POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering



Master's Degree Thesis

Wireless neural stimulation system development

Supervisors:
Ph.D. Paolo MOTTO ROS
M.Sc Stefano CONCADORO
M.Sc Letizia CANTORE

Candidate:
CHIARA VAGNI

September 2026

Abstract

The aim of this thesis is the development of a firmware that allows the acquisition of neural signals and nerve current stimulation at the same time within a bidirectional communication system connecting an implantable device and an external mechatronic structure, such as an exoprosthesis or exoskeleton. The nRF52840 DK board plays the role of the external system, sending commands and parameters through Bluetooth Low Energy (BLE) to the NUCLEO WBA65RI board connected via Serial Peripheral Interface(SPI) to the Intan RHS2116 headstage acting as a bidirectional electrophysiology interface system.

The first result obtained through this work is the possibility to calibrate the current amplitude at which the selected channel is held during stimulation separately for the positive and negative phase, to achieve the desired values as closely as possible. Under specific conditions a maximum error less than or equal to 0,05% of the required current value can be reached. Another result is the creation of a system that enables the user to select the desired biphasic stimulation waveform between the options provided and send the corresponding parameters via BLE, including the number of the channel on which the waveform will occur, and to simultaneously implement acquisition from another channel at the sample frequency of 10 kHz.

In conclusion, the outcome of this project is a firmware that provides reasonable user control of the main parameters characteristic of a stimulation waveform through bidirectional wireless communication, while maintaining simultaneous neural signals acquisition, and therefore offers a great groundwork in the development of bidirectional intelligent neural systems.

Table of Contents

List of Tables VI

List of Figures VII

Acronyms XI

1 Introduction 1
- 1.1 State of the art . . . 2
  - 1.1.1 Exoprostheses and Exoskeletons . . . 2
  - 1.1.2 Neural Interfaces . . . 3
  - 1.1.3 Neural Stimulation . . . 4
  - 1.1.4 Basics of SPI and USART . . . 6
  - 1.1.5 Basics of BLE . . . 7
  - 1.1.6 Relevant studies . . . 8
- 1.2 Aim of the study . . . 12
- 1.3 Requirements . . . 13

2 Methods 15
- 2.1 Setup . . . 15
  - 2.1.1 RHS2116 . . . 15
  - 2.1.2 LVDS Adapter Board . . . 16
  - 2.1.3 18-pin Electrode Adapter Board . . . 16
  - 2.1.4 STM32 NUCLEO BOARDS . . . 17
  - 2.1.5 Nordic nRF52840 DK . . . 17
- 2.2 Additional Hardwares . . . 18
- 2.3 Softwares . . . 19
  - 2.3.1 STM32CubeIDE . . . 19
  - 2.3.2 STBLEToolbox . . . 19
  - 2.3.3 MATLAB . . . 19
  - 2.3.4 SEGGER Embedded Studio . . . 20
- 2.4 Firmware . . . 20

2.4.1 RHD/RHS STM32 Firmware Framework 20

3 Development and Testing of Codes Implementation 21
3.1 Acquisition and stimulation through SPI and USART 21
3.1.1 *userconfig.h* 21
3.1.2 *main.c* 23
3.1.3 *userfunctions.c* 31
3.2 Introduction of BLE communication 39
3.2.1 *p2p_server_app.c* 40
3.2.2 *main.c* 46
3.3 BLE communication with nRF52840 DK 52
3.3.1 *p2p_server_app.c* 53
3.3.2 *main.c* 55
3.3.3 *userfunctions.c* 58
3.3.4 SEGGER Embedded Studio - *main.c* 59

4 Results 67
4.1 Acquisition 67
4.2 Calibration procedure 68
4.3 BLE Communication 76
4.4 Discussion of the results 78

5 Conclusion 79
5.1 Limitations 81
5.2 Possible enhancements 83

Bibliography 84

List of Tables

1.1 Bluetooth LE specifications 7

2.1 Pin mapping between RHD 16-channel headstage #C3334 or #C3335 and RHS 16-channel stim/record headstage #M4016 16

4.1 Values set in different trials for *first_phase_amplitude* (POS AMP) and *second_phase_amplitude* (NEG AMP) together with the corresponding voltage amplitude measured (POS PHASE MEAN and NEG PHASE MEAN) and the equivalent error (ERR POS and ERR NEG) with respect to the expected amplitude (TARGET). 69

List of Figures

1.1 Typical charge-balanced, current waveforms used in neural stimulation. The parameters vary widely depending on the application and size of the electrode [7]. 5
1.2 SPI Full Duplex communication between two devices. 6
1.3 USART: Asynchronous Mode 6
1.4 (A) Subject holding a cherry while removing the stem without sensory feedback. (B) Feedback enabled, subject successfully gripped the cherry and removed the stem without damaging the fruit. (C) Total force from thumb and index tip sensors when subject 1 had audiovisual feedback but not sensory feedback. Peak force during the trial is denoted with a red asterisk. (D) Total force from thumb and index tip sensors when subject 1 had both audiovisual feedback and sensory feedback.[11] 9
1.5 (A) Perceived magnitude and areas of referred sensations as a function of the injected charge in the channel of the TIME used in the tests. (B) Minimum charge needed to evoke a sensation (CMIN) and the maximum one that produced an intense but not painful sensation (CMAX), as a function of the test day.[12] 10
1.6 On the left is shown a schematic of the electrode module placed over the cortex and on the right the block diagram of the wireless neuronal recording system, comprising the electrode module, wireless system module, battery module, and receiver[13]. 11
1.7 Graphical user interface.[14] 12
1.8 General Diagram of the main NerveRepack project components[15]. In orange are highlighted the tasks met by this project. 13

2.1 NUCLEO-H723ZG [19]. 17
2.2 Digital oscilloscope Rigol MSO5104 [22]. 18
2.3 RIGOL DC Power Supply DP832A [23]. 18
2.4 HP Agilent 33120A Function generator [24]. 18

2.5 Acquisition and Stimulation Setup. In blue the LVDS Adapter Board, in red the RHS2116, in green the 18-pin Electrode Adapter Board, in purple the NUCLEO-WBA65RI, in orange the nRF52840 DK. . 19

3.1 Flowchart of *main.c* 29
3.2 Flowchart of *sample_processing_routine()* 30
3.3 Time parameters of TriphasicWaveform 34
3.4 Amplitude parameters of TriphasicWaveform 34
3.5 Flow chart of *correction_pos_phase()*. 38
3.6 Flow chart of *correction_neg_phase()*. 39
3.7 On the left is shown the interactive configuration menu developed in this work and on the right the application screen where the user enters the values. 44
3.8 Flow chart of *main()* routine in *main.c* file. 47
3.9 Flow chart of *run_zcheck_acquisition()*. 51
3.10 Flow chart of *perform_impedance_measurement()*. 52
3.11 Flow chart of *main()* routine in *main.c* file. 57
3.12 Structure of BLE packets sent by WBA65RI to nRF52840 DK. . . . 59

4.1 Sine wave. Function generator settings: 100 Hz, 2 Vpp. Expected amplitude: 9.95 mVpp. The two vertical lines attest the frequency is right. 67
4.2 Triangle wave. Function generator settings: 80 Hz, 1 Vpp. Expected amplitude: 4.975 mVpp. The two vertical lines attest the frequency is right. 68
4.3 Biphasic parameters set in the firmware and corresponding biphasic waveform samples acquired plotted on MATLAB. 69
4.4 Effect of enabling amplifier fast settle on the AC channel. 70
4.5 Effect of enabling amplifier fast settle on the DC channel. 70
4.6 Charge recovery enabled at the end of stimulation. 71
4.7 Reduction of error percentage depending on the trim value. 72
4.8 Calibration lines for positive and negative phase obtained from the correspondence between the magnitude values set in the firmware and the voltage amplitude measured. 72
4.9 The table on the top contains calibration parameters employing a resistance R = 1,0048 MΩ and *StimStepSize* = 50 nA. The table on the bottom contains calibration parameters employing R = 513 kΩ and *StimStepSize* = 100 nA. 73

4.10 On the top is plotted the stimulation waveform corresponding to a target of 3 V before the correction procedure, in the middle an intermediate stage, on the bottom is plotted the waveform after applying the correction. 75
4.11 BLE communication between NUCLEO WBA65RI and STBLEToolbox application. 76
4.12 BLE communication between NUCLEO WBA65RI and nRF52840 DK. 77

5.1 Summary diagram of the project 80
5.2 At the top, the positive (*peak_pos*) and negative (*peak_neg*) phase values recorded by the Intan DC amplifier are displayed. No correction is applied in the left plot, whereas the correction is applied in the right plot. At the bottom, the oscilloscope screen is reported first for the case without correction and then for the case with correction. 82

Acronyms

ADC
Analog to Digital Converter

ASCII
American Standard Code for Information Interchange

ASIC
Application Specific Integrated Circuit

BLE
Bluetooth® Low Energy

CNS
Central Nervous System

DAC
Digital to Analog Converter

DK
Development Kit

DLE
Data Length Extension

DMA
Direct Memory Access

DSP
Digital Signal Processing

GAP

Generic Access Profile

GATT

Generic Attribute Profile

GPIO

General-Purpose Input/Output

IDE

Integrated Development Environment

MCU

MicroController Unit

MISO

Master In Slave Out

MOSI

Master Out Slave In

MTU

Maximum Transfer Unit

NSS

Not Slave Select

PHY

Physical Layer

PNS

Peripheral Nervous System

SCK

Serial Clock

SPI

Serial Peripheral Interface

UART

Universal Asynchronous Receiver-Transmitter

USART

Universal Synchronous/Asynchronous Receiver-Transmitter

UUID

Universally Unique Identifier

Chapter 1

Introduction

In recent years, active implantable medical devices have gained an important role in treating and even curing many diseases. Among them, neural implants are increasingly attracting interest. These devices are widely used to stimulate specific areas of the brain or specific nervous pathways to reduce symptoms of various brain-based disorders such as Parkinson's, epilepsy, and peripheral neuropathy. Their employment is also being tested in the manipulation of the vagus nerve, which connects most of our key organs to the brain stem, and in the stimulation of the spinal cord to allow paralyzed people to move, stand, and even walk a short distance for the first time since sustaining spinal cord injuries. Besides all these applications, a very innovative and interesting area of research is the one of mind-controlled prostheses, through which the amputees can control robotic limbs using their thoughts. This can be achieved with a neural implant in the brain or in the extremity above the amputation. Another objective is to make these robotic limbs provide sensory feedback through nerve stimulation above the amputation site, allowing users to feel what they are touching. However, neuroprosthetic systems are still largely at the research stage and have not yet achieved widespread clinical adoption.

This thesis is part of NerveRepack, a project co-funded by the European Union, which includes Politecnico of Torino among its parters. This work involves a team of scientists, engineers, and medical professionals focusing on the development of a revolutionary implantable interfaces technology that enables bidirectional communication between the nervous system and and advanced prosthetic limbs or exoskeletons, to allow individuals affected by limb amputations and paralysis to regain both motor control and sensory feedback. The aim is to improve the quality of life of people with amputated or paralyzed limbs, not only by physical rehabilitation. The potential to feel sensations and control their movements again, indeed, can significantly improve their mental well-being and social interactions. To test the

integration between all fabricated module and components with exoprostheses and exoskeletons three demonstrators aimed at different categories of patients will be employed: with forearm amputation, with lower limbs paralysis and with single leg paralysis. This new generation of bidirectional neural systems will bring to technology advances including miniaturization, wireless communication and power supply, progresses in medical microsurgery tools and methods, new biocompatible materials and technologies. NerveRepack project is expected to reshape the approach to supporting people with disabilities with far-reaching impacts on society, economy, medicine, and technology.

The plan is to design and fabricate an implantable electronic module to ensure bidirectional communication between the Peripheral Nervous System (PNS) and the exoprosthesis or the exoskeleton, together with low energy consumption and efficient processing of the recording and stimulation electrodes signals. The electrodes will be implanted in median and ulnar nerves for exoprostheses and in the in the users' spinal cord for exoskeletons. Plug electrodes will mainly be used to record signals from specific motor fascicles. Based on them, the implanted neuronal module will transmit neuronal signals acting as command signals for exoprostheses or exoskeletons translated into movements. Moreover, it will receive signals from pressure sensors placed on exoprostheses and exoskeletons and translate them into tactile sensations for the user, stimulating specific sensorial fascicles mainly through cuff electrodes.

The description of the work carried out with this thesis is arranged in several sections. The first part above introduces the context from which this project was born and the issues to address which it was conceived. The following section examines the state of the art, citing the main studies that anticipated and brought to this project. Subsequently, the hardware setup and the softwares employed to test and debug the functioning and the performances of the developed firmware are described. After this, the key code frames and functions of this project are analyzed. The results obtained are presented in a dedicated section, followed by a discussion on the efficiency of the firmware. Finally, possible future developments are proposed.

1.1 State of the art

1.1.1 Exoprostheses and Exoskeletons

The upper limb prostheses available today can be ranked in the following main categories: passive, body-powered, myoelectric, neural and hybrid[1][2]. The first ones, also named "cosmetic", are the most basic and the lightest-weight devices, as

they do not allow active movement of the joints and incorporate few mechanical systems. Body-powered prosthetics are usually controlled by the remaining shoulder movements of the amputee, via a harness and a cable that is connected to the terminal device, elbow, or wrist. Myoelectric prostheses use electromyographic (EMG) signals from residual muscles to drive motor commands. These are the most widely available advanced prosthetic system, because electrical signals can be detected through the skin, and are consequently easier to record and decode. They offer, however, a limited number of command channels and require a contact between the electrodes and the skin that can be disrupted over time, causing signal inconsistency. Neural prostheses are controlled with signals acquired with electrodes implanted on or within the pheripheral nerves or the brain and they often use artificial electrical stimulation to replace nerve impulses emitted by the brain, to control muscle activity.
Exoskeletons are powered wearable robotic devices that resemble human limbs, with a one-to-one correspondence between human anatomical joints and the robot's joints[3]. They operate mechanically parallel to the human body to allow patients with gait motor disorders to stand up and move autonomously or to restore lost or weak functions, e.g. following a disease or a neurological condition, to their natural levels.

1.1.2 Neural Interfaces

A neural interface is a bidirectional transducer that establishes a contact between a technical device and a neural structure within the body to record bioelectrical signals from natural sensors of the body and to excite nerves and/or muscles. Myoelectric prostheses could be considered simple neural interfaces, since they take advantage of the electrical signals generated during muscle contractions, that are the result of movement commands generated in the motor cortex and propagated along peripheral nerves. Neural interfaces can be grouped into two main categories, depending on whether they address the central nervous system (CNS) or the peripheral nervous system (PNS)[4]. The first interfaces are referred to as brain-machine interfaces (BMIs) or brain-computer interfaces (BCIs) and are intended to manage neurological and psychiatric disorders through deep brain stimulation and target spinal cord neural pathways to treat chronic neuropathic pain, motor dysfunction, and sensory paralysis. Through PNS interfaces it is possible to restore lost sensory functions, such as vision and hearing. Moreover, by activating the motor regions, management of respiratory ventilation and urinary incontinence, correct foot drop, and control of prosthetic limbs with tactile feedback can be achieved.

Peripheral Nerve Interfaces

The PNS is composed of bundles of nerve fibers that extend from the brain and spinal cord to several sites to carry sensory (afferent) and motor(efferent) signals in a cable-like arrangement. Neural signals can be intercepted from peripheral motor nerve fibers by employing electrodes placed inside of or directly adjacent to nerve bundles. Moreover, stimulation of afferent nerve fibers can provide sensory and proprioceptive feedback. As regards direct nerve recording and stimulation, both extraneural and intraneural electrodes application has been investigated by researchers. Extraneural electrodes are located outside the nerve and provide simultaneous interface with many axons in the nerve, offering low selectivity for individual axons. Some of the most used are cuff electrodes, made of an insulating tubular sheath, that completely encircles the nerve and therefore records and stimulates a few to several combined nerve fascicles, depending on the number and arrangement of contacts exposed at their inner surface. Intraneural electrodes penetrate the nerve, accessing individual or small clusters of nerve fibers. Due to this, they have a much higher selectivity than the extraneural ones and require lower stimulus intensities for nerve stimulation.
Several neuroprostheses include interfacing the PNS or muscles most commonly by means of microelectrode devices that contact peripheral nerves or muscles using an electrical coupling method. Contemporary research in this field focuses on developing and experimentally testing PNS interfaces designed to prevent tissue and nerve damage. These interfaces aim to capture sensory afferent information, selectively stimulate multiple nerve fibers, and achieve graded control of muscle force. Since the electrodes are implanted adjacent, around or even within a peripheral nerve trunk or spinal root, the stimulus intensity required for activation is reduced compared with muscular or surface electrodes, decreasing the danger related to electrochemical processes and the power consumption. It is also possible to obtain selective stimulation of different fascicles of axons of the nerve. Additional studies are required on the potential nerve damage caused by implanted electrodes and about mechanical, chemical and electrical compatibility[5].

1.1.3 Neural Stimulation

The goal of electrical stimulation of excitable tissue is often the triggering of action potentials in axons, which requires the artificial depolarization of some portion of the axon membrane to threshold. In most neural applications, electrical stimulation is applied to the target tissue as a series of biphasic current pulses, where the first phase, is used to elicit the desired physiological effect and the second phase is used to reverse electrochemical processes occurring during the stimulating pulse [6]. The biphasic waveform is obtained through a combination of a cathodal and an anodal phase. A cathodal current is reducing at the stimulation electrode, with the

direction of electron flow being from the electrode to the tissue. Anodal indicates an oxidizing current with electron flow in the opposite direction. The charge delivered is the time integral of the current, therefore corresponds with the product between the pulse magnitude and width in the case of costant-current pulse[7].

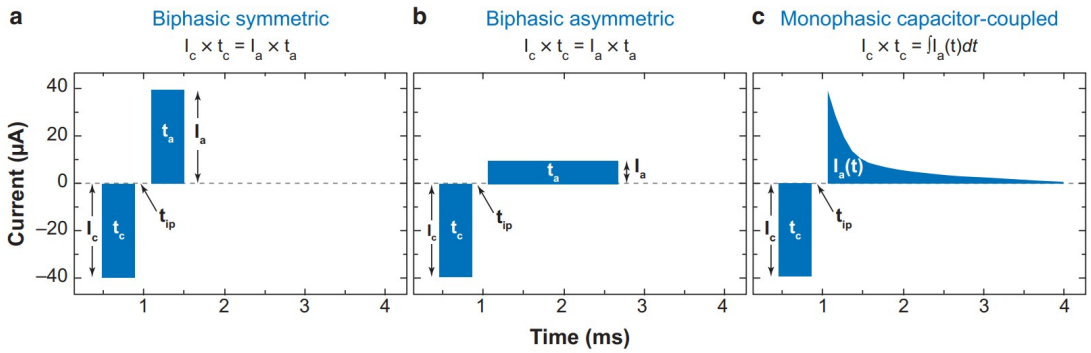


Figure 1.1: Typical charge-balanced, current waveforms used in neural stimulation. The parameters vary widely depending on the application and size of the electrode [7].

The charge-balanced (delivering a net charge of zero), cathodic first, biphasic stimulus can be considered the reference point, as it represented the most commonly used stimulus waveform in neuroprosthetic applications. The advantage of using a cathodal leading phase is that it requires less current to bring an axon to threshold compared with an anodic stimulus. The anodic phase first stimulus waveform allows strong selective activation of fibers of passage when an asymmetrical charge-balanced biphasic pulse is used, as demonstrated by McIntyre and Grill[8]. The use of charge-balanced asymmetric current waveforms has also been suggested by Cogan et al.[9] when using positively biased Ir oxide to maximize charge injection. The employment of a slightly charge imbalanced stimulation protocol has been proposed to compensate irreversible charge-injection processes, that can damage the tissue or the electrode. Monophasic pulsing causes the greatest shift of the electrode potential during pulsing away from the equilibrium potential, creating the most accumulation of unrecoverable charge of the three protocol types represented in Figure 1.1. Furthermore, there is accumulation of unrecoverable charge during the open circuit interpulse interval[6].

A critical aspect of electrical stimulation is the risk of both electrode and tissue damage. Electrical trauma to the nerves is especially related to pulse frequency and duration. Therefore, to minimize nerve damage, low stimulus frequencies with short durations, sufficient to elicit the desired clinical response, should be applied[10].

