM platforms while maintaining very high computational performance.

4.1.2 Fixed-Point Arithmetics

As mentioned above, the processor at hand does not include an FPU, and so, all decimal point operations are always performed from a combination of integer operations. Sometimes, to reduce programming complexity, it makes sense to emulate floating-point operations which, with the assistance of the compiler, are automatically translated into integer-based instructions. However, a more optimal alternative to writing computationally efficient code is through the direct application of fixed-point arithmetics, in such a way that the system submodules are then adapted for performing fixed-point calculations with little to no redundancy.

Essentially, fixed-point numbers are integers that represent decimal numbers. This means that decimal operations using fixed-points arithmetics can be easily translated into integer operations. This representation sets the decimal point at a specific bit position, usually defined as the Q format. For example, the Q1.30 format is a format containing one integer bit and thirty fractional bits, with an extra sign bit. When two numbers in this format are added with integer addition, the resulting number remains in the same format, however, if they are multiplied together, the resulting number will default to Q2.62 format. Essentially, even though fixed point arithmetics allows for the implementation of very efficient calculation procedures, the challenge in using the fixed-point arithmetics is that the data needs to be constantly converted into the correct formats in order to maintain correctness, and the risk of overflow is usually much higher.

4.1.3 Embedded Memory

Embedded devices typically aim at being extremely cost-efficient, meaning that at least the processor chip's die area should be kept to a minimum. One of the ways to achieve this is by reducing the amount of available system volatile and non-volatile area, and in the case of the ATSAM4S4A, this factor is showcased by the relatively low amounts of system memory, in the form of 64kB of Static Random Access Memory (SRAM), and limited non-volatile memory region as well, with a total 256kB of flash storage. These characteristics imply that certain strategies need to take place to not only guarantee that there is enough memory to run all the firmware systems but also run them reliably.

The memory model for the device firmware, including the new communication system feature, is such that there is never any dynamic memory allocation and all stack memory is kept to an absolute minimum. The reason for this is because it is hard to predict/calculate the usage of heap memory during run-time and guarantee that it's kept under a certain threshold, while it's very easy to compute the memory usage from data structures that are statically allocated. As for the stack usage, it is perhaps even harder to determine the worst-case scenario and accurately predict the required minimum amount of stack required for a given program, even if running directly on a given processor without any operating system abstraction. So, the system memory map can be simplified and designed in such a way that the stack/heap shared memory region is large enough to accommodate for the stack memory requirements, with a reasonable amount of slack. While the trade-off of this approach is that it may lead to redundant memory usage, if for example two of the system's sub-modules use internal temporary buffers requiring two separate and non-overlapping static memory allocations, it's also an approach that leads to a more reliable design with faster development times.

4.2 Implementation Overview

In figure 4.1 is shown a flow diagram representing the system as a sequential process. There, we can visualize the processing steps, starting from the physical signal processing, and going all the way to the final message. It has been laid out in a way that is tightly related to the actual program processing flow. The following is a description of each stage:

- **Mode-switch detected?** The mode-switch detection process is always running in the background, and its sole purpose is to notify the system that the start of a packet has been detected. This is particularly relevant from a system integration's point of view since the processing resources are scarce and must be shared by the entire system.
- **Search for SOF** The search for the start of frame (SOF) consists of the calculation of the M metric, mentioned in section 3.2.1, and includes the calculation of the packet synchronization instant.
- **Found SOF shortly after?** The SOF is ever going to occur in a given time window after the detection of the mode-switch signal. If the search exceeds this expected deadline without identifying a SOF, it means that it can not guarantee precision in timing synchronization, and so, the process falls back to normal mode and ignores the following packet signal.
- **Channel compensation factors** Once the synchronization instant is known, the program can uses the preamble signal to compute the channel compensation factors, according to the algorithms described in section 3.2.1.
- **Frequency-offset compensation factors** The frequency-offset compensation factors are also calculated using the information present in the preamble signal, as also described in section 3.2.1. These factors need to be calculated separately as they are used to update the channel compensation factors over time.

- **Wait for next payload** Before decoding an OFDM signal, the program must wait for the transmission of a number of samples corresponding to the payload block size.
- **Apply comp. factors** The OFDM payload is translated into the frequency domain and the relevant components are corrected using the channel calibration and frequency-offset compensation factors. This is done by first applying the frequency-offset factors over the channel compensation factors, and only then the resulting factors onto the payload's spectrum.
- **Decode OFDM** During this phase, the program computes the data by translating the payload samples of the OFDM signal. Once the data is extracted, it is appended to an internal packet buffer.
- **Update comp. factors** The compensation factors are updated using the information from the decoded signal, i.e. by assuming that all symbols are correct and calculating the resulting error.
- **Was last payload?** The packet contains several payload blocks, and so, the payload processing sequence must be repeated once for all the payload blocks.
- **CRC verified?** At the end of the packet data extraction, the 32 bit CRC value is calculated and compared with the corresponding field embedded in the packet data. Note that before performing any operation over this data it needs to be "unscrambled" according to the process mentioned in section 3.1.4.
- **Write packet into message slot** The packet corresponds to a chunk of the entire message under transmission. Due to the cyclic nature of the unidirectional message transmission process, these packets do not necessarily arrive in sequence and so the content data needs to be copied to the appropriate chunk of the message buffer.