1.1.4 Basics of SPI and USART

The Serial Peripheral Interface (SPI) is a synchronous full-duplex communication protocol where the Master, typically a microcontroller, selects one or more Slaves, usually a chip, to send and receive information to and from. SPI is based on two data and two control lines: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock) and SS (Slave Select) /NSS (Slave Select, active low).



Figure 1.2: SPI Full Duplex communication between two devices.

The MOSI is the line through which the master sends data to the slave and the MISO is the line across which the slave sends data back to the master. The clock signal send by the SCK line, is generated by the master to send data or to allow slaves to shift data toward itself. The NSS line is used to select the slave with which the communication takes place. Since this protocol is synchronous it provides very high baudrates, between 20 and 50 Mbps.

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is hardware that enables a device to communicate using serial protocols. It can function in a slower asynchronous mode, like a universal asynchronous receiver/-transmitter (UART), or in a faster synchronous mode with a clock signal. A USART device can use both asynchronous and synchronous communication protocols. In the first case data are transmitted sequentially over separate TX and RX lines for the two communication directions, without a synchronization signal and each device has its own local clock. Therefore, both the sender and receiver must be manually configured beforehand to use the same data rate. Extra bits are, however, added to each packet of data to ensure reliable transmission, since it is possible for the devices to become slightly out of synchronization: a start and stop bit to indicate each packet and a parity bit to indicate small errors.

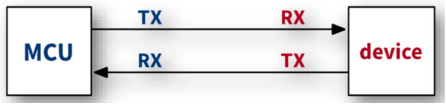


Figure 1.3: USART: Asynchronous Mode

In synchronous mode, a data and a clock line are used, so that the controller and the peripheral are synchronized at the same data rate, without the need to be

configured before and to use extra bits. It also avoids pauses when sending and enables much faster bit rates.

1.1.5 Basics of BLE

Bluetooth is a widely used wireless communication technology standard managed by the Bluetooth Special Interest Group (SIG). Bluetooth Classic is the standard radio protocol used in common items, such as wireless speakers, headphones, and in-car entertainment systems, to enable wireless audio streaming. It provides a data throughput high enough to support communication without connection problems or packet loss. In this context the need to recharge battery after some time is not an issue, but for devices that need to operate on minimal power and only send small bursts of data, like low-power wearables or massive IoT applications, frequent battery charging is not feasible.

To address this question, from Bluetooth version 4.0 the Bluetooth SIG introduced Bluetooth Low Energy (LE), designed to transmit smaller amounts of data to achieve low energy consumption by sacrificing data rate. BLE offers smaller data packets,ranging from 27 to 251 bytes, and is focused on avoiding long radio-on times, that greatly affect power consumption. In Table 1.1 are presented some of the main features of BLE.

Table 1.1: Bluetooth LE specifications

Parameter	**Value**
Operating band	2400 MHz – 2483.5 MHz (∼ 2.4 GHz)
Channel bandwidth	2 MHz
Number of RF channels	40
Maximum transmit power	20 dBm (0.1 W)
Maximum application data throughput	1.4 Mbps
Maximum range at reduced data rates (125 & 500 kbps)	∼1000 m

The GAP (Generic Access Profile) determines specific roles for nodes in a BLE network and how they can interact with each other. To establish a connection between two devices, one has to act as a peripheral that is currently advertising by sending packets to announce its presence, and one as a central that is currently scanning. The Service Universally Unique Identifier (UUID) is a number universally unique to a specific service. It can help scanners identify devices that are

interesting to connect to. If the central sends a connection request, after scanning an advertisement packet, a bi-directional connection channel is created.

The Attribute protocol (ATT) layer, and the Generic Attribute Profile (GATT) layer right above it, define how data is represented and exchanged. The ATT is based on a client-server architecture where the server, in most cases the peripheral, holds the data and can either send it directly to the client or the client can poll the data from the server. The data transferred are stored in the form of data structures called attributes, which also hold metadata describing the attribute itself, its type, security permissions, etc. Multiple attributes arranged in a specified format make up a service that can in turn have zero or more characteristics. A characteristic consists of at least two attributes and optionally more.

An important parameter is the connection interval that corresponds to how often the two devices wake up to talk, since they spend most of the time sleeping to avoid power consumption. The connection event occurs every connection interval when the central sends a packet to the peripheral. Other relevant connection parameters are the data length and MTU (Maximum Transfer Unit). The MTU is the number of bytes that can be sent in one GATT operation (for example, a send operation), while the data length is the number of bytes that can be sent in one Bluetooth LE packet, which can be increased from the default 27 bytes to up to 251 bytes thanks to Data Length Extension (DLE).

At the physical layer (PHY), BLE supports different modulation and transmission modes, including the 1M and 2M PHYs. The 2M PHY provides a nominal physical-layer data rate of 2 Mbit/s, compared with 1 Mbit/s for the 1M PHY.

1.1.6 Relevant studies

The term neuroprosthesis appeared for the first time in scientific literature in 1971 to describe an intraspinal implant that allowed bladder voiding after paraplegia realized by Nashold et *al.*. The very first neuroprosthesis, however, based on functional electrical stimulation (FES) was a peroneal nerve stimulator invented in 1961 by Liberson and colleagues to treat foot drop after hemiplegia. This field was also extended to technologies that extract motor commands from brain signals to control external devices.

Electrical neurostimulation can be delivered to a motor nerve innervating a targeted muscle to activate the efferent axons and produce a desired movement, but it can also be employed to elicit somatic sensations in individuals with limb amputation or paralysis. One of the advantages of somatosensory feedback is the possibility for the user to sense the amount of pressure exerted by the prosthesis, a crucial factor when handling fragile object.

In 2014 Tan et *al.*[11] employed electrical stimulation through peripheral nerve cuff electrodes implanted on median, radial and ulnar nerves of two subjects.

Beside demonstrating that peripheral nerve cuff electrode interfaces provided more than a year of multi-location, multiperception sensory feedback in human subjects, Tan and colleagues also shown that sensory feedback improves functional performance and user confidence. In both of the subjects, percepts were produced at multiple, independent, small, and well-defined locations on the hand, including the thumb and fingertips. The sensory feedback for the thumb and index finger corresponded to the measured pressure at equivalent positions on the prosthesis. With trains of identical, charge-balanced, square electrical pulses with stimulation frequency ranging from 1 to 1000 Hz, the subjects reported an unnatural sensation of paresthesia, while stimulation intensity modulation with a time-variant pulse width resulted in "natural" pressure perception. Moreover, changing average stimulation intensity controlled the size of the percept area, while changing stimulation frequency controlled sensation strength. As shown in Fig. 1.4, without sensory feedback the subject was often unable to adequately control the grip force in a delicate task such as holding a cherry while removing the stem (A), while he felt contact with the cherry and the force applied and completed the task successfully with the sensory feedback enabled (B).

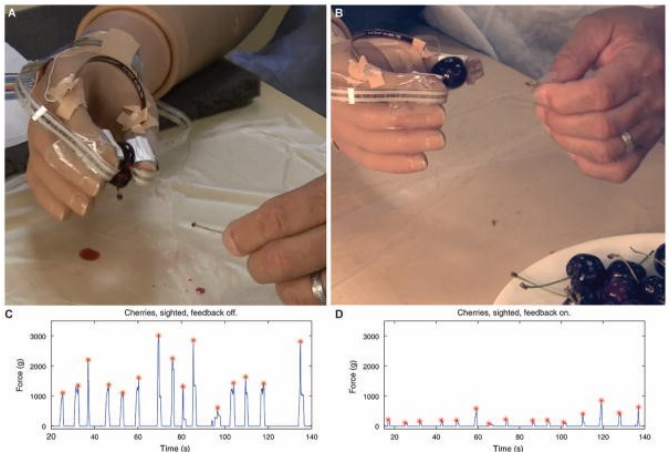


Figure 1.4: (A) Subject holding a cherry while removing the stem without sensory feedback. (B) Feedback enabled, subject successfully gripped the cherry and removed the stem without damaging the fruit. (C) Total force from thumb and index tip sensors when subject 1 had audiovisual feedback but not sensory feedback. Peak force during the trial is denoted with a red asterisk. (D) Total force from thumb and index tip sensors when subject 1 had both audiovisual feedback and sensory feedback.[11]

The outcomes of Tan et *al.*'s work are supported and extended by the study of Clemente et *al.* [12], by showing that intraneural sensory feedback of the grip force (GF) allows for learning and improves motor coordination. The subject

was a transradial amputee controlling a myoelectric prosthesis. During a stacking cups test (CUP) she was trained to use the sensory feedback and was then able to generalize and transfer this skill to an alien task, which was a pick and lift test (PLT). Sensory stimulation proportional to the GF produced by the hand prosthesis was provided, through one channel of a TIME (transverse intrafascicular multichannel electrode) surgically implanted in the ulnar nerve. The GF was measured by a sensor in the artificial middle finger and the stimulation elicited a vibratory sensation referred to the little finger of the phantom hand. The neural stimulator delivered biphasic, charge-balanced, cathodic-first, rectangular pulses at a frequency of 50 Hz. The pulse width was fixed (80 µs), while the amplitude was linearly mapped to the GF and updated in real time. In particular, the injected charge varied between CMIN and CMAX, experimentally assessed before each session and corresponding to the just perceivable sensation and to the charge that produced a comfortable but not painful one. The intraneural stimulation elicited a stable sensation of vibration referred to the little finger for the duration of this study, as assessed before each experimental session. However, the amount of CMIN and CMAX, slightly increased with time. Moreover, the area of the perceived sensation increased with increasing charge, although it never extended beyond the palmar region of the little finger. During the CUP, subject performed better in the condition with sensory feedback than in the one without it, in terms of both percentage of singularly transferred cups and time needed to complete the task. In the PLT, the subject showed better motor coordination in the first condition.

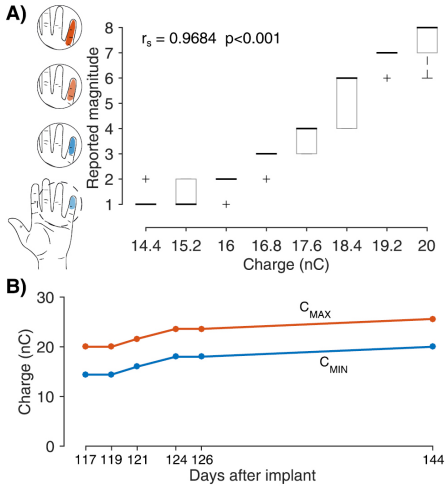


Figure 1.5: (A) Perceived magnitude and areas of referred sensations as a function of the injected charge in the channel of the TIME used in the tests. (B) Minimum charge needed to evoke a sensation (CMIN) and the maximum one that produced an intense but not painful sensation (CMAX), as a function of the test day.[12]

Recently, implantable wireless devices are spreading to overcome the drawbacks of using percutaneous cables between electrodes and external devices in neuroprostheses. Beside restricting the subject's mobility, indeed, they also involves a severe risk of infection penetrating the skin. Additionally, the neural signal quality can be affected by external noise due to subject's movements and nearby power lines. Two important aspects that need to be achieved in wireless devices are a wide range for data communication and low power consumption.

In 2021 Idogawa et *al.* built a BLE-based wireless neuronal recording system connected to a single-channel microneedle electrode and evaluated it on mice[13]. The block diagram of the system shown in Fig.1.6 consists of a Bluetooth transmitter module connected to an amplifier circuit module to measure neuronal activity (<500 μV). The acquired signals are amplified to a voltage level that can be detected using an analog–digital converter (ADC). The BLE receiver module comprises the Nordic nRF52840 DK and a serial-to-USB converter to send the data to a personal computer where they are processed. The performance of the system was tested acquiring neuronal activities for less than 1.5 s, at a sampling rate of 10 kHz, with no significant differences in the recorded data compared to conventional wired systems and a reduction of the external noise. The employment of BLE enabled bidirectional wireless communication up to 5 m from the subject and a power consumption of 28.6 mW, also favored by the use of a 3.7 V lithium battery (weighing 0.85 g) to supply power to the neuronal recording system, and of a Texas Instruments power regulation circuit module providing a supply of ±3 V, one for the analog circuit and the other for the digital one. BLE technology also enables data acquisition using inexpensive devices, such as a PC and a smartphone, instead of conventional large, expensive recording systems.

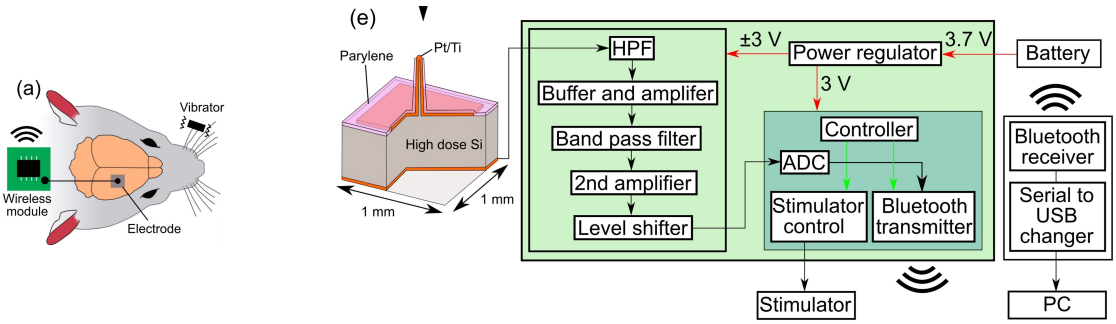


Figure 1.6: On the left is shown a schematic of the electrode module placed over the cortex and on the right the block diagram of the wireless neuronal recording system, comprising the electrode module, wireless system module, battery module, and receiver[13].

A relevant example of BLE implementation to control stimulation remotely and in real-time is the neurostimulation platform realized by Jiang et *al.*[14], designed to be implanted in large animal models for applications involving peripheral nerves. Their system features a neural implant consisting of three parallel stimulators integrated onto three application specific integrated circuits (ASICs) using a high voltage 0.6 μm CMOS technology and a wearable transmitter equipped with a BLE radio link, together with a custom graphical user interface (GUI). A custom BLE dongle was designed and fabricated for the radio link of the host computer. The GUI, shown in Fig. 1.7is developed in Matlab R2020a and controls the Bluetooth connection, implant connection, and stimulation parameter settings. The parameters set are sent in packets via the Bluetooth link to the transmitter, which repackages the data into frames and forwards them to the implant over the inductive link. Jiang and his colleagues demonstrated the capability of the stimulator to provide variable stimulation on multiple channel under remote control, allowing simultaneous current pulses on different channels with minimal channel interference due to crosstalk. They also pointed out the need of enhancements from the point of view of power consumption, physical size and communication latency of their model.

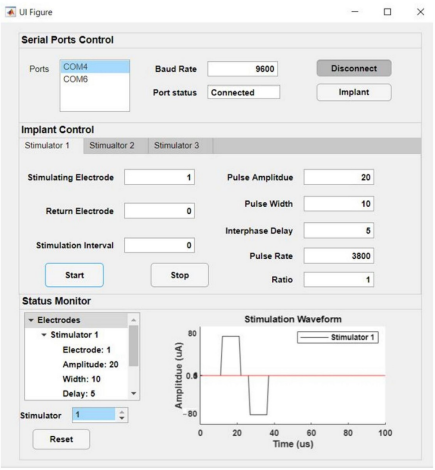


Figure 1.7: Graphical user interface.[14]

1.2 Aim of the study

This thesis involves the Intan RHS2116 microchip, a bidirectional electrophysiology interface system that allows both neural recording and stimulation. The microchip is connected to a test resistance through an headstage, recording signals from 16 low-noise amplifier channels, and delivering stimulation pulses on the same channels.

The chip is controlled by the NUCLEO WBA65RI that simulates the implantable module. The aim of this work is to develop a firmware to execute on the STM32 board to accomplish two main tasks. The first is to ensure the generation of a stimulation waveform as a feedback signal most possible in compliance with the parameters provided by the user through an appropriate interface. The second objective is to enable bidirectional wireless communication via BLE between the Nordic nRF52480 DK that acts as the external system and the NUCLEO WBA65RI for the transmission of stimulation parameters, acquired samples and eventual feedback notifications.

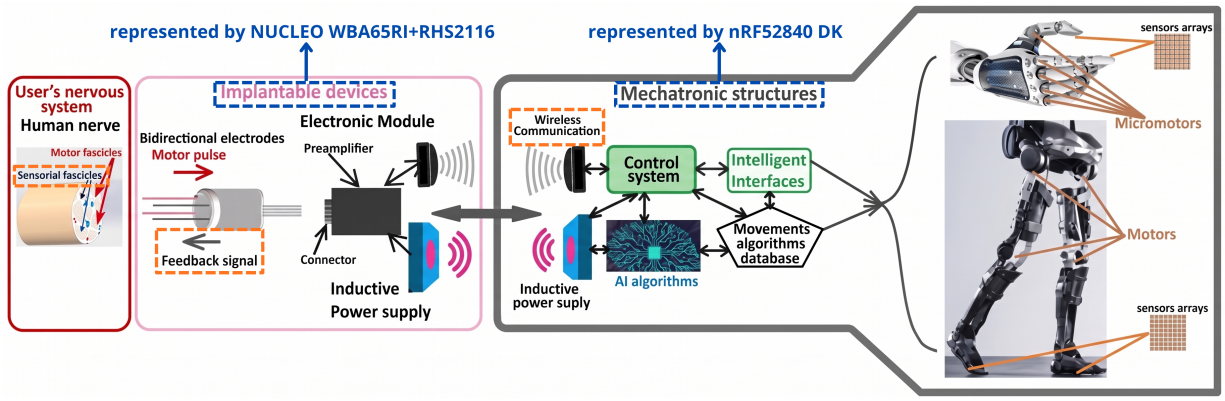


Figure 1.8: General Diagram of the main NerveRepack project components[15]. In orange are highlighted the tasks met by this project.

Providing a neural feedback to the amputee or paralyzed person will aid him/her in obtaining a great control of the exoprothesis or exoskeleton and in regaining several movements skills characteristic of an healthy individual. The accuracy of stimulation parameters is important not only to obtain the desired feedback, but also to avoid both electrode and tissue damage.

Moreover, the introduction of BLE communication will reduce the latency in data transmission and receiving times and will get rid of the issues related to the employment of percutaneous cables, first among all possible infections and device bulk, collaborating to the miniaturization of the system.

1.3 Requirements

In this section are declared the main requirements established for the NerveRepack project and the signal characteristics to expect. The amplitude of the signal acquired from the motor nervous system will be around 5 mVpp, while the minimum amplitude of the signal delivered from the ASIC, which will be created for the implanted electronic module, to the microcontroller should be 3,3V. The minimum

amplitude required for signals sent to the sensorial nervous fascicles is 100-200 μVpp.

The ASIC will have five recording channels and five stimulation channels and the frequency of the received and generated signals will not exceed 1000 Hz. The implantable module will have to deliver to the external device radio frequency signals in a distance range of minimum 10 cm.

Chapter 2

Methods

2.1 Setup

2.1.1 RHS2116

The RHS2116 microchip [16] is a device developed by Intan Technologies as a complete bidirectional electrophysiology interface system. The device is suitable for a wide variety of biopotential interfacing applications, thanks to the possibility of connecting an array of electrodes directly to one side of the chip and using a digital bus on the other side to control stimulation and read digitized electrode signals. The RHS2116 contains an array of 16 stimulator/amplifier blocks controlled through a digital SPI interface. Each channel includes a constant-current stimulator with programmable amplitude and two parallel sensing amplifiers: an AC-coupled high-gain amplifier with dynamically programmable bandwidth for observing small electrophysiological signals (ranging from μV to mV), and a DC-coupled low-gain amplifier for monitoring larger electrode potentials in response to stimulation. The upper cutoff frequency of the AC amplifiers can be adjusted from 100 Hz to 20 kHz and the lower cutoff frequency from 0.1 Hz to 1 kHz. The stimulators source and sink currents range from 10 nA to 2.55 mA over a 14 V range. Each current source and sink is controlled by an 8-bit current-output Digital to Analog Converter (DAC) which has an 8-bit "trim" setting that can adjust the current by $\pm 28\%$ to compensate for variations between devices. All the amplifiers share one on-chip ADC, through a low-distortion, high-speed analog multiplexer (MUX). The ADC can sample each channel up to 44.6 kSamples/s. Moreover, integrated charge-recovery circuits to avoid oxidation-reduction reactions at the electrode-tissue interface and fast amplifier settling to minimize residual artifacts can be employed following stimulation pulses and additional on-chip circuitry enables in situ electrode impedance measurements at user-programmable frequencies. Thanks to all these features, the RHS2116 replaces all analog instrumentation circuitry in

electrophysiology monitoring and stimulation systems.

2.1.2 LVDS Adapter Board

The Intan LVDS Adapter Board [17] is a hardware development tool used to interface RHS headstage to commercial microcontrollers. It translates standard Transistor-Transistor Logic or CMOS-level digital signals to low-voltage differential signals (LVDS) used by Intan headstages. These signals are essential for reliable high-speed digital signaling over long SPI interface cables. The board provides two independent units for the Intan RHS and RHD series chip, electrically isolated from one other. Each one has an Omnetics connector to interface with an Intan headstage or SPI interface cable. The +3.3 V must be provided to power the Texas Instruments SN65LVDT chips, responsible for the translation, and the Intan headstages. The RHS stim/record headstages require ±7V supplies, not ±9V supplies as printed on the board, to avoid permanent hardware failures.

2.1.3 18-pin Electrode Adapter Board

The 18-pin electrode adapter board [18] brings out all the electrode contacts, reference and ground from a headstage to solder holes. The numbers printed on the board correspond to channel numbers on the RHD 16-channel unipolar-input headstage. Table 2.1 provides the corresponding channel numbers for RHS.

RHD 16-channel headstage (as labeled)	**RHS 16-channel stim/record headstage**	**RHD 16-channel headstage (as labeled)**	**RHS 16-channel stim/record headstage**
8	0	17	9
9	1	18	10
10	2	19	11
11	3	20	12
12	4	21	13
13	5	22	14
14	6	23	15
15	7	REF	REF
16	8	GND	GND

Table 2.1: Pin mapping between RHD 16-channel headstage #C3334 or #C3335 and RHS 16-channel stim/record headstage #M4016

2.1.4 STM32 NUCLEO BOARDS

The NUCLEO-H723ZG [19], represented in Figure **??**, is an STM32 Nucleo-144 development board that incorporates an STM32H723ZG MCU. It relies on an Arm® Cortex®-M7 32-bit RISC core operating up to 550 MHz, equipped with a floating-point unit (FPU) that handles Arm® double-precision and single-precision data processing instructions and data types, combined with a full set of DSP instructions. A memory protection unit (MPU) increases the security of the application. The device provides high-speed embedded memories, featuring error code correction, with up to 1 Mbyte of flash memory and 564 Kbytes of RAM. The board supports various peripherals including I^2C, USART and SPI. Thanks to its internal or external SMPS (Switch Mode Power Supply), the NUCLEO-144 board allows to significantly reduce power consumption in Run mode.

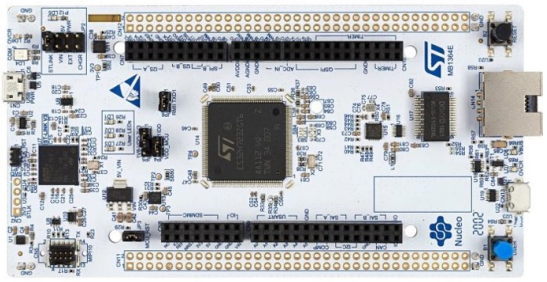


Figure 2.1: NUCLEO-H723ZG [19].

The NUCLEO-WBA65RI [20] is an STM32 Nucleo-64 wireless and ultra-low-power development board characterized by an STM32WBA65RI MCU. The microcontroller integrates a high-performance, ultra-low-power 2.4 GHz radio, compliant with Bluetooth® Low Energy (BLE) and IEEE 802.15.4-2015 PHY and MAC, enabling its employment in networks based on the Thread, Matter and Zigbee® protocols.

2.1.5 Nordic nRF52840 DK

The nRF52840 DK [21] is a Nordic single-board development kit for BLE, Bluetooth mesh, Thread, Zigbee, ANT, 802.15.4 and 2.4 GHz proprietary applications using the nRF52840 multi-protocol SoC. It incorporates an Arm® CortexTM-M4 with floating point support and is engineered to take advantage of BLE's major feature advancements and benefits from enhanced performance capabilities including long range and high throughput modes.

2.2 Additional Hardwares

The RIGOL MSO5104 oscilloscope was used to check the stimulation waves generated by the RHS2116, operating the probes in 10x.



Figure 2.2: Digital oscilloscope Rigol MSO5104 [22].

A dual voltage was provided to the LVDS Adapter Board by the RIGOL DC Power Supply DP832A. The VSTIM+ was connected to 6.5 V and the VSTIM- to -6.5V.



Figure 2.3: RIGOL DC Power Supply DP832A [23].

The HP Agilent 33120A Function generator was used to generate sine waves and triphasics in order to test their correct acquisition from the RHS2116 chip.

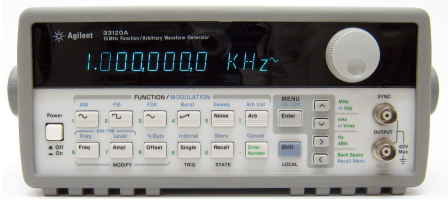


Figure 2.4: HP Agilent 33120A Function generator [24].

In Fig. 2.5 the final setup of signal acquisition and stimulation is represented.

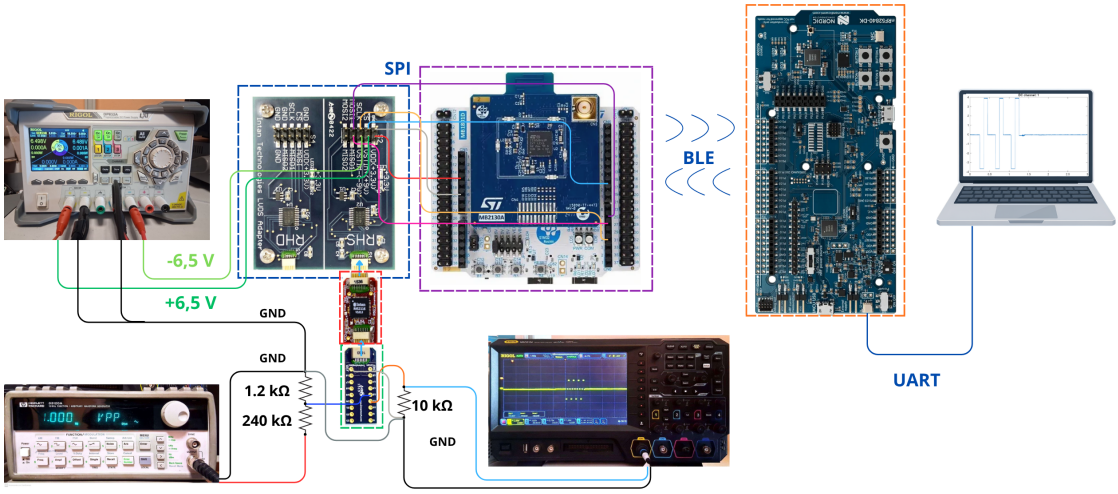


Figure 2.5: Acquisition and Stimulation Setup. In blue the LVDS Adapter Board, in red the RHS2116, in green the 18-pin Electrode Adapter Board, in purple the NUCLEO-WBA65RI, in orange the nRF52840 DK.

2.3 Softwares

2.3.1 STM32CubeIDE

STM32CubeIDE [25] is a multi-OS, C/C++ integrated development environment (IDE), based on Eclipse®, to edit, compile and debug STM32 code. It was used togheter with the graphical configuration tool STM32CubeMX. The enviroment supports STM32 products based on the Arm® Cortex® processor. For this project the 1.18.1 version and the 2.1.1 version are used.

2.3.2 STBLEToolbox

The STBLEToolbox [26] is an app available for Android and iOS. It scans for BLE devices and allows to debug each device connected, showing all the peripheral services and characteristics. The STBLEToolbox enables interaction for reading, writing, and notifications with any Bluetooth characteristic.
FIGURA DI ESEMPIO

2.3.3 MATLAB

The MATLAB programming and numeric computation enviroment [27] was used to receive the acquired samples via USART, and to process and graph them to

check the stimulation wave form obtained. It was also used as an interface to communicate with the user, printing directions in the command window.

2.3.4 SEGGER Embedded Studio

SEGGER Embedded Studio (SES) [28] is a streamlined IDE employed for writing, compiling, and debugging code on the nRF52840 DK board. For this project the 5.42a version of SES is used.

2.4 Firmware

2.4.1 RHD/RHS STM32 Firmware Framework

The open-source STM32 firmware framework provided by Intan Technologies is taken as the starting point to implement this project [29]. It includes C code written for the commercially-available STM32U5 and STM32H7 microcontroller series produced by STMicroelectronics to facilitate the development of electrophysiology recording systems using the RHD or RHS series of microchips. For this thesis, the example code "H7 rhs_acquisition" is considered. It is developed to run acquisition via SPI with an RHS2116, transmit acquired data, either in real time or offline, via USART with an STM32H7 chip, and to trigger stimulation by pushing the blue user button on the NUCLEO board.

Chapter 3

Development and Testing of Codes Implementation

This chapter provides a detailed explanation of the code written to implement this project. The sections follow the step carried out during the work. DA MIGLIORARE

3.1 Acquisition and stimulation through SPI and USART

The code developed in STM32CubeIDE in the first step of the project runs on the NUCLEO-H723ZG and is implemented starting from the RHD/RHS STM32 Firmware Framework mentioned in section 2.4. It includes data acquisition from RHS2116 via SPI, stimulation and calibration of the stimulation waveform. The sample frequency used in this phase is 20 kHz. To test the correct functioning of the firmware the acquisition and stimulation setup represented respectively in figure **??** and AGGIUNGI FIGURA were employed. The stimulation was initially triggered by pushing the blue user button within the acquisition period, and was later triggered automatically.

3.1.1 *userconfig.h*

In the file *userconfig.h* the user can decide to employ real-time acquisition or to define the macro OFFLINE_TRANSFER in order to acquire a short period of data and then transmit them offline. In this step the second option is chosen and the samples are sent via USART to the PC and processed through MATLAB. A led to detect when an error occurs can be set in this file too. In this project, the

yellow led, user LD2, is chosen, as the pin connected to the red led, user LD3, is used for SPI communication.

```
// If acquiring a short period of data, then exiting and transmitting data offline is desired, leave this uncommented.
// If instead transmitting data in real-time is desired, leave this commented.
#define OFFLINE_TRANSFER

// Error detect GPIO
#define ERROR_DETECTED_PORT         LED_YELLOW_GPIO_Port
#define ERROR_DETECTED_PIN          LED_YELLOW_Pin
```

By default, for the RHS chip after 16 CONVERT commands in each sample period, there are four auxiliary command slots. Each command is a 32-bit SPI word. If the macro AUTO_STIM_CMD_MODE is defined these slots are used for on-the-fly WRITE commands to stimulation control registers during acquisition so that pre-defined stimulation sequences can be executed, as reported in subsection 3.1.2. Otherwise, the contents of these auxiliary commands can be customized in *configure_aux_commands()* in *userfunctions.c*, but if left unchanged, this will have command 1 cycle through all registers, re-writing each according to software-configured values, and the remaining commands simply repeat dummy READ commands on ROM registers.

If AUTO_STIM_CMD_MODE is defined, then a rising-edge signal on a specific General-Purpose Input/Output (GPIO) pin (mapped to the blue user button so that pressing this button acts as a rising-edge) during acquisition will trigger user-specified stimulation pulses on 2 channels.

```
// How many CONVERT commands are sent in a single sequence,
// which occurs every time the period defined by INTERRUPT_TIM occurs (default 20 kHz).
// Default of 16 indicates that 16 amplifier channels are each sampled once per sequence.
#define CONVERT_COMMANDS_PER_SEQUENCE 16
#define AUX_OFFSET CONVERT_COMMANDS_PER_SEQUENCE

// How many AUX commands are sent in a single sequence
// Default of 4 indicates that 4 auxiliary command lists
// each execute a single command per sequence.
#define AUX_COMMANDS_PER_SEQUENCE 4

// How many AUX commands are contained in a single auxiliary
// command list (excluding zcheck_DAC command lists).
// Default of 128 indicates that for each sequence, each of the
// AUX_COMMANDS_PER_SEQUENCE (default 4) command lists executes a single command, so that 128 sequences must occur before an auxiliary command list finishes and repeats execution from the beginning again.
```

```
#define AUX_COMMAND_LIST_LENGTH 128

#define AUTO_STIM_CMD_MODE

#define NUM_STIM_SEQUENCES 2
```

Key parameters of acquisition are then set. Table 2.1 must be taken into account to know what is the number to assign to FIRST_SAMPLED_CHANNEL corresponding to the 18-pin Electrode Adapter Board channel desired.

```
// When recording offline, how many seconds of data should be acquired before the acquisition loop is escaped. With the default sample rate of 20000, 1 second of data corresponds to 20000 samples per channel.
// Note that on-chip RAM is limited, so setting this number excessively high will cause the chip to run out of memory during program execution.
#define NUMBER_OF_SECONDS_TO_ACQUIRE 3

// Which of the RHS chip's amplifier channels is selected as the starting point to have its data saved and transmitted via USART.
#define FIRST_SAMPLED_CHANNEL 3

// How many channels (starting from FIRST_SAMPLED_CHANNEL) to have their data saved and transmitted via USART.
#define NUM_SAMPLED_CHANNELS 1
```

Several tests were conducted changing the sampled channels and the NUMBER_OF_SECONDS_TO_ACQUIRE. It is not possible with the sample frequency used to acquire more than 3 seconds, if transmission offline is employed.

3.1.2 *main.c*

main.c contains the basic framework of the code. First of all necessary variables are defined. The variable *amp_parameters*, of type *Parameters_t*, is placed in the *.noinit* memory section, which is not cleared or initialized during system startup.

```
    /* USER CODE BEGIN PV */
volatile uint32_t counter_stim = 0;
volatile bool trigger_flag = 0;
volatile bool acquisition_finished=0;

extern bool change_voltage_value;
extern float peak_pos;
extern float peak_neg;

// place this variable into a specific memory section named .noinit
```

```
__attribute__((section(".noinit"))) Parameters_t amp_parameters;

/* USER CODE END PV */
```

The *Parameters_t* structure is defined in *userfunctions.h* as follows:

```
// userfunctions.h file

typedef struct {
     int16_t trim_pos;
     int16_t trim_neg;
     int16_t mag_pos;
     int16_t mag_neg;
     uint16_t trim_counter_pos;
     uint16_t trim_counter_neg;
     float amplitude_desired;
     bool automatic_stim;
} Parameters_t;
```

Two functions are defined to correctly initialize stimulation waveform parameters for the calibration.

The *init_value()* function initializes *amp_parameters* fields to 0 exclusively if the system reset was triggered by a Power-On Reset (POR) or a Brown-Out Reset (BOR).

```
    void init_value(void)
{
    uint32_t por_reset = __HAL_RCC_GET_FLAG(RCC_FLAG_PORRST);
    uint32_t bor_reset = __HAL_RCC_GET_FLAG(RCC_FLAG_BORRST);

    if (por_reset || bor_reset)
    {
        amp_parameters.trim_pos = 0;
        amp_parameters.trim_neg = 0;

        amp_parameters.mag_pos = 0;
        amp_parameters.mag_neg = 0;

        amp_parameters.trim_counter_pos = 0;
        amp_parameters.trim_counter_neg = 0;

        amp_parameters.automatic_stim = 0;

        amp_parameters.amplitude_desired = 0;
    }

```

```
22      __HAL_RCC_CLEAR_RESET_FLAGS();
23  }
```

The *check_trim()* function defines the desired voltage amplitude, checks if the amplitude and trim value are within the range accepted by the firmware and when the code is executed for the first time (*amp_parameters.mag_pos* and *amp_parameters.mag_neg* are equal to 0) calculates the initial value to assign to *amp_parameters.mag_pos* and *amp_parameters.mag_neg*. The equations used to compute these values were obtained as described below in section 4.2 of Chapter 4.

```
1  void check_trim(void){
2      if(amp_parameters.amplitude_desired < 1 || amp_parameters.amplitude_desired >
       5){
3              amp_parameters.amplitude_desired = 2;
4          }
5
6      TARGET_V_DC = amp_parameters.amplitude_desired;
7
8      if (amp_parameters.trim_pos < -128 || amp_parameters.trim_pos > 127) {
9          // trim out of range
10         amp_parameters.trim_pos = 0;
11         amp_parameters.trim_counter_pos = 0;
12     }
13
14     if (amp_parameters.trim_neg < -128 || amp_parameters.trim_neg > 127) {
15             // trim out of range
16             amp_parameters.trim_neg = 0;
17             amp_parameters.trim_counter_neg = 0;
18     }
19     if (amp_parameters.mag_pos== 0 || amp_parameters.mag_pos < -255 ||
       amp_parameters.mag_pos > 255) {
20         amp_parameters.mag_pos = (int)roundf((TARGET_V_DC + 0.017f)/0.0423f);
21     }
22     if (amp_parameters.mag_neg == 0 || amp_parameters.mag_neg < -255 ||
       amp_parameters.mag_neg > 255) {
23         amp_parameters.mag_neg = -(int)roundf((TARGET_V_DC - 0.0002f)/0.0384f);
24     }
25 }
```

At the beginning of the *main()* function, the two functions explained above are executed together with the configuration and initialization of MCU peripherals, including SPI and Direct Memory Access (DMA), through auto-generated functions based on the .ioc file. An array named *sample_memory* is allocated to store acquired data and the RHS registers are initialized with values written to the chip via SPI.

Subsequently, a list of 16 CONVERT commands and a list of 4 auxiliary commands are created and stored in memory for later use.

```
    configure_convert_commands();
#ifdef AUTO_STIM_CMD_MODE
  configure_stim_sequences();
  process_stim_sequences();
#else
  configure_aux_commands();
#endif

  copy_next_aux_commands_to_MOSI();
```

The green led, user LD1, is then turned on to announce the beginning of acquisition and the timer peripheral configured to trigger at 20 kHz is started. After this the data acquisition loop begins and only breaks when the *loop_escape()* function returns 1, which will not occur until a number of seconds of data equal to NUMBER_OF_SECONDS_TO_ACQUIRE have been acquired. The dedicated pin *Main_Monitor_Pin* is continuously written high to check when the loop is processing. Every sample period, *sample_interrupt_occured* is set to true and the *sample_processing_routine* is executed.
This function sends 20 commands (16 CONVERT commands and 4 auxiliary commands). The SPI transfer uses DMA to iterate through the full command list and this routine only begins the transfer. The routine also checks if the previous SPI sequence is complete through the variable *command_transfer_state*: if the sample period is shorter than the time required for each SPI sequence and the next sequence is triggered before the previous one finishes, *SampleClip* error occurs. If a SPI transfer sequence correctly ends an interrupt triggers a specific function and the results of CONVERT commands are written to memory. *command_transfer_state* is set to TRANSFER_WAIT once the transfer begins, TRANSFER_COMPLETE once the transfer finishes, and TRANSFER_ERROR if some error is detected.

```
    // Keep looping, doing nothing (other than handling interrupts) until enough
    data has been gathered
  while (1) {

    // Break infinite loop when loop_escape() condition has been met.
    if (loop_escape()) break;

    // During infinite loop, monitor if GPIO pin needs writing from low to high
    if (main_loop_active && !main_pin_status) {
        write_pin(Main_Monitor_GPIO_Port, Main_Monitor_Pin, true);
        main_pin_status = true;
    }

```

```
      if (sample_interrupt_occurred) {
          sample_processing_routine();
      }

      if (trigger_flag) {
              trigger_flag = 0;
              process_trigger(1);
            button = 1;
          }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

```

Besides this, after a time period predefined in *stm32h7xx_it.c*, the variable *trigger_flag* is set to 1 and the function *process_trigger(N)* starts the stimulation, that must occur within the acquisition period. This function needs to be called with N as a unique integer representing a specific trigger. The value 1 used in the code matches the *trigger_source* value for auto-generated biphasic and triphasic waveform, as explained in subsection 3.1.3. The maximum value for N is limited to NUM_STIM_SEQUENCES, (seen above in subsection 3.1.1).

```
// stm32h7xx_it.c file
bool stim_ok = 1;

    void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM3)
    {
        // 20 kHz task
        sample_interrupt_occurred = true;
        if(stim_ok){
        counter_stim++;
        }

        // Set trigger_flag = 1 after 125 ms
        if (counter_stim >= 2500)
        {
            counter_stim = 0;
            trigger_flag = 1;
            stim_ok = 0;
        }
    }
}
```

After the loop is escaped the accumulated samples from a chosen channel are transmitted via USART in many small DMA transfers. The on-chip memory limitations of the STM32 will not allow for long recordings , especially at high sample rates and channel counts. The previously allocated memory is then freed and the functions *correction_pos_phase()* and *correction_neg_phase()*, explained in subsection 3.1.3, are executed. If the variable *change_voltage_value* is set to true the green LED is switched off, indicating the end of the acquisition and the program enters a final infinite loop. If its value remains false the green LED is switched off and a software reset of the microcontroller is triggered, bringing the processor core and all on-chip peripherals to their default states, except for the values in the *.noinit* section, and restarting the firmware execution from the beginning. In Fig. 3.1 the flow chart of the *main.c* code is shown.