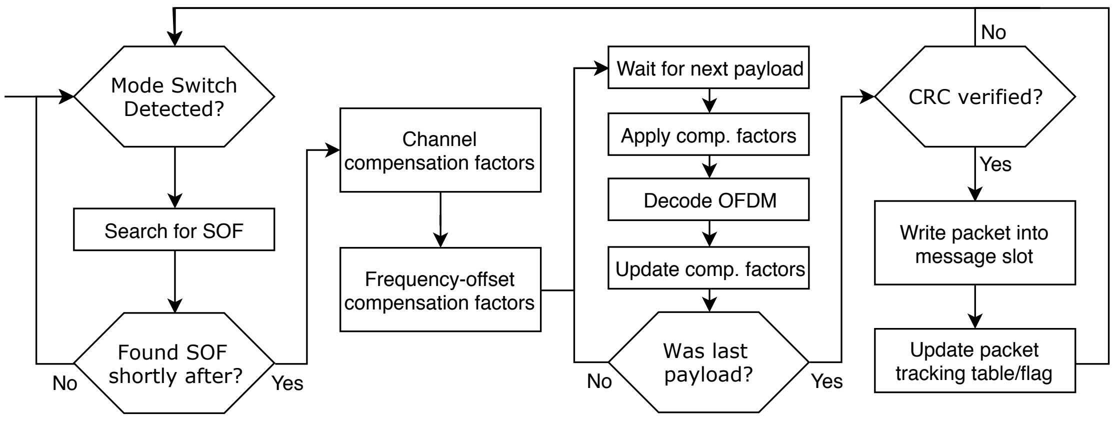

Figure 4.1: System component level flow diagram.

- **Update packet tracking table/flag** The message is only transmitted once all the corresponding packets have been received. For this purpose, a binary table is updated every time, in order to determine whether the packet is new, and a counter is maintained to determine whether the full message has been received. Once the message is complete, a flag is set to signal that the message buffer contains a valid message.

4.2.1 Product integration

As mentioned in section 1.1, the acoustic communication system is to implemented across many capable devices of the Darkglass product line, and naturally, this system must be added as a supplementary and independent feature. This means that it is running at a lower priority level relative to the other basic product functions, such as digital audio signal processing, and also, responding to the user interface and external switches. As such one of the major challenges is in integrating the feature in a way that it is simultaneously accessible, i.e. easy to activate and use, but also while limiting its ability to interfere with the remaining software systems.

Mode-switch detection

The system integration pattern is designed around the idea of two different modes of operation: the STANDBY mode and the ACTIVE mode. When in STANDBY mode, the communication system's processes run at a low priority level while barely loading the CPU, and the system is tasked with scanning the audio signal for the start of a new message. Once the start of a new message is found, the system switches to an ACTIVE mode, where it starts to run at a high priority level and using most of the processor time available.

When active, the communication system must be continuously running in the background so that it can immediately detect an incoming communication whenever the user wishes to use the feature. While this greatly improves the accessibility and the ease of use, it also means that there must exist a process that can continuously scan for the start of a new message. With the remaining device modules running concurrently within the system, more notably the audio FIR filters, the processor time available is very limited, and so, this background process needs to be relatively light. Moreover, the process also needs to be selective enough so that it does not issue false positives, i.e. indicating that there is an incoming message signal when there is not.

The Mode-switch detection scheme, already introduced in 3.2.2, is a crucial component for system integration, providing a way for it to continuously poll the audio channel while running as a "background" process with a very low processor usage rate. Empirical tests show that this scheme provides a high selectivity rate without interfering with the remaining processor systems.

Processing model

The system is running in "baremetal" mode, meaning that the firmware is running directly, without an underlying operating system abstraction layer. While the program flow depicted in figure 4.1 shows a sequential progression of the system, in reality, the underlying process runs off two concurrent threads. The number of samples contained in each packet signal is too large to be buffered all at once, so the DSP necessarily needs to run in a real-time fashion, i.e. by simultaneously sampling and processing the incoming audio signal.

The audio data is communicated through the SAI interface and is routed to an internal buffer in system memory through the use of peripheral-to-memory direct memory access (DMA). Through DMA, the system can be set up in such a way that the samples arriving at the SAI FIFO are copied to an internal buffer without requiring any intervention by the CPU. Essentially, this allows the processor to operate over blocks of samples, when a DMA interrupt is issued, rather than being interrupted for every new audio sample.

The need for two concurrent threads comes from the fact that we cannot guarantee that the heavier computations can be completed in the time that spans between two consecutive blocks of audio samples. Some computations such as the OFDM payload processing, not only require somewhat complex mathematical calculations such as the FFT but also typically involve a large number of samples. Fortunately, thread concurrency can be achieved through the handling of the DMA interrupts occurring at a fixed time rate, i.e. once a new block of audio samples has been written in the buffer. Every time this interrupt routine is called, it preempts the main loop, and so it is as if there is a thread running concurrently at a higher priority level.

The system's processing model is depicted in figure 4.2, showing the two concurrent threads, Master and Slave, side by side. In this case, the Master thread corresponds to the interrupt callback routine, and the Slave thread corresponds to a function running repetitively inside the main loop. The Master thread is in charge of maintaining the serial audio interface flow, by storing the incoming samples and issuing new transfers, and at the same time is also in charge of dictating the tasks for the slave thread to perform.

In this model, the Master thread can communicate with the Slave thread by writing values to the Slave's state variable, depicted in the picture as *slave.state*. The Master is

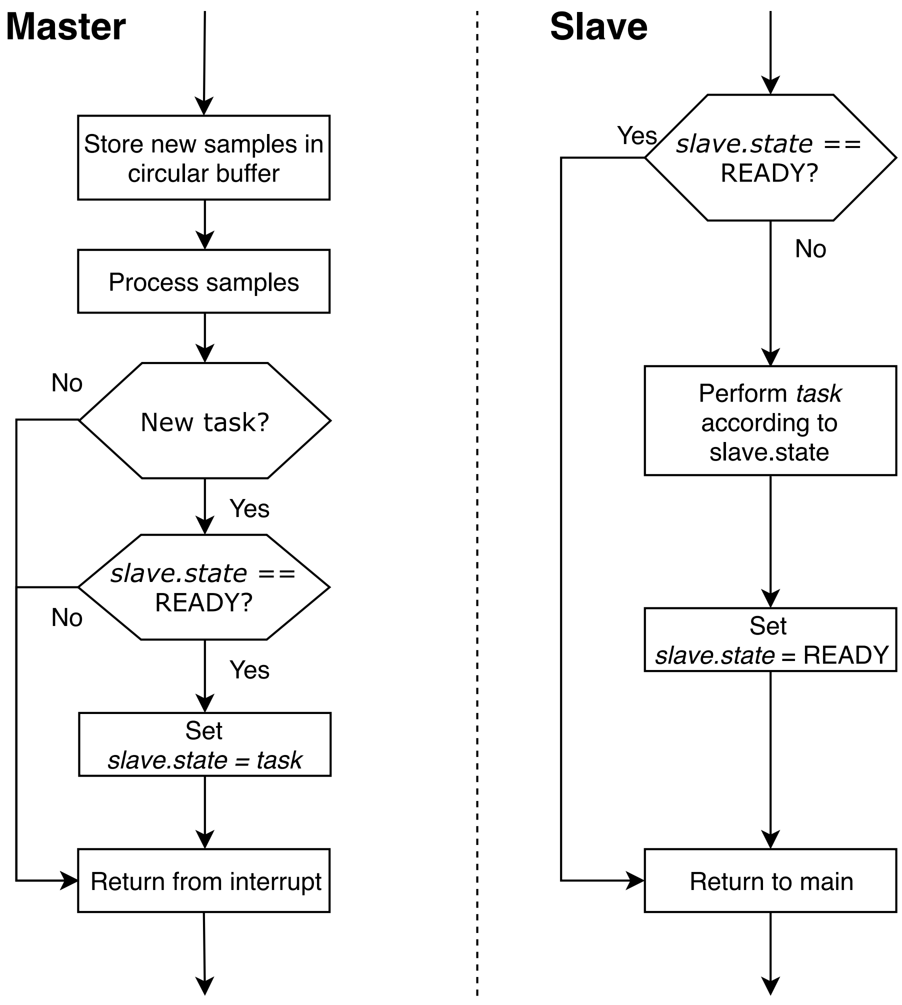

Figure 4.2: Processing model diagram depicting the concurrent master and slave threads.

only ever allowed to set the variable if the *slave.state* is set to state READY, and the Slave thread is only allowed to return the variable to READY state. With this scheme, and since this is a single processor system, we can guarantee that there will never be any race conditions regarding the access to the *slave.state* variable.

The *task* mentioned in the processing model diagram in figure 4.2 can correspond to several different operations, depending on the packet processing stage. For example, once the timing synchronization instant is identified, the Master thread will issue a 'Calibrate channel' task to the Slave so that it can start processing the preamble signal information and obtain the channel compensation factors. Nevertheless, the Master thread will also perform some light DSP operations, as is the case for example of the mode-switch detection routine discussed later on.

4.3 Implementation results

To test out the implementation, the system is running in development mode on the *Microtubes X Ultra distortion pedal*. This section aims at showcasing the fundamental behavioural characteristics of the system's implementation on the ATSAM4S4A platform, namely its runtime performance metrics and noise immunity.