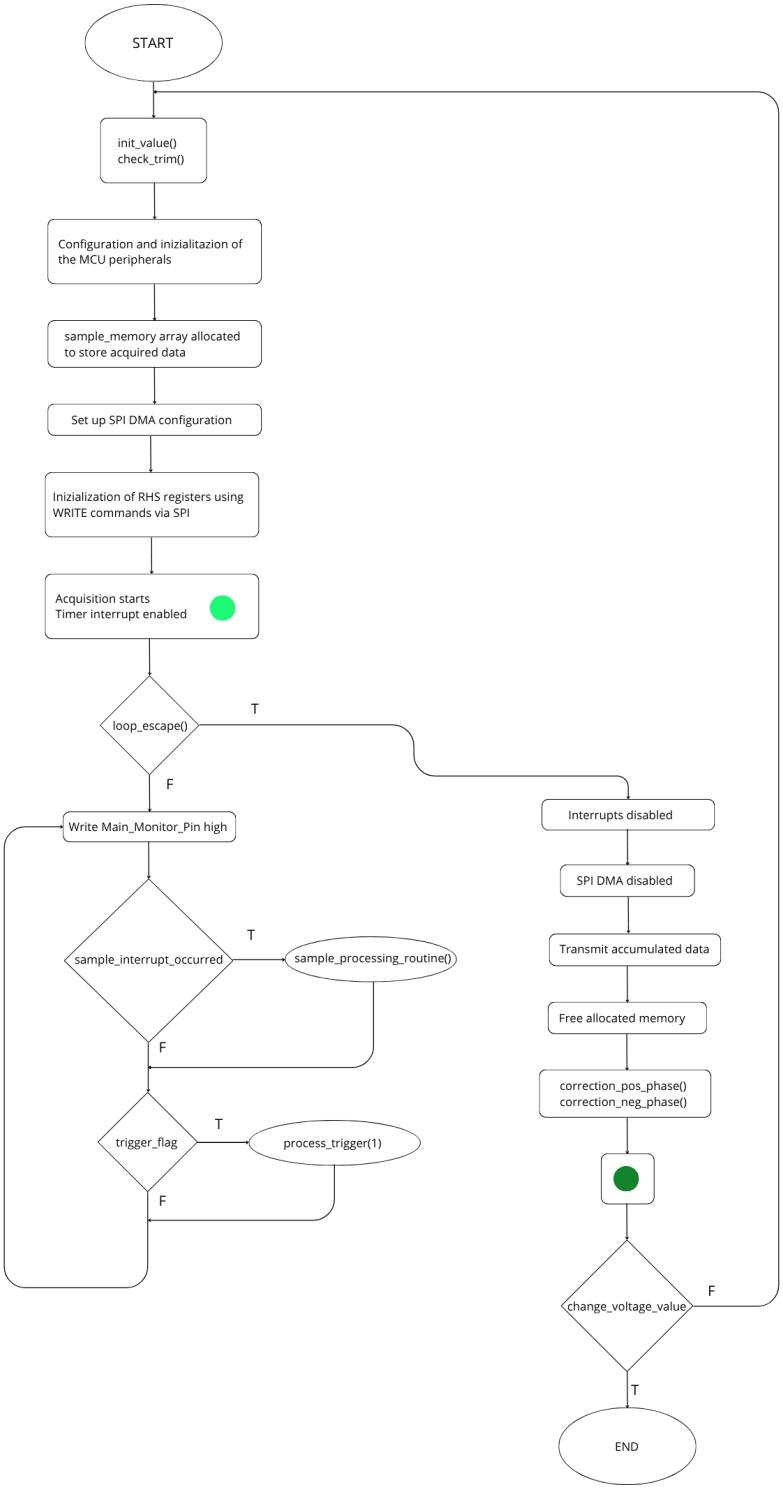


Figure 3.1: Flowchart of *main.c*

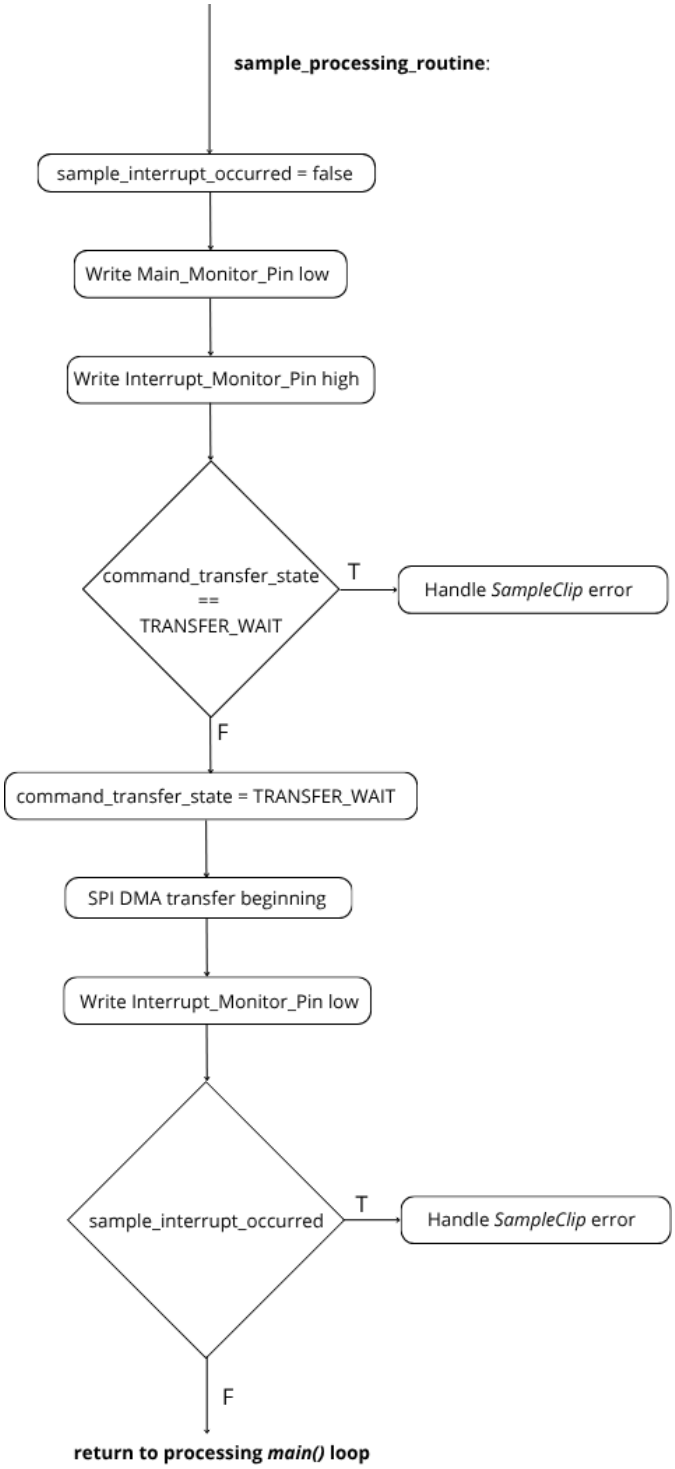


Figure 3.2: Flowchart of *sample_processing_routine()*

3.1.3 *userfunctions.c*

In the file *userfunctions.c* the main functions that allow the stimulation are defined. To better understand how they work is necessary to introduce the stimulation current step size and the time step size. Each stimulation sequence (the structure of which is further explained below) consists of a series of amounts of current and periods of time for which to hold those currents. However, the units for these values are not fixed. The period for a single time step is equivalent to the reciprocal of the sample rate. So, at the default sample rate of 20 kHz, it is equal to 50 µs. The current step size, named stim step size, can range from 10 nA to 10 µA. The function *set_stim_step_size()* and the following enum, defined in *rhsregisters.h*, allow the user to easily set this parameter:

```
// rhsregisters.h file

    typedef enum StimStepSize {
    StimStepSizeUnrecognized = -1,
    StimStepSizeMin = 0,
    StimStepSize10nA = 1,
    StimStepSize20nA = 2,
    StimStepSize50nA = 3,
    StimStepSize100nA = 4,
    StimStepSize200nA = 5,
    StimStepSize500nA = 6,
    StimStepSize1uA = 7,
    StimStepSize2uA = 8,
    StimStepSize5uA = 9,
    StimStepSize10uA = 10,
    StimStepSizeMax = 11
} StimStepSize;
```

Going back to *userfunctions.c*, at the top of the file the variables and the macros are declared:

```
// Thresholds to detect the stimulation segment
float peak_pos = 0.5f;
float peak_neg = -0.5f;

// Desired voltage amplitude
float TARGET_V_DC;

// Calibration & correction parameters
#define TOLERANCE   0.01*TARGET_V_DC  // Tolerable error
#define CORRECTION_STEP_POS   2
#define CORRECTION_STEP_NEG   2

#define SEGMENT_DURATION 4000
```

```
14
15 bool change_voltage_value = false;
16 bool pos_trim_ok = false;
17
18 float v = 0;
19
20 float buffer_pos[SEGMENT_DURATION];
21 int index_pos = 0;
22 int len_pos = 0;
23
24 float buffer_neg[SEGMENT_DURATION];
25 int index_neg = 0;
26 int len_neg = 0;
27
28 bool store_pos = 1;
29 bool store_neg = 1;
```

This work takes advantage of the possibility to auto-generate biphasic and triphasic stimulation waveforms starting from two example waveforms supplied by the Intan firmware framework. These waveforms, one biphasic and one triphasic, are populated, validated, and created into their respective StimSequence structures in the *configure_stim_sequences()* function. Below the BiphasicWaveform and TriphasicWaveform components are described:

- **channel:** number of the channel on which the waveform should occur on (0-15);
- **trigger_source:** number corresponding to the trigger source for this stimulation sequence. A value of 0 represents inactive or no trigger, while 1 originally represented a manual trigger generated by rising-edge activity on the GPIO pin routed to the blue button on the NUCLEO board. However, in this implementation, the value of 1 is now associated with the automatic trigger controlled by the variable *trigger_flag*;
- **num_pulses:** number representing how many pulses the waveform should be made of, limited to a maximum value of MAX_NUM_PULSES_PER_TRAIN
- **pulse_train_period:** this number represents the number of time steps between the start of each pulse within the pulse train. If num_pulses is equal to 1, this number must be set to 0;
- **resting_amplitude:** in units of stim step size, the current amplitude at which the channel is held before and after stimulation pulses (and, if interphase_delay is not 0, between the first and second phases);
- **first_phase_amplitude:** in units of stim step size, the current amplitude at which the channel is held during the first stimulation phase;
- **second_phase_amplitude:** in units of stim step size, the current amplitude at which the channel is held during the second stimulation phase;

- **third_phase_amplitude:** in units of stim step size, the current amplitude at which the channel is held during the third stimulation phase;
- **post_trigger_delay:** number of time steps after the trigger event specified in trigger_source occurs, but before the first phase of stimulation occurs;
- **first_phase_duration:** number of time steps the channel is held at first_phase_amplitude;
- **interphase_delay:** number of time steps the channel is held at resting_amplitude between the first and second phases of stimulation;
- **second_phase_duration:** number of time steps the channel is held at second_phase_amplitude;
- **third_phase_duration:** number of time steps the channel is held at third_phase_amplitude;
- **refractory_period:** number of time steps after the last stimulation pulse phase ends for which subsequent triggers are ignored. This must be at least as large as the largest of post_stim_amp_settle, post_stim_charge_recovery_on, and post_stim_charge_recovery_off;
- **enable_amp_settle:** if true, the amplifier settle is enabled at any point during this waveform. If false, all the following parameters regarding amplifier settle must be set to 0;
- **pre_stim_amp_settle:** number of time steps before the first stimulation phase occurs for which amplifier settle is active. This number should not be larger than post_trigger_delay;
- **post_stim_amp_settle:** number of time steps after the last stimulation pulse phase occurs for which amplifier settle is active. This number should not be larger than refractory_period;
- **maintain_amp_settle:** if true, amplifier settle is held high even during the inactive portion of pulse_train_period during which stimulation is not occurring. If false, amp settle will only be held high in the periods before, during, and after each individual pulse;
- **enable_charge_recovery:** if true, post-stimulation charge recovery is enabled at any point during this waveform. If false, all the following parameters regarding charge recovery must be set to 0;
- **post_stim_charge_recovery_on:** number of time steps after the last stimulation pulse phase occurs before the charge recovery function begins. This number should not be larger than post_trigger_delay, or larger than post_stim_charge_recovery_off;
- **post_stim_charge_recovery_off:** number of time steps after the last stimulation pulse phase occurs before the charge recovery function ends. This number should not be larger than refractory_period, or smaller than post_stim_charge_recovery_on.

A visual example of these parameters is shown in Fig. 3.3 and Fig. 3.4.

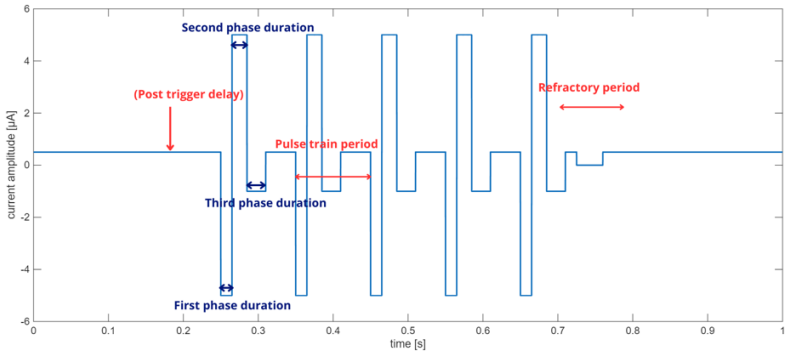


Figure 3.3: Time parameters of TriphasicWaveform

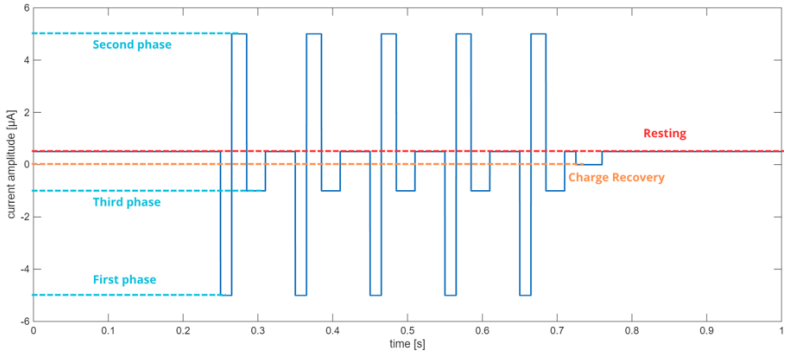


Figure 3.4: Amplitude parameters of TriphasicWaveform

The "validate" functions check if the parameters set by the user respect the rules imposed by the firmware and if any violations are discovered, the program enters an infinite loop and the *InvalidStimWaveform* error is set.

```
    void configure_stim_sequences(void)
{
    // Define settings for a biphasic waveform by populating the BiphasicWaveform
    structure.
    BiphasicWaveform biphasic_example = {
            .channel = 3,
            .trigger_source = 1,
            .num_pulses = 3,
            .pulse_train_period = 12000,
            .resting_amplitude = 2,
            .first_phase_amplitude = amp_parameters.mag_pos,
            .second_phase_amplitude = amp_parameters.mag_neg,
//          .first_phase_amplitude = 30,
//          .second_phase_amplitude = -30,
            .post_trigger_delay = 20,
            .first_phase_duration = 4000,
```

```
            .interphase_delay = 0,
            .second_phase_duration = 4000,
            .refractory_period = 4000,
            .enable_amp_settle = true,
            .pre_stim_amp_settle = 20,
            .post_stim_amp_settle = 4000,
            .maintain_amp_settle = true,
            .enable_charge_recovery = true,
            .post_stim_charge_recovery_on = 0,
            .post_stim_charge_recovery_off = 2000
    };

    // Define settings for a triphasic waveform by populating the TriphasicWaveform
     structure.
    TriphasicWaveform triphasic_example = {
            .channel = 1,
            .trigger_source = 1,
            .num_pulses = 5,
            .pulse_train_period = 2000,
            .resting_amplitude = 5,
            .first_phase_amplitude = -50,
            .second_phase_amplitude = 50,
            .third_phase_amplitude = -10,
            .post_trigger_delay = 0,
            .first_phase_duration = 300,
            .second_phase_duration = 400,
            .third_phase_duration = 500,
            .refractory_period = 1200,
            .enable_amp_settle = true,
            .pre_stim_amp_settle = 0,
            .post_stim_amp_settle = 600,
            .maintain_amp_settle = true,
            .enable_charge_recovery = true,
            .post_stim_charge_recovery_on = 300,
            .post_stim_charge_recovery_off = 1000
    };


    // Check validity of parameters specified above; if any invalid combinations
    are present
    validate_biphasic(&biphasic_example_ch8);
    validate_triphasic(&triphasic_example_ch9);

    // Create a biphasic waveform at sequence 0 on channel 3
    create_biphasic_waveform(&sequences[0], &biphasic_example);

    // Create a triphasic waveform at sequence 1 on channel 1
    create_triphasic_waveform(&sequences[1], &triphasic_example);

```

```
    // Set stim step size
    set_stim_step_size(&parameters, StimStepSize100nA);

    // Read segment 0 for all sequences into parameter values, updating initial
    parameter settings
    for (int i = 0; i < NUM_STIM_SEQUENCES; i++) {
        set_parameters_for_segment(sequences[i].segments[0], sequences[i].channel);
    }

    // Immediately write these changed register values via SPI
    sync_parameters_immediate();
}
```

The calibration procedure implemented in the project and described below is designed and optimized only for the biphasic waveform, since it is the one used in neural stimulation. Therefore, from this point forward, unless otherwise specified, the expression 'stimulation waveform' refers exclusively to the biphasic waveform.