4.3.1 System configurations

The system's modules have been implemented in a way that they support different configurations, depending on the format of the transmission signal, such as OFDM payload, CP length and the bandwidth usage of the signal itself. These configurations may directly impact the transmission speed and they will essentially determine whether the implementation is viable or not, depending on whether the microprocessor is powerful enough to compute the underlying processes at the required rate. The following is a list of the main system configuration parameters:

- **CP length (CPL)** corresponds to the number of samples per cyclic prefix, i.e. the length of this signal. Larger numbers lead to longer guard intervals, less ISI, however, it also increases the amount of signal overhead, due to the fact that it contains redundant information. It is also possible to increase the length of this parameter to artificially increase the slack time between consecutive OFDM payloads.

- **OFDM payload length (PL)** corresponds to the number of samples per OFDM signal. The frequency resolution of the signal's DFT depends directly on this number, meaning that we can pack more symbol carriers. However, larger PLs also mean an increase in computational complexity, as mentioned in section 2.1.1.

- **Carriers per payload (CPP)** is the number of M-QAM symbol carriers per OFDM signal. Essentially, for a fixed PL value, this parameter translates the bandwidth of the system, so, the larger its value, the more symbols are packed into a single payload. Also, a greater CPP value will inversely decrease the average power per carrier, which negatively influences the perceived SNR at the receiver's end.

- **Index of the first carrier ($\mathbf{C_0}$)** corresponds to the index of the first frequency component of the OFDM payload signal. This component can be calculated for a given sampling rate f_s as $f_0 = \frac{f_s C_0}{PL}$, and so, this parameter works hand in hand with CPP in defining the signals band usage.

- **Payloads per packet (PPP)** is the number of symbol payloads that form a single packet. With more payloads being packed into the same packet, there is more content data and less overhead caused by the preamble and mode-switch signals. Nevertheless, fewer payloads also mean finer-grained packets and less data buffer memory requirements.

- **Modeswitch length (ML)** defines the length of the mode-switch signal. This length influences the selectivity of the mode-switch pattern, i.e. by decreasing the likelihood of detecting false positives of the mode-switch signal.

Table 4.1: System configuration parameters used for the performance test trial.

Parameter	**Value**
f_s	46.875 kHz
CPL	256 samples ($\simeq 5.4$ ms)
PL	512 samples ($\simeq 10.8$ ms)
CPP	120 carriers ($\simeq 11$ kHz bandwidth)
C$_0$	44 ($f_0 \simeq 4$ kHz)
PPP	16
ML	1024 ($\simeq 21.6$ ms)

For system testing, the testbench configuration parameters were decided on based on empirical trials, by tweaking parameter values until arriving at a setup that shows simultaneously the best performance and reliability. Table 4.1 provides a compilation of the system configuration parameters used for this test. Considering that packets are continuously transmitted ad infinitum, the theoretical maximum data transmission rate for this system configuration is around 3kB/s ($\simeq 3065$ B/s).

4.3.2 Frequency response of the channel interface

As mentioned earlier, the communication channel, denominated as the speaker-pickup pair is mainly composed of two elements: the phone's speaker serves as the acoustic transmitter, and the bass/guitar pickup serves as the receiver. The performance of the entire system strongly depends on the quality of the signal, as it travels through the communication channel, and conversely, the behaviour of the channel depends mainly on the type and quality of these components. It is safe to say that there is no single configuration that can run on all possible configuration, for example, some lesser quality speakers may distort the signal to the extent that the data is then unobtainable from the receiver's end.

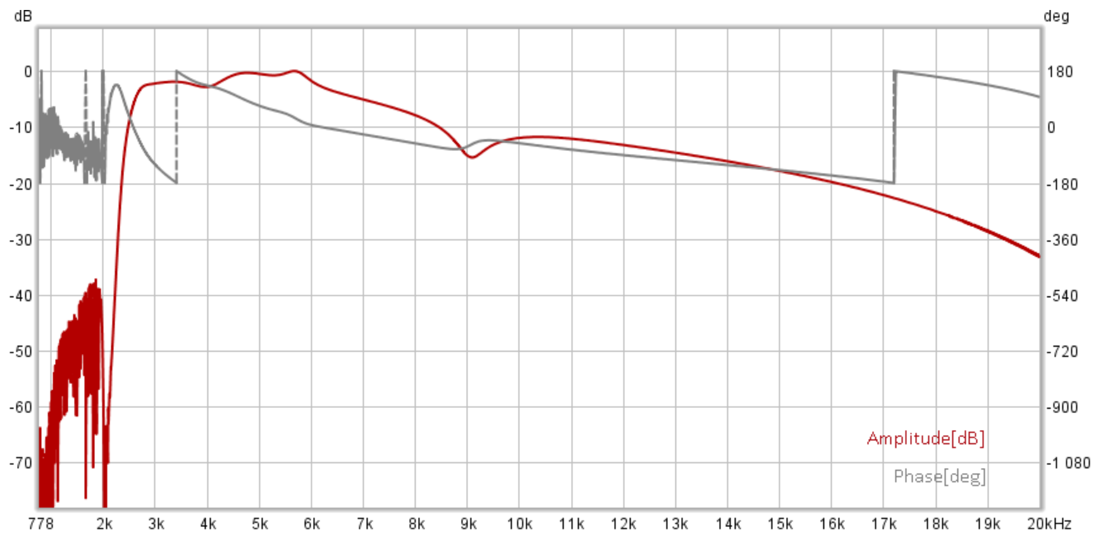

Figure 4.3: Channel frequency response with the speaker-pickup pair setup.

Nevertheless, from the company's perspective, it is vital to find at least one system configuration that works reliably for most users. For this project, the system has been modelled around a setup consisting of a generic bass pickup and an iPhone 8 speaker.

The graph that is shown in figure 4.3 represents the normalized frequency response of the given setup. One of the most noteworthy characteristics is the channel's passband, i.e. the frequency band that least attenuates the signal, which is important because it significantly influences the value of the received signal's SNR. By setting a threshold at -20dB, meaning a 10x normalized attenuation factor in the worst-case scenario, we can roughly consider the passband to be from around 2.5kHz to around 15 kHz. Also, notice that the phase response is not linear, which confirms an earlier assumption made in the design phase.

Finally, the other important characteristic to consider is the impulse response length, which influences the amount of ISI present in the system, as mentioned in section 3.1.5. The graph in figure 4.3 represents the amplitude, in normalized dBs, of the impulse response, derived from a sinusoidal sweep measurement. As with channel bandwidth, the definition of the impulse length dramatically depends on the application at hand, and for this case we may arbitrarily decide that the ISI should be no higher than -20dB lower than

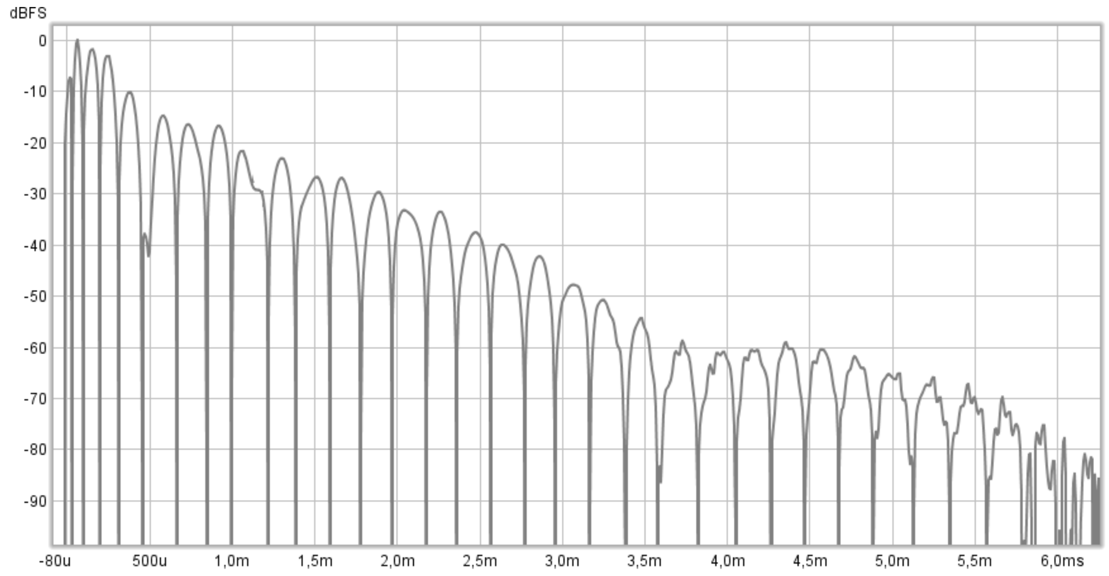

Figure 4.4: Impulse response's amplitude over time with the speaker-pickup pair setup.

the most attenuated frequency, i.e. the impulse length corresponds to the time instant at which the amplitude of the impulse response does not exceed -40dBs (normalized).

4.3.3 Runtime performance analysis

The SEGGER Systemview toolkit was used To conduct the runtime performance analysis of system implementation [19]. This application profiling tool allows us to track events issued by the system, such as the start and end of specific tasks, with microsecond precision, very low computational overhead, and all in a realtime fashion. It means that we can extract an almost exact model of the running system behaviour and, for this analysis, we are interested in measuring the computation times of different processes and determine the viability of different configurations.