Through the *write_data_to_memory()* function samples are saved into the *sample_memory* array and the positive and negative phase of the stimulation waveform are detected by comparison with the threshold values *peak_pos* and *peak_neg*. For DC-coupled low-gain amplifiers [30], the electrode voltage is given by:

$$V_{\text{elec(DC)}} = -19.23 \text{ mV} \times (\text{ADC result} - 512) \tag{3.1}$$

The formula used to calculate the variable v comes from the previous equation.

```
    void write_data_to_memory(void)
{
#ifdef OFFLINE_TRANSFER
    // Save single sample to sample_memory array.
    for (int i = 0; i < NUM_SAMPLED_CHANNELS; i++) {
        uint32_t raw_sample = command_sequence_MISO[FIRST_SAMPLED_CHANNEL + i + 2];
        sample_memory[(sample_counter * NUM_SAMPLED_CHANNELS) + i] = raw_sample;

     // If stimulation occured check the values of the samples
     if(button==1){
        uint16_t wordDC = (uint16_t)(raw_sample & 0xFFFF);
        v = -0.01923f * ((float)wordDC - 512.0f);

        // Store values > peak_pos
        if (store_pos && v > peak_pos){
            //peak_pos = v;
            buffer_pos[index_pos] = v;
            index_pos++;
        }
        // Store values < peak_neg
```

```
21          else if (store_neg && v < peak_neg){
22              //peak_neg = v;
23              buffer_neg[index_neg] = v;
24              index_neg++;
25          }
26          // This part is added in order to have enough values in the buffers to mean
27
28          // If there are enough values and the segment ended disable the storage
29          else{
30              if(index_pos >= SEGMENT_DURATION-10 ){
31                  store_pos = 0;
32                  len_pos = index_pos;
33                  index_pos=0;
34              }
35              if(index_neg >= SEGMENT_DURATION-10 ){
36                  store_neg = 0;
37                  len_neg = index_neg;
38                  index_neg = 0;
39              }
40              // If there aren't enough successive values not consider them
41              else index_pos = 0, index_neg = 0;
42              }
43           }
44
45          }
46          sample_counter++;
47  #endif
48  }
```

After the end of acquisition, reported by assigning 1 to the variable *acquisition_finished* in *main.c*, corrections are made either to current amplitude value or to trim value through *correction_pos_phase()* and *correction_neg_phase()* functions, separately for the positive and the negative phase. In both these procedures, the samples collected from the corresponding stimulation waveform phase are averaged and the error is calculated as the difference between this value and the TARGET_V_DC. If the related percentage is greater than 5% a 'rough' correction is made to *amp_parameters.mag_pos* or *amp_parameters.mag_neg* value, otherwise a 'fine' adjustment is applied to *amp_parameters.trim_pos* or *amp_parameters.trim_neg* value, used in *stimscheduler.c* file. In the firmware the trim value ranges from -128 to 127, with the 0 midpoint being equivalent to no trim. This variable can be used to modulate the value of stimulation current over a range of approximately ±28% [30]. Values less than 0 reduce the current by approximately 0.2%/step, values greater than 0 increase the current by approximately 0.22%/step.

When both positive and negative phase related errors are smaller than TOLERANCE, the variable *change_voltage_value* is set to true. In Fig. 3.5 and Fig. 3.6

the flow charts of these functions are reported to better understand their operation.

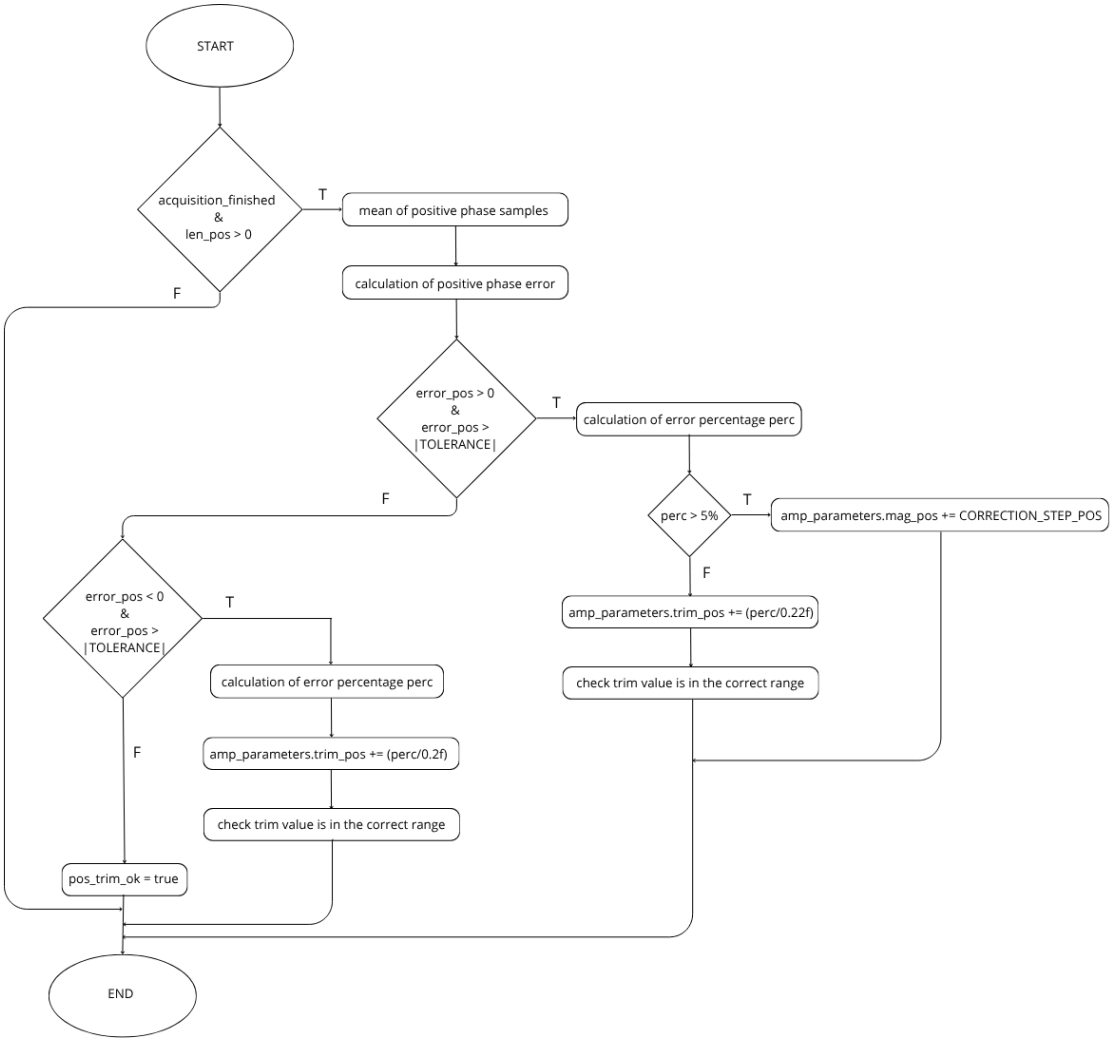


Figure 3.5: Flow chart of *correction_pos_phase()*.

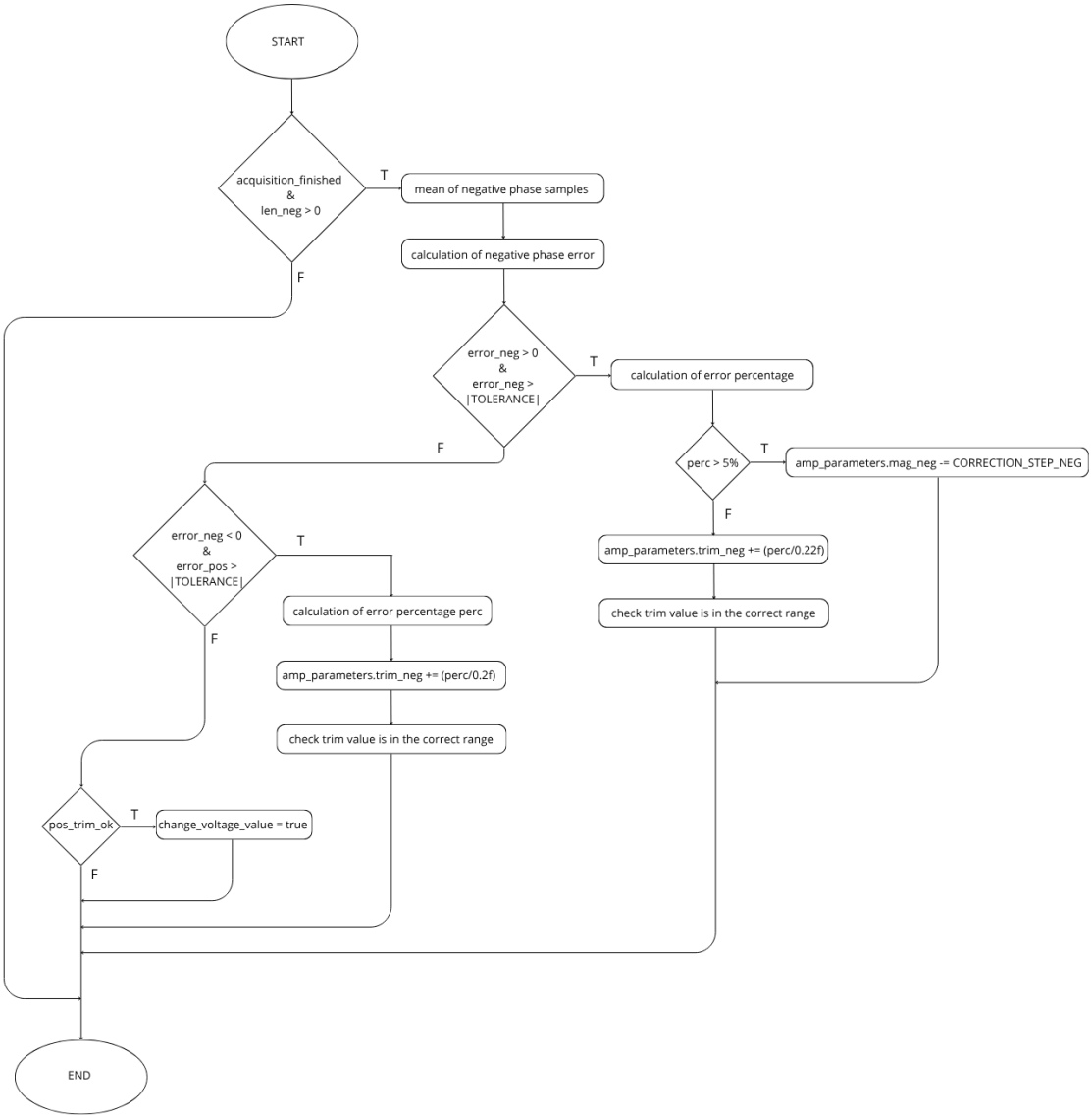


Figure 3.6: Flow chart of *correction_neg_phase()*.

3.2 Introduction of BLE communication

The second main step consists of introducing the opportunity for the user to send the desired values for the stimulation waveform to the STM32 board via BLE. To achieve this goal the NUCLEO-H723ZG is replaced by the NUCLEO-WBA65RI. The code described in section 3.1 is adapted to the new board used. Moving from a high-performance architecture to an ultra-low-power wireless MCU requires modifications to the DMA, SPI, timer, clock, and GPIO configurations.

To introduce BLE communication the STM32WBA *Peer to Peer Server* (P2P) application was chosen as the starting point [31]. It contains the P2P Service, which exposes two characteristics (Notification and Write) in order to create a bidirectional data communication. The WBA board plays the role of the GATT server. In this testing phase, a smartphone acts as the client via the STBLEToolbox application. The sample frequency used from now on is 10 kHz.

3.2.1 *p2p_server_app.c*

The file *p2p_server_app.c* contains the operations performed by the server and a state machine to correctly interpret data coming from the client.

The write without response event of the characteristic originally dedicated to control LED1 is modified to receive text-based input. The payload transmitted by the user is processed as an ASCII string and then *atoi()* and *atof()* functions are implemented to convert it in a variable of type *int* or *float*. In the *main.h* file a enum to check the end of parameters setting and a structure to store the number obtained after the conversion are defined:

```
//main.h
typedef enum {
    STATE_BLE_WAITING,
    STATE_BLE_RECEIVED,
    STATE_ACQUISITION_STIMULATION
} GlobalState_t;

typedef struct{
    uint8_t cmd_type;
    uint16_t channel;
    uint16_t num_pulses;
    int current_amplitude;
    uint16_t resting;
    float pulse_train_period;
    float phase_duration;
    float interphase_delay;
    float post_trigger_delay;
    float refractory_period;
    bool enable_amp_settle;
    float pre_amp_settle;
    float post_amp_settle;
    bool mantain_amp_settle;
    bool enable_charge_recovery;
    float charge_rec_on;
    float charge_rec_off;
    uint16_t step_size;
} StimulationParams_t;
```

A switch statement interprets the number obtained depending on the value of *actual_state*.

```

SystemState_t actual_state = MAIN_MENU_STATE;

// stim_params saved in .noinit section
__attribute__((section(".noinit"))) StimulationParams_t stim_params;

void P2P_SERVER_Notification(P2P_SERVER_NotificationEvt_t *p_Notification)
{
  /* USER CODE BEGIN Service1_Notification_1 */

  /* USER CODE END Service1_Notification_1 */
  switch(p_Notification->EvtOpcode)
  {
    /* USER CODE BEGIN Service1_Notification_Service1_EvtOpcode */

    /* USER CODE END Service1_Notification_Service1_EvtOpcode */

    case P2P_SERVER_LED_C_READ_EVT:
      /* USER CODE BEGIN Service1Char1_READ_EVT */

      /* USER CODE END Service1Char1_READ_EVT */
      break;

    case P2P_SERVER_LED_C_WRITE_NO_RESP_EVT:
      /* USER CODE BEGIN Service1Char1_WRITE_NO_RESP_EVT */

       // Buffer created to contain ASCII string sent (max 20 characters)
       char rx_buffer[20];
       uint8_t len = p_Notification->DataTransfered.Length;

       // Avoid buffer overflow
       if(len > 19) len = 19;

      // Copy of received data in the buffer and addition of string terminator
       memcpy(rx_buffer, p_Notification->DataTransfered.p_Payload, len);
       rx_buffer[len] = '\0';

       // If the user sends "0", force return to the main menu from any state
       if (rx_buffer[0] == '0' && len == 1) {
        actual_state = MAIN_MENU_STATE;
        APP_BSP_LED_Off(LED_BLUE);
        printf("\r\n[INFO] Back to Main Menu (Received 0)\r\n");
        PrintMainMenu();
        break;
    }

```

```
47      // State Machine
48      switch(actual_state)
49      {
50          case MAIN_MENU_STATE:
51              int choice = atoi(rx_buffer);
52          if (choice == 1) {
53
54              /* Code omitted: printing of parameters */
55
56              actual_state = MAIN_MENU_STATE;
57              APP_BSP_LED_Off(LED_BLUE);
58          }
59          else if (choice == 2) {
60              printf("\r\n[MENU] Parameters reset.\r\n");
61              stim_params = (StimulationParams_t){0};
62              APP_BSP_LED_Off(LED_BLUE);
63          }
64          else if (choice == 3) {
65              printf("\r\n[MENU] Send data. Disconnecting BLE and starting
    stimulation... \r\n");
66
67              // Start disconnection
68              Force_BLE_Disconnect();
69
70              // Reset menu state
71              actual_state = MAIN_MENU_STATE;
72              APP_BSP_LED_Off(LED_BLUE);
73          }
74          else if (choice == 4) {
75              actual_state = CHANNEL_WAIT_STATE;
76              APP_BSP_LED_On(LED_BLUE);
77              printf("\r\n[MENU] 'Set the channel'. Enter the value:\r\n");
78          }
79
80          /* Code omitted: The following condintional branches have the same
    structure of the one just above */
81          /*...*/
82
83          case CHANNEL_WAIT_STATE:
84              stim_params.channel = atoi(rx_buffer);
85              printf("\r\n[OK] Selected channel: %d\r\n",
     stim_params.channel);
86
87              // Number entered, back to main menu
88              actual_state = MAIN_MENU_STATE;
89              APP_BSP_LED_Off(LED_BLUE);
90              printf("Back to menu... (Send a number in the menu or send 0 to print
    the menu)\r\n");
91              break;
```

```
92
93          /* Code omitted : The following case branches have the same structure of
        the one just above */
94          /*...*/
95
96      }
97      /* USER CODE END Service1Char1_WRITE_NO_RESP_EVT */
98      break;
99
100     case P2P_SERVER_SWITCH_C_NOTIFY_ENABLED_EVT:
101       /* USER CODE BEGIN Service1Char2_NOTIFY_ENABLED_EVT */
102       P2P_SERVER_APP_Context.Switch_c_Notification_Status =
        Switch_c_NOTIFICATION_ON;
103       printf("-- P2P APPLICATION SERVER : NOTIFICATION ENABLED\n");
104       printf(" \n\r");
105       PrintMainMenu();
106       /* USER CODE END Service1Char2_NOTIFY_ENABLED_EVT */
107       break;
108
109     case P2P_SERVER_SWITCH_C_NOTIFY_DISABLED_EVT:
110       /* USER CODE BEGIN Service1Char2_NOTIFY_DISABLED_EVT */
111       P2P_SERVER_APP_Context.Switch_c_Notification_Status =
        Switch_c_NOTIFICATION_OFF;
112       printf("-- P2P APPLICATION SERVER : NOTIFICATION DISABLED\n");
113       printf(" \n\r");
114       /* USER CODE END Service1Char2_NOTIFY_DISABLED_EVT */
115       break;
116
117     default:
118       /* USER CODE BEGIN Service1_Notification_default */
119
120       /* USER CODE END Service1_Notification_default */
121       break;
122   }
123   /* USER CODE BEGIN Service1_Notification_2 */
124
125   /* USER CODE END Service1_Notification_2 */
126   return;
127 }
```

When the client enables notifications on the notify characteristic, the server triggers the *PrintMainMenu()* function to output the interactive configuration menu, shown in Fig. 3.7, via USART. In our case, it is viewed on MATLAB, but it is also possible to use a serial terminal emulator, such as PuTTY.

```
-- P2P APPLICATION SERVER : NOTIFICATION ENABLED
================================
STIMULATION PARAMETERS SETTING
================================
1: Print current parameters
2: Reset parameters
3: Send data
4: Select the channel
5: Select the number of pulses
6: Select the Stim Step Size
7: Set the amplitude of the positive and negative phase
8: Set the resting amplitude
9: Set pulse train period
10: Set the duration of the positive and negative phase
11: Set the interphase delay
12: Set post trigger delay
13: Set refractory period
14: Set the amplitude settle
15: Set charge recovery
--------------------------------
NB: If you want to set a parameter to 0 you must send 00 or 0.0
Send a number from ST BLE Toolbox...
```

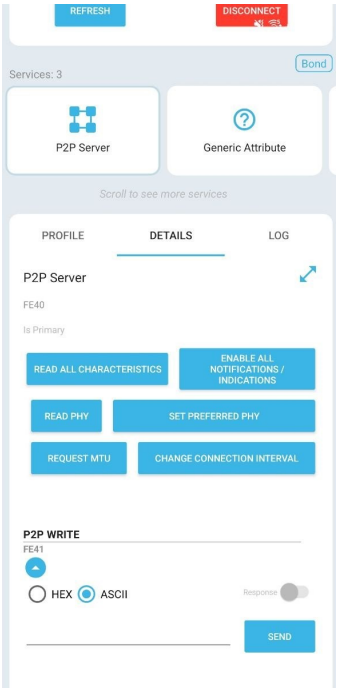


Figure 3.7: On the left is shown the interactive configuration menu developed in this work and on the right the application screen where the user enters the values.

The first option allows the user to review the values entered, while the second resets all parameters to 0. In the range of 4 to 15, depending on the number inserted in the app, a certain state between the ones reported below is assigned to *actual_state*:

```
    typedef enum {
    MAIN_MENU_STATE = 0,
    CHANNEL_WAIT_STATE = 1,
    NUM_PULSES_STATE = 2,
    STIM_STEP_SIZE_STATE = 3,
    PHASE_AMPLITUDE_STATE = 4,
    RESTING_AMPLITUDE_STATE = 5,
    PULSE_TRAIN_PERIOD_STATE = 6,
    PHASE_DURATION_STATE = 7,
    INTERPHASE_DELAY_STATE = 8,
    POST_TRIGGER_DELAY_STATE = 9,
    REFRACTORY_PERIOD_STATE = 10,
    AMP_SETTLE_STATE = 11,
    CHARGE_RECOVERY_STATE = 12,
} SystemState_t;
```

For each state except *MAIN_MENU_STATE* the user has to insert one or more values for the parameter selected as shown in the *case* example at line 87 of the *switch()* block in the code frame above. The number inserted is converted to the *int* or *float* type depending on the parameter and the value obtained is stored in the corresponding field of *stim_params* structure. The *MAIN_MENU_STATE* is then restored.

When all the parameters have been set, the option '3' of the menu has to be selected to force disconnection between the board and the smartphone and start the stimulation. The *Force_BLE_Disconnect()* function is executed.