Figure 4.5 showcases a sequence of processes running over a period of around 320 ms. It corresponds to the period in which a message packet is being interpreted, with each coloured block corresponding to a different process running from the master thread, slave thread and other unrelated device modules. While there are other concurrent system processes, the following is a descriptive list of the more relevant types that explicitly appear

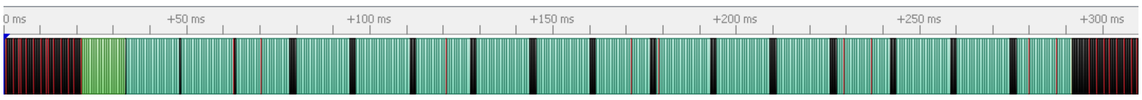

Figure 4.5: Performance analysis profiling result with a default system configuration

as coloured blocks in the diagram due to requiring more processor time:

- **Black** blocks represent the *idle* process, which, roughly speaking, includes every process that is not directly related to the communication system.
- **Red** blocks, here rather appearing as stripes, represent the master thread processes, i.e. the processes that are called upon the arrival of every new SAI DMA block.
- **Green** blocks correspond to the *calibrate channel* task and the **Blue** blocks correspond to the *OFDM decode payload* task, both performed by the slave thread.

At the start, the processor is mostly idle, while being periodically interleaved by the master thread. The red stripes, even though not always explicitly visible, appear at a fixed rate throughout the whole time-span, however, during this first phase, it is noticeable that these are thicker in form, indicating that the master thread is loaded with more substantial computations. In this case, the master thread is tasked with the *'mode-switch detection'* task, i.e. trying to identify the signal right at the start of a new packet.

Immediately following this initial phase, once the mode-switch signal is detected, the preamble related tasks are issued, and the slave thread takes over most of the processing time. After the preamble is transmitted, the OFDM payload decoding tasks are issued for every OFDM payload transmitted in sequence, appearing as the blue sections. Even though the different payloads are not explicitly distinguishable, for the most part, they should be separated by the black stripes. In essence, this indicates that the slave thread has successfully completed those computations on time, and can withhold of processor usage in the meantime. With this system configuration, the packet is composed of a total of 16 OFDM payloads, which can be confirmed by looking closely at the diagram. After decoding all 16 payload blocks, the cycle repeats, and the system returns to a mostly idle state, while the master thread goes back to running the mode-switch detection task once more.

A crucial metric that can be inferred from the runtime analysis is the processor load. From the previous analysis, we see that the system appears to be running close to its computational load limit. Ideally, there should be more idle blocks in between the slave tasks, i.e. a longer slack time to decrease the restrictiveness of the deadlines. Nevertheless, these results show that the current configuration is able to run reliably under average system load conditions.

4.3.4 Noise immunity

Assuming that CPL and PL remain constant, the system's data rates directly depend on the number of OFDM carriers used, which translates into the bandwidth of the signal. This spectral utilization should not only be adapted to the channel's passband but must also consider how the noise immunity varies with different numbers of OFDM carriers. For example, in the case of the speaker-pickup pair, where the 4 kHz to 16 kHz band typically shows the best SNR performance, the number of carriers can vary between 1 and 120, given the default system configurations presented in Table 4.1.

To test the system's noise immunity, we collected bit-error-rate (BER) data for different test signals artificially infected with additive gaussian noise. The graph represented in Figure 4.6, where the BER appears plotted against the SNR for different system configurations. From these empirical results, we define the noise immunity level as the SNR value at which the BER drops bellow 100 ppm, translating into the successful transmission of over 99 % of all packets. As expected, configurations with fewer carriers show improved noise immunity. Furthermore, as expected, the noise immunity is directly proportional to the average power per carrier, i.e. if we halve the number of carriers noise immunity improves by 3dB.

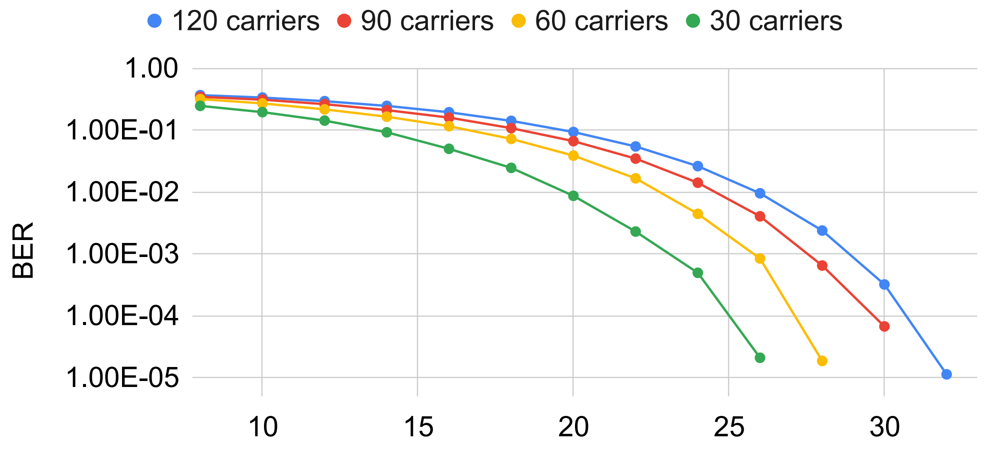

Figure 4.6: BER vs SNR with different bandwidth utilizations.

4.3.5 Testing the limits

To test the limits of the final system implementation, we experimented with different settings of the system while trying to achieve the highest data rates and relatively reliable transmission. With $CPP = 160$, and all other configuration parameters set to the default values, the system was able to wirelessly transmit a 160 kB message in 40 s without dropping a single packet. These settings correspond to a data rate of 32 kb/s, corresponding to 33% faster speeds than the ones possible with the default system settings.

5 Conclusion

This thesis presents a promising solution for extending IoT connectivity to resource-constrained embedded devices through high-speed acoustic data transmission. The main contribution of this work has been the design and development of an OFDM-based one-way acoustic communication system that can be implemented in a wide variety of microcontrollers. Also, it provides a case study for the implementation communication modems as a software system, while presenting an ample view of the design challenges of a software-based physical layer of the communication stack.

In order to efficiently utilize the available bandwidth of a given channel, the system needs to be set up as a real-time process so that it may adequately operate over streams of incoming data. This aspect proved to be the most significant challenge not only due to the imprecise timing behaviour of general-purpose processors but also primarily due to the challenges of integration and sharing of computation resources with the external software components, running concurrently on the same platform.

The experimental trials show that OFDM succeeds as an effective signalling scheme by providing a simple method for configuration of the system's spectral utilization. With careful carrier placement, the resulting spectrum is such that the frequency band is adapted to the frequency response of the communication channel at hand. Moreover, the total number of carriers per OFDM symbol inversely influences the average power per carrier, and thus, it is used as an additional parameter to control the average SNR value of each component. This behaviour was verified with extensive experimentation results, comparing how SNR influences the BER with different configuration settings. Despite expecting varying frequency responses depending on the specific speaker and pickup devices used as the physical interface channel, we can estimate that the passband is placed in the mid-range acoustic band, i.e. around 5 kHz to 15 kHz. Nevertheless, as we gather more user feedback from their experience with the feature and the deployed devices, the goal is that we can eventually arrive at the optimal configuration settings that lead to a system that is simultaneously reliable and supports high data rates.

The runtime performance of the proposed algorithms can depend significantly on the available speed and DSP features of the underlying platform, and as such, it was necessary to include methods for system configuration parameters that can adapt to the computational load constraints. For processing the OFDM payload symbols, the length of the CP and Payload block can be adjusted according to the available slack as observed during runtime analysis. While the CP should be long enough to encompass the entire GI period, there is no practical upper limit, and so a more extended CP effectively leads to a longer waiting period in between successive payload blocks. As for the Payload block itself, the processing computation complexity is expressed by $\mathcal{O}(N \log(N))$, and so, shorter blocks yield less processing time per sample.

The design of the preamble signal also proved to be effective in achieving system coherency, through timing synchronization, and compensation of the channel-distortion and frequency-offset effects. The start-of-frame detection algorithm and channel compensation factor calculation are shown to be realizable in embedded software, and an optional delay can be appended immediately after the preamble to compensate for any delays required due to the heavier computational load of this process. Also, the *mode-switch detection* signalling scheme has shown to be a surprisingly simple, lightweight and selective algorithm for identifying the beginning of transmission during idle times.

From a data format point of view, the proposed packets-based protocol provides a simple design for lossless transmission of independent messages. While not intended as a one-fits-all solution, this strategy is reliable enough for most application use-cases that don't have strict security requirements. Simultaneously, this scheme also contributes towards lower amounts of data overhead, consisting of only 2 bytes for the packet header, and 4 bytes for the CRC-32 code integrity check code. The overhead corresponds to a fixed total of 6 bytes per packet, which translates into under 1% of the packet length for most reasonable configuration settings. In essence, the signalling overhead due to the *mode-switch detection signal*, *preamble* and CPs far outweighs the impact of the packet data overhead.

To conclude, the works presented in this thesis provide a substantial collection of material that can be applied towards the replication of the implementation of the communication system on virtually any other embedded system with minimal capabilities. There is a compelling motivation for the integration of such a feature in the digital audio processing consumer products market, as it enables a new and exciting way of user interaction at no added production costs. Furthermore, with the proposed approach, we have been able to obtain transmission speeds of over one order of magnitude over the state-of-the-art and the most widely used commercial solutions.

5.1 Future works

While the contents presented in this thesis detail the implementation of a complete and operational system, the result is not by any means a final and optimal solution. In particular, we've identified some key areas of the design that can still be improved in order to enhance the system's overall reliability, computational efficiency, and speed. The following is a brief discussion of these ideas.

5.1.1 Error correction codes

The system discussed in this thesis has the unusual property of being implemented as a uni-directional communication interface while also requiring lossless data transmission. This characteristic makes it all the more critical that the receiver can adequately identify if any corrupted data, and continue waiting until it has received a complete and unadulterated message. The current approach uses a rough integrity check process based on 32 bit CRC codes appended to each packet, however, it would be interesting to consider alternative ways that include some degree of error correction as well.