P2P_SERVER_APP_Context.ConnectionHandle is the identifier of the active connection and the code *0x13* indicates *"Remote User Terminated Connection"*.

```
void Force_BLE_Disconnect(void)
{
aci_gap_terminate(P2P_SERVER_APP_Context.ConnectionHandle, 0x13);
}
```

An automatic system event called *P2P_SERVER_DISCON_HANDLE_EVT* is triggered. It changes the value of *CurrentGlobalState* variable, defined in *main.c*, to *STATE_BLE_RECEIVED*. Moreover, the *UTIL_SEQ_PauseTask()* function is called to suspend the notification task. This prevents the microcontroller from attempting to send BLE packets when no client is connected, saving computational resources.

```
void P2P_SERVER_APP_EvtRx(P2P_SERVER_APP_ConnHandleNotEvt_t *p_Notification)
{
  /* USER CODE BEGIN Service1_APP_EvtRx_1 */

  /* USER CODE END Service1_APP_EvtRx_1 */

  switch(p_Notification->EvtOpcode)
  {
    /* USER CODE BEGIN Service1_APP_EvtRx_Service1_EvtOpcode */

    /* USER CODE END Service1_APP_EvtRx_Service1_EvtOpcode */
    case P2P_SERVER_CONN_HANDLE_EVT :
    /* USER CODE BEGIN Service1_APP_CONN_HANDLE_EVT */
    // Save connection handle to execute aci_gap_terminate later
   P2P_SERVER_APP_Context.ConnectionHandle = p_Notification->ConnectionHandle;
    /* USER CODE END Service1_APP_CONN_HANDLE_EVT */
    break;

    case P2P_SERVER_DISCON_HANDLE_EVT :
    /* USER CODE BEGIN Service1_APP_DISCON_HANDLE_EVT */
    P2P_SERVER_APP_LED_BUTTON_context_Init();

```

```
23      if (currentGlobalState == STATE_BLE_WAITING)
24          {
25              currentGlobalState = STATE_BLE_RECEIVED;
26
27              UTIL_SEQ_PauseTask(1U << CFG_TASK_SEND_NOTIF_ID);
28          }
29
30        /* USER CODE END Service1_APP_DISCON_HANDLE_EVT */
31        break;
32
33      default:
34        /* USER CODE BEGIN Service1_APP_EvtRx_default */
35
36        /* USER CODE END Service1_APP_EvtRx_default */
37        break;
38    }
```

3.2.2 *main.c*

In this step of code development, changes are made to the *main.c* file in order to first manage the communication via BLE with the smartphone, then the impedance check and finally the acquisition and stimulation procedure described in section 3.1. Below are reported the main definitions:

```
1 #define SYSTEM_READY_SIGNATURE 0x5A5A5A5A
2
3 // variables stored in .noinit section
4 __attribute__((section(".noinit"))) Parameters_t amp_parameters;
5 __attribute__((section(".noinit"))) uint32_t restart_stimulation;
```

The SYSTEM_READY_SIGNATURE is used to be able to differentiate between the first code execution, during which data from the application have to be received and impedance check has to be implemented, from the following executions necessary only for the calibration.

In the *main()* routine, whose flow chart is shown in Fig. 3.8 the three main stages cited are handled.

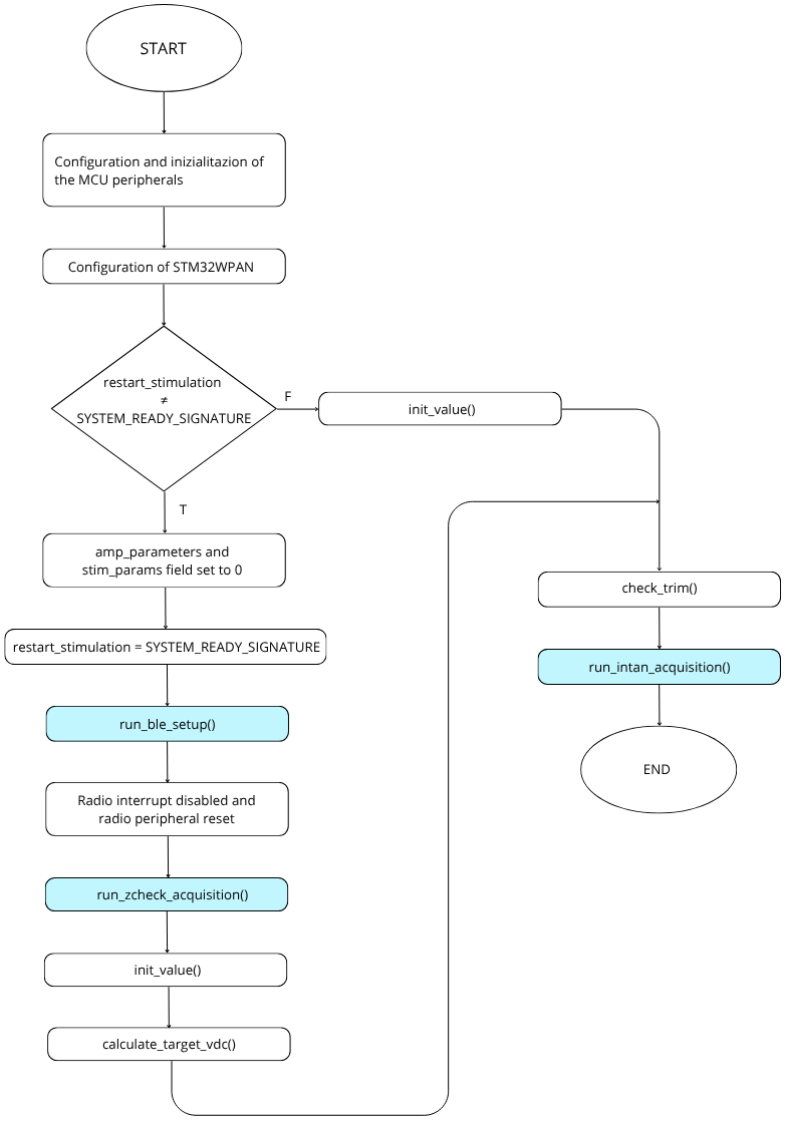


Figure 3.8: Flow chart of *main()* routine in *main.c* file.

STM32WPAN (STM32 Wireless Personal Area Network) is an embedded software middleware package by STMicroelectronics designed to support wireless networking protocols, such as BLE, on wireless-enabled STM32 microcontrollers like STM32WBA series.

The *MX_APPE_Config()* initialization function activates the board's Bluetooth subsystem. When the board is powered on for the first time, all the fields of *amp_parameters* and *stim_params* are set to 0.

Data Exchange via BLE

Through the *run_ble_setup()* routine the main program flow is halted and *MX_APPE_Process()* is continuously called to handle incoming radio events and user inputs. This loop is broken when the state transition to *STATE_BLE_RECEIVED* is triggered.

```
void run_ble_setup(void)
{
    // Initialize BLE enviroment
    MX_APPE_Init(NULL);
    printf("==========================================\r\n");
    printf("   SETUP MODE     \r\n");
    printf("==========================================\r\n");
    printf("Waiting for parameters from ST BLE Toolbox...\r\n");

    while (currentGlobalState != STATE_BLE_RECEIVED)
    {
        MX_APPE_Process(); // Manage background radio events
    }
}
```

Subsequently, the board disconnects from the smartphone and the reset of the radio peripheral is forced. Impedance check is then implemented.

Impedance Check

THE RHS2116 chip provides built-in circuitry to select and directly access any of the amplifier input pins and additional on-chip circuitry to generate a low-magnitude AC current waveform for the purpose of measuring the impedance of electrodes connected to the chip.

The waveform generator consists of an 8-bit digital-to-analog converter (DAC) followed by a two-pole 10 kHz low-pass filter to smooth the “stairstep” edges of the DAC waveform. The voltage produced by the DAC varies from a minimum of 0 V to a maximum of $(255/256) \times 1.225\ \text{V} = 1.220\ \text{V}$. To approximate a continuous sinusoidal test signal, the internal DAC must be continuously updated. A series capacitor Cs connects the resulting "test waveform" to the selected electrode and converts the AC voltage into AC current. Cs value can be chosen between 0.1 pF, 1.0 pF, or 10 pF. The resulting voltage waveform on the electrode passes through a high-gain amplifier and may be observed by the ADC.

The impedance can be calculated as the ratio of the resulting peak voltage to peak current.

Impedance check is managed through the *run_zcheck_acquisition()* function as explained by the flow chart in Fig. 3.9. The following values are used:

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define NUM_MEASUREMENTS        20U     // number of impedance measures
#define MEAS_INTERVAL_MS        50U   // time interval between measures in ms
/* USER CODE END PD */
```

The actual impedance calculation is implemented within the function named *perform_impedance_measurement()*, defined in *userfunctions.c* file and whose flow chart is represented in Fig. 3.10. The procedure enables the DAC that works as waveform generator. Then the amplifier connected to this, the value of the capacitor Cs and the electrode are selected. The generator is turned on and within the *run_sampling_loop()* routine the *sample_processing_routine()* function retrieves the next discrete value from a pre-calculated array and sends it to the DAC. At the same time, it acquires the incoming ADC voltage sample. Subsequently, the values of the samples are converted to μV and the amplitude and phase of the voltage waveform are estimated through a digital dual-phase demodulation algorithm [32] to extract the AC signal of interest from a noisy background. The formula for computing the current peak is obtained from the following equation from the RHS datasheet[30]:

$$I_{\mathrm{DAC}}(t) = 2\pi f C_{\mathrm{S}} V_{\mathrm{A}} \cos(2\pi f t) \tag{3.2}$$

Below is the code frame that calculates impedance value, where M_PI is equal to π value, the *amplitude_uv* field of the *result* structure contains the voltage waveform amplitude estimated, while in the *test_frequency_hz* the test waveform frequency value is saved and *c_series* variable is equal to the chosen value for Cs. For this project a frequency of 1 kHz and a Cs value of 1 pF were used.

```
#define IMPEDANCE_TEST_DAC_AMPLITUDE_STEPS     128.0
// Datasheet: amplitude max = 1.225V
// Sine wave -> amplitude max =  1.225V / 2 = 0.6125V
double dac_voltage_amplitude = (IMPEDANCE_TEST_DAC_AMPLITUDE_STEPS / 128.0) *
    0.6125;

// Compute actual current
double total_current_amps = 2.0 * M_PI * result->test_frequency_hz * c_series *
    dac_voltage_amplitude;

// Convert voltage value from uV to V
double voltage_volts = result->amplitude_uv * 1.0e-6;

//Ohm's Law: Z = V / I
double z_raw_ohm = voltage_volts / total_current_amps;
```

The *run_zcheck_acquisition()* function keeps track of the minimum and maximum impedance value obtained and computes the mean and the median of the values stored. To visually signal the performing impedance measurements the blue led on the NUCLEO-WBA65RI, LD1, is turned on at the beginning and turned off at the end. The values measured are transmitted via USART and displayed on MATLAB, as their mean and median.

Then the *init_value()* function is executed and the *calculate_target_vdc()* function employs the median value together with the current amplitude and stimulation step size, entered by the user, to calculate the value to assign to TARGET_V_DC for the calibration procedure.

```
void calculate_target_vdc(void){
amp_parameters.amplitude_desired =  median_impedance_ohm*(step_size_values_nA[
    stim_params.step_size]*(uint32_t)stim_params.current_amplitude)*1e-9;
}
```

To extract the stimulation step size value the lookup table reported below is used:

```
const uint32_t step_size_nA[11] = {
    0,      // 0: StimStepSizeMin
    10,     // 1: 10 nA
    20,     // 2: 20 nA
    50,     // 3: 50 nA
    100,    // 4: 100 nA
    200,    // 5: 200 nA
    500,    // 6: 500 nA
    1000,   // 7: 1 uA
    2000,   // 8: 2 uA
    5000,   // 9: 5 uA
    10000   // 10: 10 uA
};
```

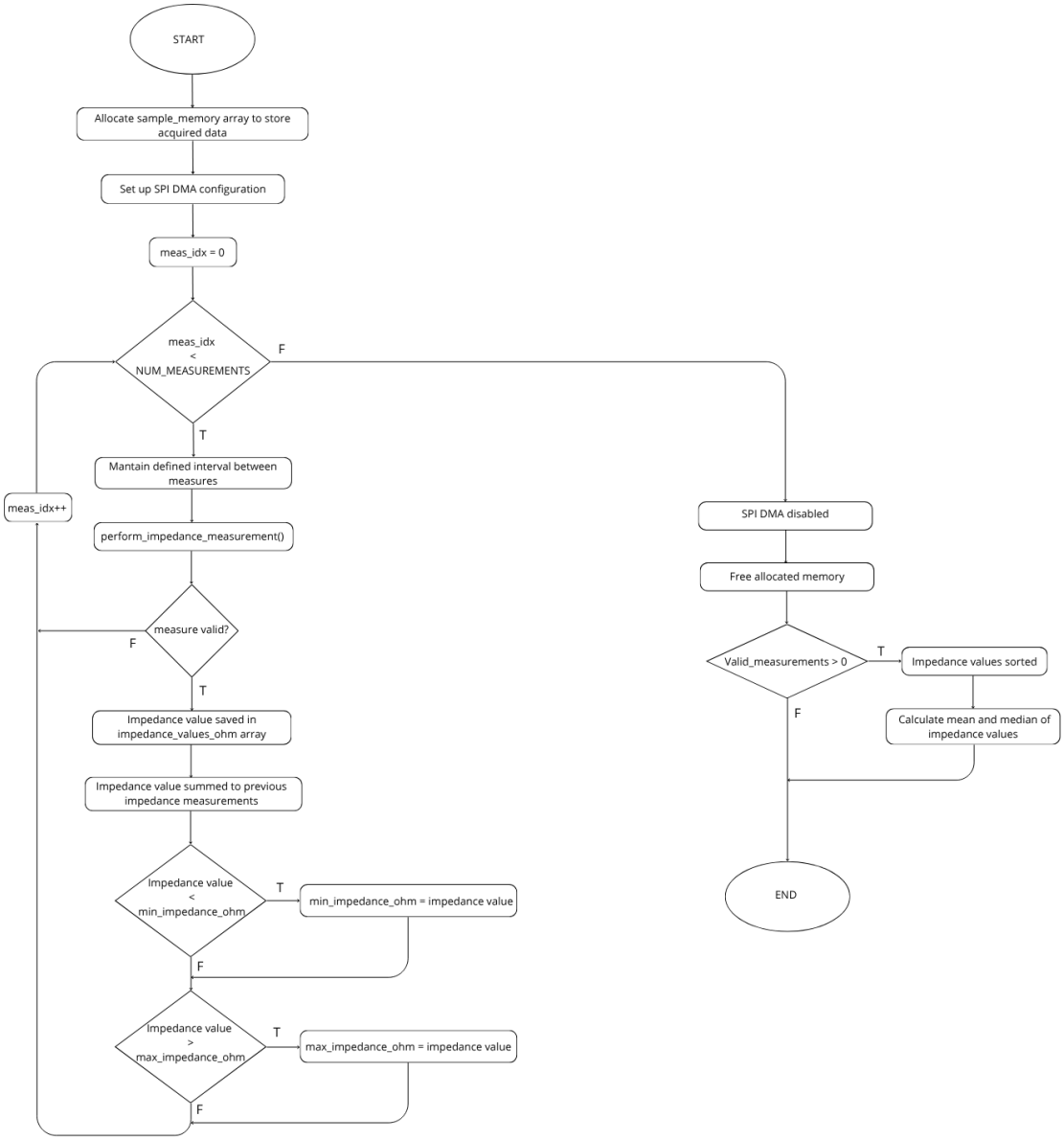


Figure 3.9: Flow chart of *run_zcheck_acquisition()*.

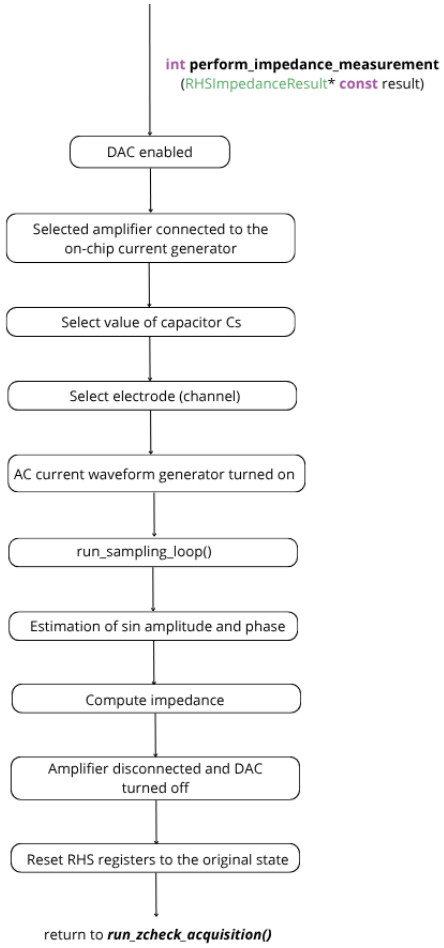


Figure 3.10: Flow chart of *perform_impedance_measurement()*.

Intan Acquisition and Stimulation

In all the code executions the *check_trim()* function, described in section 3.1 and the *run_intan_acquisition()* function are called. This second procedure basically follow the same steps represented in Fig. 3.1 apart from *init_value()* and *check_trim()* functions that are executed before.

3.3 BLE communication with nRF52840 DK

In this third step of the work the STBLEToolbox is replaced by the Nordic nRF52840 DK. To text the performance of the communication from the latter to the NUCLEO WBA65RI, instead of entering every single parameter value, two biphasic waveforms with different values of parameters are assigned to the push

of BUTTON 1 and BUTTON 2, with the possibility to add other waveforms corresponding to BUTTON 3 and BUTTON 4.

3.3.1 *p2p_server_app.c*

The write without response event is modified to receive both text-based input and binary packages. If the first byte read is the character "W", the entire payload received is converted into the specific stimulation struct named *BleWaveCmd_t*. If the character "K" is received a flag is set to true to restart with the successive calibration cycle, if needed, as explained in section 3.3.3.

Moreover, it is checked whether the received string starts with the "BLE_TP" prefix in order to extract the measure of the throughput from the WBA65RI to the Nordic board, when it is transmitted by the nRF52840 DK.

```
case P2P_SERVER_LED_C_WRITE_NO_RESP_EVT:
    /* USER CODE BEGIN Service1Char1_WRITE_NO_RESP_EVT */
{
    // Buffer created to contain ASCII string sent (max 20 characters)
    char rx_buffer[20];
    // Extract the raw packet and its length from the radio
    uint8_t *payload = p_Notification->DataTransfered.p_Payload;
    uint8_t len = p_Notification->DataTransfered.Length;

    // Avoid buffer overflow
    if(len > 19) len = 19;

    // Copy of received data in the buffer ad addition of string terminator
    memcpy(rx_buffer, p_Notification->DataTransfered.p_Payload, len);
    rx_buffer[len] = '\0';

    if (strncmp(rx_buffer, "BLE_TP:", 7) == 0)
        {
            uint32_t throughput = 0;

            sscanf(rx_buffer + 7, "%lu", &throughput);

            printf("\r\n[BLE] BLE throughput WBA -> Nordic: %lu kbps\r\n",
                   throughput);

            break;
        }

    if (payload[0] == 'W') {

    // Raw buffer casted to the structure pointer
    BleWaveCmd_t *waveform_received = (BleWaveCmd_t *)payload;

```

```
34      stim_params.channel = waveform_received->channel;
35
36      /*Code omitted: every parameter is assigned to the correspondig stim_params
        field*/
37      /*...*/
38
39      currentGlobalState = STATE_BLE_RECEIVED;
40
41      uint8_t length = p_Notification->DataTransfered.Length;
42
43      /*Code omitted: printing of stim_params fields*/
44      /*...*/
45
46      printf("\r\n[MENU] Disconnecting BLE and starting stimulation... \r\n");
47
48      // reset to MAIN_MENU_STATE
49      actual_state = MAIN_MENU_STATE;
50
51      // The following state machine is skipped
52      break;
53      }
54      if (rx_buffer[0] == 'K') {
55      matlab_ready = true; // MATLAB received all samples
56      break;
57      }
58
59      // If the user sends "0", force return to the main menu from any state
60      if (rx_buffer[0] == '0' && len == 1) {
61      actual_state = MAIN_MENU_STATE;
62      APP_BSP_LED_Off(LED_BLUE);
63      printf("\r\n[INFO] Back to Main Menu (Received 0)\r\n");
64      PrintMainMenu();
65      break;
66      }
67
68  // State Machine
69      switch(actual_state)
70      {
71          /*Code omitted: same code of section 3.2*/
72          /*...*/
73      }
74  }
75  /* USER CODE END Service1Char1_WRITE_NO_RESP_EVT */
76  break;
```

The *BleWaveCmd_t* struct is defined in the *main.h* file.