The Reed-Solomon codes [20] are a common approach to solve this problem, and can even be found in other similar solutions like the one implemented with *chirp.io*'s solution. Similarly to the current method, it consists of a code that can be calculated and appended to a given data-set, providing some extra redundant information that can identify and reconstruct a number of faulty bits. For example, one common code setup is defined as RS(255,223), where for every 255 bytes of data, 223 are actual content, and the remaining is composed of the Reed-Solomon code. With this setup, the Reed-Solomon decoder can identify and recover up to 16-byte errors occurring in any arbitrary location of the data-set.

5.1.2 System confidentiality

For applications requiring a secure and confidential communication link, the solution proposed in this thesis does not fulfil this specification. The system was designed without the premise of data confidentiality built-in, and thus further considerations are necessary in order to achieve this goal. The standard way to implement such a feature is by applying a symmetric-key encryption scheme, such as the advanced encryption standard (AES), where both transmitter and receiver use a common secret key to encrypt and decrypt the messages respectively. Note that due to the uni-directional nature of the communication channel, the keys have to be statically set in both sides before transmission.

5.1.3 Research performance results using fixed-point DSP

For this project, some of the DSP routines are implemented using floating-point operations, namely during the payload processing and channel estimation phase. Given that the test platform does not possess a native hardware FPU, this approach is likely less computationally efficient than an alternative implementation using fixed-point arithmetics. The extent to which this is true greatly depends on the computer architecture of the underlying platform, nevertheless it would be interesting to evaluate the performance results of this alternative implementation on an ATSAM4S4A chip or similar.

References

[1] S. Andreev *et al.*, “Understanding the iot connectivity landscape: A contemporary m2m radio technology roadmap”, *IEEE Communications Magazine*, vol. 53, no. 9, pp. 32–40, 2015.

[2] Z. Sheng, C. Mahapatra, C. Zhu, and V. C. M. Leung, “Recent advances in industrial wireless sensor networks toward efficient management in iot”, *IEEE Access*, vol. 3, pp. 622–637, 2015. DOI: `10.1109/ACCESS.2015.2435000`.

[3] A. Ericsson, “Cellular networks for massive iot—enabling low power wide area applications”, *no. January*, pp. 1–13, 2016.

[4] J. G. *et al.*, “Internet of things (iot): A vision, architectural elements, and future directions”, *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[5] M. T. Anowar *et al.*, “A survey of acoustic underwater communications and ways of mitigating security challenges”, 2016.

[6] É. Gillet. (). Eurorack, Mutable Instruments SARL, [Online]. Available: `https://github.com/pichenettes/eurorack` (visited on 09/21/2019).

[7] D. Bubley, “Data over sound technology: Device-to-device communications and pairing without wireless radio networks”, Disruptive Analysis Ltd., Jun. 2017, p. 29.

[8] I. F. Akyildiz *et al.*, “Underwater acoustic sensor networks: Research challenges”, 2005.

[9] Gang Qiao, Songzuo Liu, Zongxin Sun, Feng Zhou, “Full-duplex, multi-user and parameter reconfigurable underwater acoustic communication modem”, 2013.

[10] Simon Haykin, *Communication systems*, 4th ed. John Wiley & Sons Inc., 2001.

[11] IBM Corporation and Microsoft Corporation, “Multimedia programming interface and data specifications 1.0”, 1991.

[12] Y. Rahmatallah and S. Mohan, “Peak-to-average power ratio reduction in ofdm systems: A survey and taxonomy”, *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1567–1592, 2013.

[13] Timothy M. Schmidl and Donald C. Cox, “Robust frequency and timing synchronization for ofdm”, 1997.

[14] David C. Chu, “Polyphase codes with good periodic correlation properties”, 1972.

[15] Maja Sliskovic, “Sampling frequency offset estimation and correction in ofdm systems”, 2001.

[16] Atmel, *Sam4s series*, Atmel Corporation, 1600 Technology Drive, San Jose, CA 95110 USA, Jun. 2015.

[17] ARM, *Arm® cortex®-m4 processor*, ARM ltd., 110 Fulbourn Road, Cambridge, England CB1 9NJ, Feb. 2015.

[18] T. Lorenser, “The dsp capabilities of arm® cortex®-m4 and cortex-m7 processors”, 2, Nov. 2016, p. 19.

[19] SEGGER, *Segger systemview user guide*, SEGGER Microcontroller GmbH, In den Weiden 11, D-40721 Hilden, Germany, Aug. 2018.

[20] I. S. Reed, G. Solomon, “Polynomial codes over certain finite fields”, 1960.

[21] K. Marneweck *et al.*, “Why data-over-sound is an integral part of any iot engineer’s toolbox: Chirp + arm = frictionless low power connectivity”, 2019.