```
//main.h

    typedef struct {
    uint8_t  cmd_type;
    uint8_t  channel;
    uint8_t  num_pulses;
    uint16_t current_amplitude;
    uint16_t resting;
    uint16_t pulse_train_period;
    uint16_t phase_duration;
    uint16_t interphase_delay;
    uint16_t post_trigger_delay;
    uint16_t refractory_period;
    uint8_t  enable_amp_settle;
    uint16_t pre_amp_settle;
    uint16_t post_amp_settle;
    uint8_t  mantain_amp_settle;
    uint8_t  enable_charge_recovery;
    uint16_t charge_rec_on;
    uint16_t charge_rec_off;
    uint8_t  step_size;
} BleWaveCmd_t;
```

3.3.2 *main.c*

A significant change from the code of subsections 3.1.2 and 3.2.2 is the removal of the calibration forced reset procedure and the employment of flags to handle the repetitions of this part of the code. The *need_zcheck* variable is set to true when the impedance check has to be implemented, while *change_voltage_value* variable is still used to know if the voltage value corresponding to the desired current has been reached. If this is not the case the microcontroller is not reset, but the *while(1)* cycle in the code restarts from the beginning. An additional flag, named *change_waveform*, is added to allow the user to select a different waveform by pressing one of the buttons on the Nordic board during the transmission of the acquired samples. In this case, the calibration restarts using the new waveform parameters.

Another important modification is that BLE connection is never interrupted. Within the *run_intan_acquisition()* function, while samples are collected, the radio peripheral interrupts are temporarily disabled to satisfy the strict real-time constraints required by the SPI communication with the RHS chip. This ensures the CPU is not interrupted to handle incoming network packets. The data acquired

are now transmitted via BLE to the nRF52840 DK, by the function described in the next subsection.

Moreover, the memory is allocated only once at the beginning of the *main()* routine and the array is reset to 0 before both impedance check and acquisition.

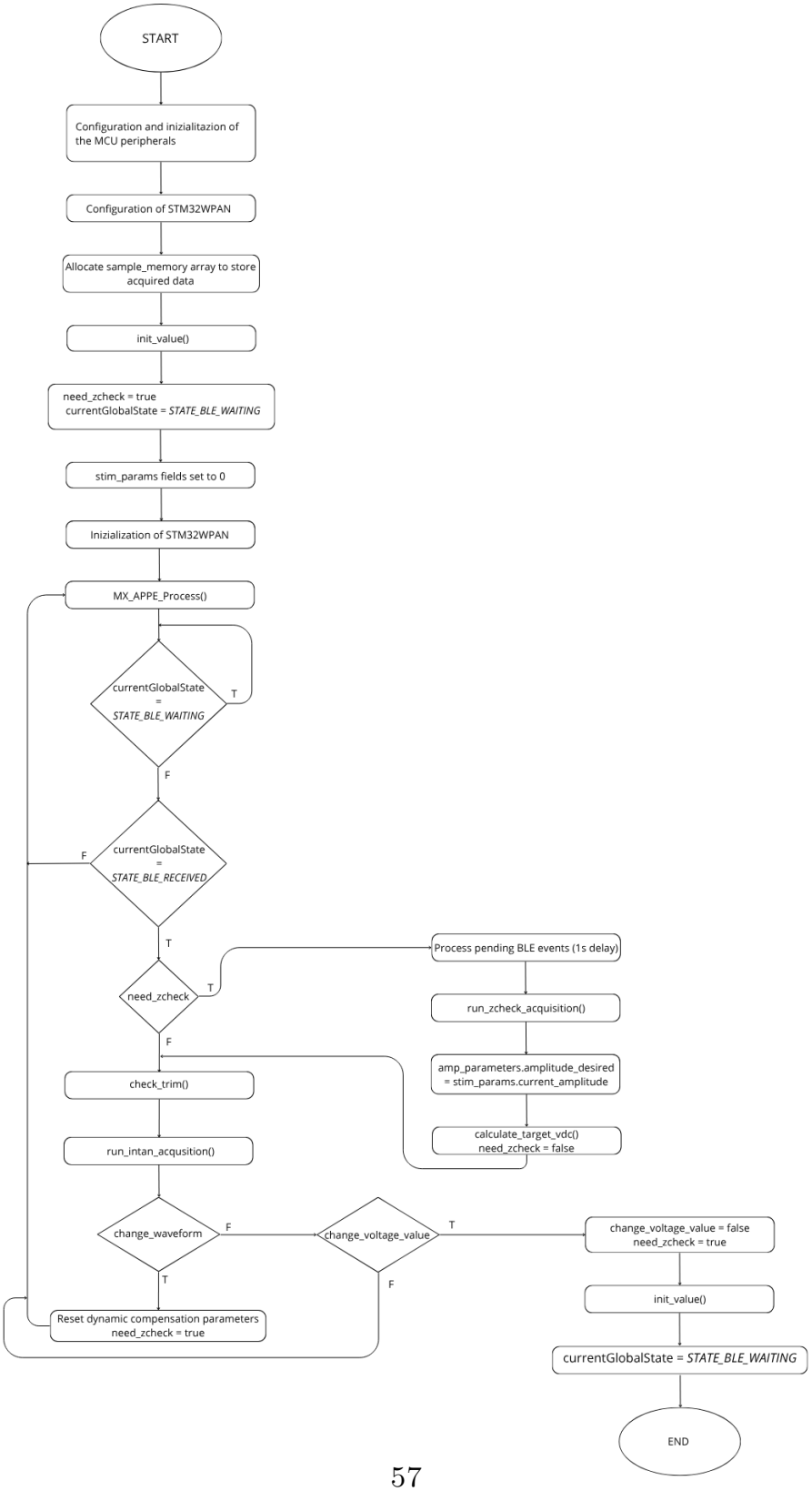


Figure 3.11: Flow chart of *main()* routine in *main.c* file.

3.3.3 *userfunctions.c*

A new function named *transmit_data_ble()* is called inside the *run_intan_acquisition()* procedure to transmit, via BLE, the samples acquired from two channels of the RHS2116 to the Nordic board, one of which is used for stimulation. The function computes the total number of data to send and divides the entire array into smaller chunks including a number of samples equal to BLE_SAMPLES_PER_CHUNK. The packets are prepared and then transmitted through the notification characteristic. The return value of the transmission function is checked to determine whether the packet has been successfully accepted. If not, the BLE processing function is executed to allow the radio buffer to be emptied before retrying. The function ends when the *matlab_ready* flag is set to true.

```
#define BLE_SAMPLES_PER_CHUNK 60

    void transmit_data_ble(void)
{
    // Compute the total number of samples acquired
    const uint32_t total_samples_in_memory = NUM_SAMPLED_CHANNELS *
    calculate_sample_rate() * NUMBER_OF_SECONDS_TO_ACQUIRE;
    uint32_t samples_sent = 0;

    // Data structure required from ST function
    P2P_SERVER_Data_t ble_payload_data;

    while (samples_sent < total_samples_in_memory) {
        uint32_t samples_left = total_samples_in_memory - samples_sent;
        uint16_t current_chunk_samples = (samples_left > BLE_SAMPLES_PER_CHUNK) ?
    BLE_SAMPLES_PER_CHUNK : samples_left;
        uint16_t chunk_bytes = current_chunk_samples * sizeof(uint32_t);

        // Packet created
        ble_payload_data.p_Payload = (uint8_t*)&sample_memory[samples_sent];
        ble_payload_data.Length = chunk_bytes;

        // Send the packet through SWITCH_C characteristic
        tBleStatus ret = P2P_SERVER_UpdateValue(P2P_SERVER_SWITCH_C, &
    ble_payload_data);

        if (ret == BLE_STATUS_SUCCESS) {
            // Succesfully sent
            samples_sent += current_chunk_samples;

            // Small pause to not instantly saturate the radio buffer
            HAL_Delay(3);
        } else {
            // Call the BLE process to empty the radio buffer and receive commands.
            MX_APPE_Process();
```

```
        }
    }

        // Reset the flag before waiting for MATLAB
        matlab_ready = false;

        // The BLE connection is kept active to prevent disconnection.
        while (!matlab_ready) {
            MX_APPE_Process();
        }
}
```

The number of samples per chunk is limited to a maximum of 60 due to the packet structure, shown in Fig. 3.12. Each sample is represented by a 32-bit word, with the 16 most significant bits corresponding to the AC value and the 16 least significant bits to the DC value.

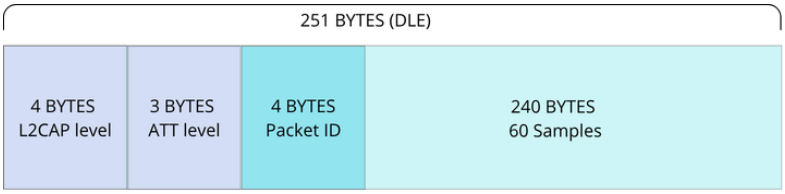


Figure 3.12: Structure of BLE packets sent by WBA65RI to nRF52840 DK.

3.3.4 SEGGER Embedded Studio - *main.c*

The code that runs on the nRF52840 DK is developed using SEGGER Embedded Studio starting from the project *ble_app_uart_c* of the software development kit for the nRF52 series.

The Nordic board plays the role of central, scanning for advertisement packets sent by the surrounding peripheral devices. The scan filter is modified to target the device name instead of the UUID. The *nrf_ble_scan* software module processes the Advertising Data searching for the *Complete Local Name* field. When an exact string match with the name "WBA_RHS" is detected, the central stops the scanning and sends a connection request to the NUCLEO WBA65RI.

```
    static void scan_init(void)
{
    ret_code_t          err_code;
    nrf_ble_scan_init_t init_scan;

    memset(&init_scan, 0, sizeof(init_scan));

```

```
    // ---- Force connection interval to 7.5 ms ----
    static ble_gap_conn_params_t conn_params;
    memset(&conn_params, 0, sizeof(conn_params));

    conn_params.min_conn_interval = MSEC_TO_UNITS(7.5, UNIT_1_25_MS);
    conn_params.max_conn_interval = MSEC_TO_UNITS(7.5, UNIT_1_25_MS);
    conn_params.slave_latency     = 0;
    conn_params.conn_sup_timeout  = MSEC_TO_UNITS(4000, UNIT_10_MS);

    init_scan.p_conn_param = &conn_params;
    // ------------------------------------------------------------

    init_scan.connect_if_match = true;
    init_scan.conn_cfg_tag     = APP_BLE_CONN_CFG_TAG;

    err_code = nrf_ble_scan_init(&m_scan, &init_scan, scan_evt_handler);
    APP_ERROR_CHECK(err_code);

    // Search for the name of STM32 device instead of UUID
    char const * target_name = "WBA_RHS";
    err_code = nrf_ble_scan_filter_set(&m_scan, SCAN_NAME_FILTER, target_name);
    APP_ERROR_CHECK(err_code);

    // Enable scan name filter
    err_code = nrf_ble_scan_filters_enable(&m_scan, NRF_BLE_SCAN_NAME_FILTER, false
    );
    APP_ERROR_CHECK(err_code);
}
```

Once the BLE connection is established, the connection parameters are configured to optimize data throughput. As shown in the previous code frame, a connection interval of 7.5 ms is requested. The use of the 2M PHY for both transmission and reception is also selected and handled in the PHY update request event to ensure that the requested PHY is applied when a PHY update procedure is triggered.

DLE is enabled in the *sdk_config.h* file to allow the transmission of 251-byte packets.

```
// <i> Requested BLE GAP data length to be negotiated.

#ifndef NRF_SDH_BLE_GAP_DATA_LENGTH
#define NRF_SDH_BLE_GAP_DATA_LENGTH 251
#endif
```

After these parameters have been configured, the nRF52840 DK performs aGATT service discovery to identify the custom service and its associated characteristics

used for data exchange. The UUIDs replicated in *ble_nus_c.h* file from the STM32CubeIDE *p2p_server.c* file, are used as search keys to find to find the custom Service and the specific channels for reading and writing data. To make the STM32 Base UUID compatible with the single root allocation of *ble_nus_c* module, the RX and TX characteristic UUIDs in the server firmware were modified to share the exact same 128-bit root as the primary service, differing only by their 16-bit identifiers (0xFE41 and 0xFE42).

```
1  //STM32CubeIDE-p2p_server.c
2  /* Private functions ---------------------------------------------------------*/
3  #define COPY_UUID_128(uuid_struct, uuid_15, uuid_14, uuid_13, uuid_12, uuid_11,
       uuid_10, uuid_9, uuid_8, uuid_7, uuid_6, uuid_5, uuid_4, uuid_3, uuid_2, uuid_1,
        uuid_0) \
4  do {\
5      uuid_struct[0] = uuid_0; uuid_struct[1] = uuid_1; uuid_struct[2] = uuid_2;
       uuid_struct[3] = uuid_3; \
6      uuid_struct[4] = uuid_4; uuid_struct[5] = uuid_5; uuid_struct[6] = uuid_6;
       uuid_struct[7] = uuid_7; \
7      uuid_struct[8] = uuid_8; uuid_struct[9] = uuid_9; uuid_struct[10] = uuid_10;
       uuid_struct[11] = uuid_11; \
8      uuid_struct[12] = uuid_12; uuid_struct[13] = uuid_13; uuid_struct[14] = uuid_14
       ; uuid_struct[15] = uuid_15; \
9  }while(0)
10
11 /*
12  The following 128bits UUIDs have been generated from the random UUID
13  generator:
14  0000FE40CC7A482A984A7F2ED5B3E58F: Service 128bits UUID
15  0000FE418E2245419D4C21EDAE82ED19: Characteristic 128bits UUID
16  0000FE428E2245419D4C21EDAE82ED19: Characteristic 128bits UUID
17  */
18 //#define COPY_P2P_SERVER_UUID(uuid_struct)       COPY_UUID_128(uuid_struct,0x00,0
       x00,0xfe,0x40,0xcc,0x7a,0x48,0x2a,0x98,0x4a,0x7f,0x2e,0xd5,0xb3,0xe5,0x8f)
19 //#define COPY_LED_C_UUID(uuid_struct)       COPY_UUID_128(uuid_struct,0x00,0x00,0
       xfe,0x41,0x8e,0x22,0x45,0x41,0x9d,0x4c,0x21,0xed,0xae,0x82,0xed,0x19)
20 //#define COPY_SWITCH_C_UUID(uuid_struct)       COPY_UUID_128(uuid_struct,0x00,0x00
       ,0xfe,0x42,0x8e,0x22,0x45,0x41,0x9d,0x4c,0x21,0xed,0xae,0x82,0xed,0x19)
21
22 /* USER CODE BEGIN PF */
23 // Modified UUID to share the same root
24 #define COPY_P2P_SERVER_UUID(uuid_struct)     COPY_UUID_128(uuid_struct,0x00,0x00,0
       xfe,0x40,0xcc,0x7a,0x48,0x2a,0x98,0x4a,0x7f,0x2e,0xd5,0xb3,0xe5,0x8f)
25 #define COPY_LED_C_UUID(uuid_struct)          COPY_UUID_128(uuid_struct,0x00,0x00,0
       xfe,0x41,0xcc,0x7a,0x48,0x2a,0x98,0x4a,0x7f,0x2e,0xd5,0xb3,0xe5,0x8f)
26 #define COPY_SWITCH_C_UUID(uuid_struct)       COPY_UUID_128(uuid_struct,0x00,0x00,0
       xfe,0x42,0xcc,0x7a,0x48,0x2a,0x98,0x4a,0x7f,0x2e,0xd5,0xb3,0xe5,0x8f)
27 /* USER CODE END PF */
```

```
//ble_nus_c.h

// STM32 WBA65RI UUID
#define NUS_BASE_UUID {{0x8F, 0xE5, 0xB3, 0xD5, 0x2E, 0x7F, 0x4A, 0x98, 0x2A, 0x48,
     0x7A, 0xCC, 0x00, 0x00, 0x00, 0x00}}
#define BLE_UUID_NUS_SERVICE           0xFE40 // P2P_SERVER Service
#define BLE_UUID_NUS_RX_CHARACTERISTIC 0xFE41 // Writing towards WBA
#define BLE_UUID_NUS_TX_CHARACTERISTIC 0xFE42 // Notifications from WBA
```

The Board Support Package (BSP) handler code is modified to have a specific biphasic waveform generated by the RHS when a specific button is pushed. After pressing the button, a structure of *BleWaveCmd_t* type is populated and sent as an array of bytes. The time required to perform the packet transmission request is measured using two functions from the Nordic Application Timer library.

```
    void bsp_event_handler(bsp_event_t event)
{
    ret_code_t err_code;

    switch (event)
    {
        case BSP_EVENT_SLEEP:
            nrf_pwr_mgmt_shutdown(NRF_PWR_MGMT_SHUTDOWN_GOTO_SYSOFF);
            break;

        case BSP_EVENT_DISCONNECT:
            err_code = sd_ble_gap_disconnect(m_ble_nus_c.conn_handle,
    BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;

        //Configuration of BUTTON 1
        case BSP_EVENT_KEY_0:
         if (m_ble_nus_c.conn_handle != BLE_CONN_HANDLE_INVALID) {
              BleWaveCmd_t biphasic1;
              biphasic1.cmd_type = 'W';
              biphasic1.channel = 3;
              biphasic1.num_pulses = 3;
              biphasic1.current_amplitude = 90;

              /*Code omitted: assign a value to all biphasic1 fields*/
              /*...*/

        //Record the transmission request time
```

```
32          uint32_t tx_start = app_timer_cnt_get();
33
34          // Save the function result in 'err_code'
35          uint32_t err_code = ble_nus_c_string_send(&m_ble_nus_c, (uint8_t *)&
    biphasic1, sizeof(BleWaveCmd_t));
36
37        //Transmission end time
38        uint32_t tx_end = app_timer_cnt_get();
39
40        uint32_t elapsed_ticks = app_timer_cnt_diff_compute(tx_end, tx_start);
41
42        float elapsed_ms = ((float)elapsed_ticks * 1000.0f) / 32768.0f;
43
44        uint32_t elapsed_ms_int = (uint32_t)elapsed_ms;
45        uint32_t elapsed_ms_dec = (uint32_t)((elapsed_ms - elapsed_ms_int) * 100);
46
47        NRF_LOG_INFO("Elapsed time: %d.%02d ms", elapsed_ms_int, elapsed_ms_dec);
48
49        if (err_code == NRF_SUCCESS) {
50          // Add the packet bytes to the counter
51          m_bytes_sent += sizeof(BleWaveCmd_t);
52      }
53
54          NRF_LOG_INFO("Struct size: %d byte", sizeof(BleWaveCmd_t));
55
56          if (err_code != NRF_SUCCESS) {
57              NRF_LOG_ERROR("Error in BLE sending: %d", err_code);
58          } else {
59              NRF_LOG_INFO("Waveform succesfully sent");
60          }
61      }
62            break;
63
64      // Configuration of BUTTON 2
65      case BSP_EVENT_KEY_1:
66
67          /*Code omitted: same structure of BUTTON 1 configuration*/
68          /*...*/
69
70          break;
71
72      default:
73          break;
74      }
75  }
```

In the BLE Nordic UART Service (NUS) client event handler a routine to forward the data received from the WBA65RI to MATLAB is added. A circular buffer is implemented to temporarily store the data received via BLE before forwarding them to MATLAB, preventing the BLE reception from being blocked by the UART transmission.

```
    static void ble_nus_c_evt_handler(ble_nus_c_t * p_ble_nus_c, ble_nus_c_evt_t
    const * p_ble_nus_evt)
{
    ret_code_t err_code;

    switch (p_ble_nus_evt->evt_type)
    {
        case BLE_NUS_C_EVT_DISCOVERY_COMPLETE:
        /*Code omitted*/
        break;

        case BLE_NUS_C_EVT_NUS_TX_EVT:
        {
            if (rx_packets == 0){
                // First packet
                rx_start_tick = app_timer_cnt_get();
            }
            rx_packets++;

            // Save the arrival time of this packet rx_last_packet_tick =
    app_timer_cnt_get();
            ble_bytes_received += p_ble_nus_evt->data_len;

            // Forwarding to MATLAB
            for (uint32_t i = 0; i < p_ble_nus_evt->data_len; i++){

            uint32_t next_write = (write_index + 1) % UART_BUFFER_SIZE;

            if (next_write != read_index)
            {
            uart_buffer[write_index] = p_ble_nus_evt->p_data[i];
            write_index = next_write;

            uart_bytes_received++;
            }
            else{
            uart_buffer_overflow++;
            }
        }
            break;
        }
          case BLE_NUS_C_EVT_DISCONNECTED:
              NRF_LOG_INFO("Disconnected.");
```

```
42                scan_start();
43                break;
44        }
45    }
```

The UART event handler waits for a message from MATLAB, indicating that data reception has been completed, and then computes the BLE communication throughput, sending the value obtained to the STM32 board. The value 32768 used for the calculation corresponds to the hardware frequency in Hz of the microcontroller's internal clock."

```
1  void uart_event_handle(app_uart_evt_t * p_event)
2  {
3      static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
4      static uint16_t index = 0;
5      uint32_t ret_val;
6
7      switch (p_event->evt_type)
8      {
9          /**@snippet [Handling data from UART] */
10         case APP_UART_DATA_READY:
11
12         UNUSED_VARIABLE(app_uart_get(&data_array[index]))
13         index++;
14
15         if (data_array[index - 1] == 'K')
16         {
17             matlab_ready = true;
18
19             NRF_LOG_INFO("K received from MATLAB");
20
21             uint32_t elapsed_ticks = app_timer_cnt_diff_compute(rx_last_packet_tick,
       rx_start_tick);
22
23             uint32_t elapsed_ms = (uint32_t)(((uint64_t)elapsed_ticks * 1000) /
      32768);
24
25             if (elapsed_ms > 0){
26             kbps = (ble_bytes_received * 8) / elapsed_ms;
27             }
28
29             char msg[32];
30
31             snprintf(msg, sizeof(msg),"BLE_TP:%lu\r\n", kbps);
32
33             ble_nus_c_string_send(&m_ble_nus_c,(uint8_t *)msg,strlen(msg));
34
```

```
            index = 0;

            // Reset the counters
            rx_packets = 0;
            ble_bytes_received = 0;
            uart_bytes_received = 0;
            uart_bytes_sent = 0;
            uart_buffer_overflow = 0;

            return;
            }
            /*Code omitted*/
        break;

        /*Code omitted*/
}
```

The main loop then continuously reads the buffered data and forwards them to MATLAB through the UART interface.

```
int main(void)
{
    /*Code omitted: Initialize and start execution*/

    // Enter main loop.
    for (;;)
    {
    idle_state_handle();

    uint32_t count = 0;

    while (read_index != write_index)
    {
        if (app_uart_put(uart_buffer[read_index]) == NRF_SUCCESS)
        {
            read_index = (read_index + 1) % UART_BUFFER_SIZE;
            uart_bytes_sent++;
            count++;
        }
        else
        {
            break;
        }
    }
  }
}
```

Chapter 4

Results

4.1 Acquisition

The acquisition trials were implemented through the setup shown in Fig. 2.5. The test waveforms employed were first sine waves with a frequency multiple of 50 Hz and then triangle waves with a frequency not multiple of 50 Hz in order to avoid to mistake the output waveform with the sinusoidal noise that an unconnected amplifier could pick up from surrounding mains. In Fig. 4.1 and Fig. 4.2 are shown two example plots of the samples acquired from the channel into which the test waveforms were injected. The expected output voltage amplitude is obtained by multiplying the peak to peak voltage (Vpp) set in the function generator by the division ratio k obtained from the following equation:

$$k = \frac{1.2\ \text{k}\Omega}{1.2\ \text{k}\Omega + 240\ \text{k}\Omega} = \frac{1}{201} \tag{4.1}$$

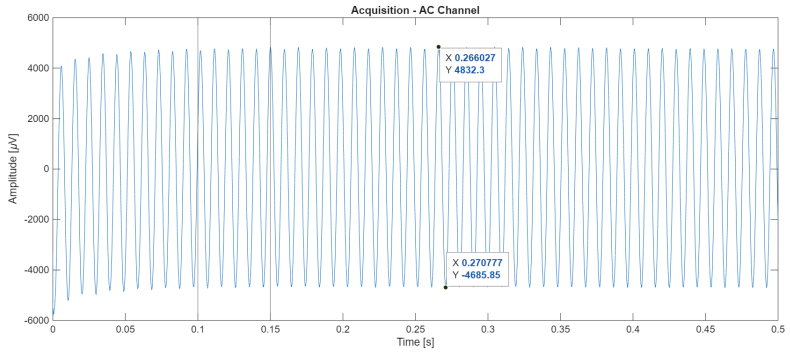


Figure 4.1: Sine wave. Function generator settings: 100 Hz, 2 Vpp. Expected amplitude: 9.95 mVpp. The two vertical lines attest the frequency is right.

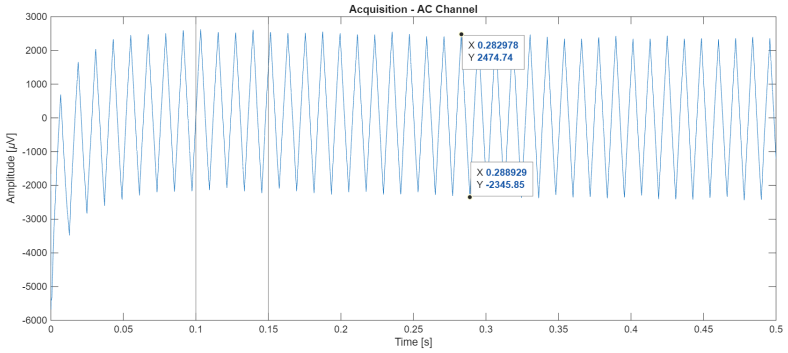


Figure 4.2: Triangle wave. Function generator settings: 80 Hz, 1 Vpp. Expected amplitude: 4.975 mVpp. The two vertical lines attest the frequency is right.

4.2 Calibration procedure

Default waveform evaluation

The first stimulation tests were employed with the setup of Fig. 2.5 selecting a Stimulation Step Size of 100 nA and connecting the channel on which the stimulation occurred to a 1 MΩ resistance. The default biphasic parameters provided by the Intan Firmware Framework and the waveform obtained in output are shown in Fig. 4.3. The expected phase duration of 5 ms, obtained by multiplying the value set in the firmware by the reciprocal of the sample frequency, was correctly obtained. The positive and negative phase amplitude were lower than the expected values of 1 V and -1 V, obtained by the equation 4.2, where *phase_amplitude* corresponds to *first_phase_amplitude* and *second_phase_amplitude* and R to the resistance value. Moreover, a spike at the beginning of the waveform can be noticed, related to the fact that resting amplitude is set to 0.

$$Amplitude[V] = phase_amplitude * StimStepSize * R \tag{4.2}$$

The *first_phase_amplitude* and the *second_phase_amplitude* values were then varied in order to obtain expected voltage amplitudes between 1 V and 5 V in output, with a step of 0.5 V, to assess if the error exhibited a linear trend across the different cases. The stimulation step size was changed to 50 nA. The mean of the acquired positive and negative phase samples and the corresponding percentage of error with respect to the target value are reported in Table 4.1.

```
BiphasicWaveform biphasic_example_ch8 = {
        .channel = 8,
        .trigger_source = 1,
        .num_pulses = 1,
        .pulse_train_period = 0,
        .resting_amplitude = 0,
        .first_phase_amplitude = 10,
        .second_phase_amplitude = -10,
        .post_trigger_delay = 0,
        .first_phase_duration = 100,
        .interphase_delay = 0,
        .second_phase_duration = 100,
        .refractory_period = 200,
        .enable_amp_settle = false,
        .pre_stim_amp_settle = 0,
        .post_stim_amp_settle = 0,
        .maintain_amp_settle = false,
        .enable_charge_recovery = false,
        .post_stim_charge_recovery_on = 0,
        .post_stim_charge_recovery_off = 0
};
```

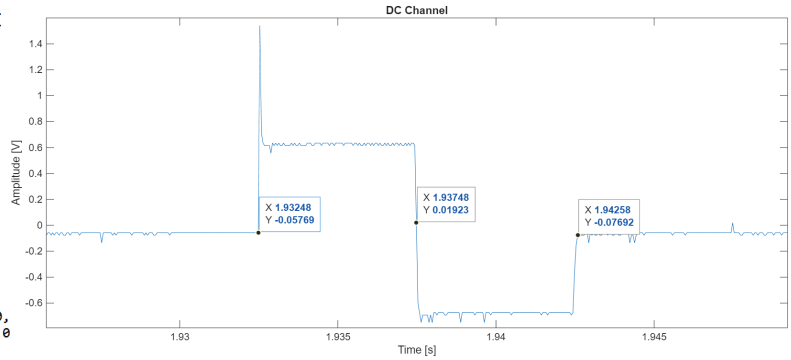


Figure 4.3: Biphasic parameters set in the firmware and corresponding biphasic waveform samples acquired plotted on MATLAB.

TARGET [V]	POS VALUE	POS PHASE MEAN [V]	ERR POS[%]
1	20	0,817	18,3
1,5	30	1,247	16,867
2	40	1,636	18,2
2,5	50	2,096	16,16
3	60	2,479	17,367
3,5	70	2,876	17,829
4	80	3,381	15,475
4,5	90	3,747	16,733
5	100	4,172	16,56
TARGET [V]	**NEG VALUE**	**NEG PHASE MEAN [V]**	**ERRE NEG[%]**
-1	-20	-0,769	23,1
-1,5	-30	-1,148	23,47
-2	-40	-1,52	24
-2,5	-50	-1,922	23,12
-3	-60	-2,287	23,767
-3,5	-70	-2,64	24,57
-4	-80	-3,075	23,125
-4,5	-90	-3,44	23,556
-5	-100	-3,825	23,5

Table 4.1: Values set in different trials for *first_phase_amplitude* (POS AMP) and *second_phase_amplitude* (NEG AMP) together with the corresponding voltage amplitude measured (POS PHASE MEAN and NEG PHASE MEAN) and the equivalent error (ERR POS and ERR NEG) with respect to the expected amplitude (TARGET).

Amplifier Fast Settle function was also enabled to clamp the output of the amplifier at baseline when each pulse occurred with the aim to reset the AC-coupled amplifier if a large stimulation artifact causes the output signals to saturate. In Fig. 4.4, the effect of enabling amplifier fast settle on AC data stream (AC channel) can be noticed.

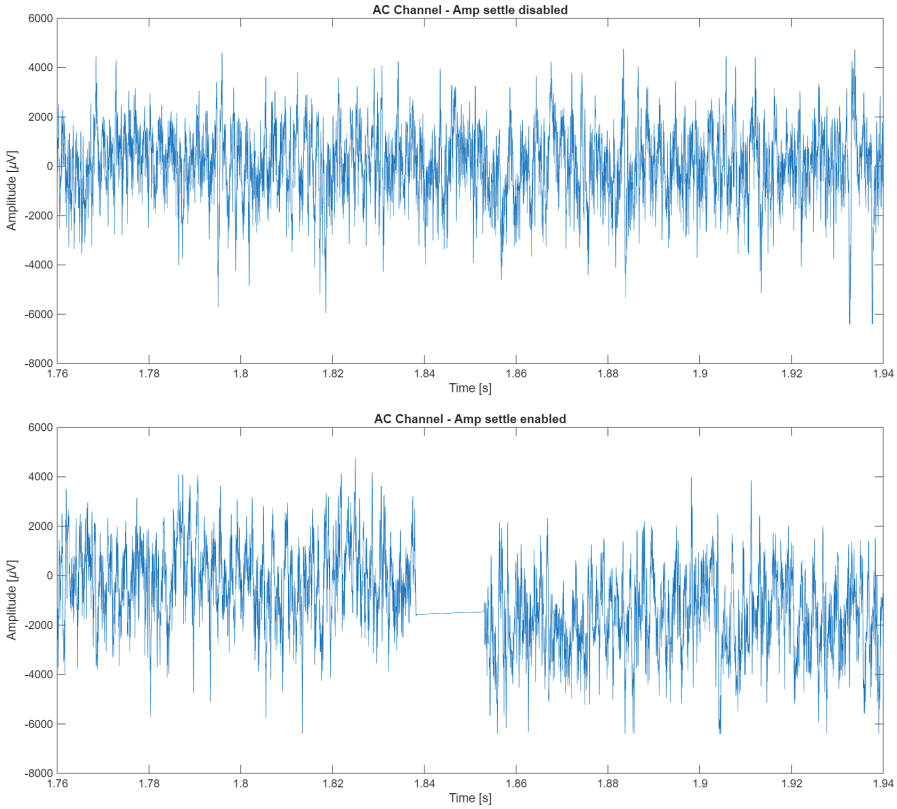


Figure 4.4: Effect of enabling amplifier fast settle on the AC channel.

In Fig. 4.3, the impact of enabling amplifier fast settle on the DC channel can be observed.

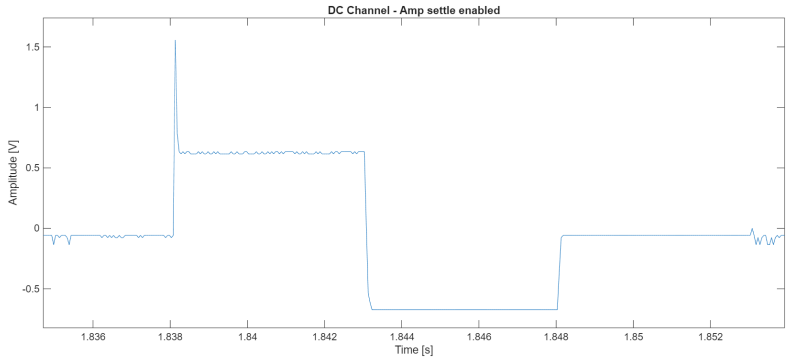


Figure 4.5: Effect of enabling amplifier fast settle on the DC channel.

Moreover, the number of pulses was increased to 3, the *resting_amplitude* value was set to 2 and the charge recovery switch was enabled to briefly connect an electrode to a common pin typically tied to ground, bleeding off residual charge after stimulation pulses. In Fig. 4.6 is highlighted the segment in which the signal amplitude is equal to 0.

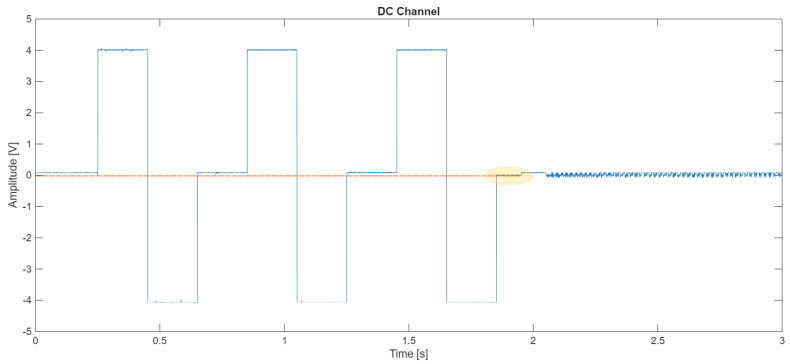


Figure 4.6: Charge recovery enabled at the end of stimulation.

Amplitude and Trim Correction

The first attempt to minimize the error percentage in stimulation waveform amplitude was to introduce a correction only through the trim value. The rationale behind the correction is the same explained in Fig. 3.5 and Fig. 3.6, but with a fixed value of *first_phase_amplitude* and *second_phase_amplitude*. The relation between trim value and error percentage is displayed in Fig. 4.7. For each target between 4 and 6 correction cycles were needed. Since the trim value was not fixed between different targets but fell in an almost wide range and moreover the maximum value of the negative trim was always reached without getting to the target, the relation between the current magnitude set in the firmware and the one measured was studied. At the *first_phase_amplitude* and *second_phase_amplitude* values were subsequently added respectively 2 and -2, equivalent to 0,1 V e -0,1 V, starting from a value of 20 and -20, which should correspond to an amplitude of 1 V and -1 V, until an effective amplitude of nearly 5 V was reached. The results obtained are plotted in Fig. 4.8. A linear relationship was found between the two parameters. The obtained calibration lines equations are used in the firmware to establish the first value to assign to *amp_parameters.mag_pos* and *amp_parameters.mag_neg*, as declared in section 3.1. To implement this approach, equation 4.3 must be satisfied, where R is the value of the resistance.

$$StimStepSize = \frac{0.1}{2R} \tag{4.3}$$

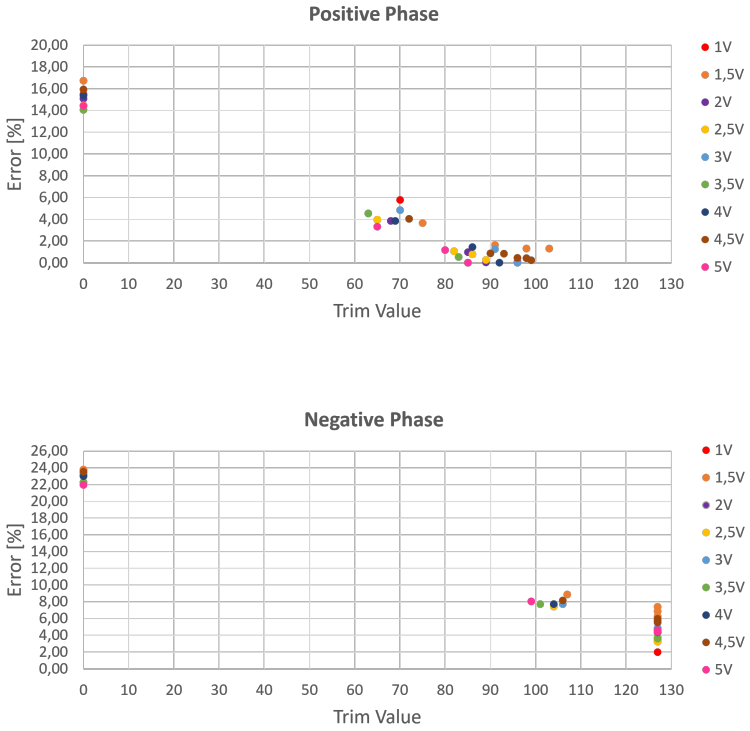


Figure 4.7: Reduction of error percentage depending on the trim value.

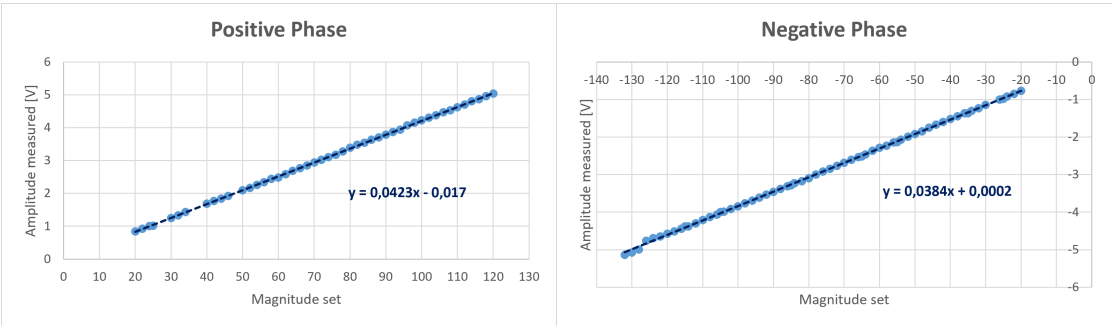


Figure 4.8: Calibration lines for positive and negative phase obtained from the correspondence between the magnitude values set in the firmware and the voltage amplitude measured.

The following amplitude correction procedure considers magnitude as the 'rough' dial to increase current, and trim as the 'fine' dial, as explained in section 3.1.

The first trial was conducted by connecting the same resistance of the previous tests, with an actual value of 1,0048 MΩ and maintaining a *StimStepSize* value of 50 nA. To verify the success of the procedure in a different case from the one considered, a 513 kΩ resistance was used and a *StimStepSize* of 100 nA was set. These parameters were kept constant for all subsequent tests. As can be observed from the tables in Fig. 4.9, in both examples considered at most two calibration cycles were needed to obtain an amplitude that differs in the worst case 2% from the one required. The threshold to stop calibration was then lowered to 0,01%.

Amplitude desired [V]		Magnitude set		Trim		Amplitude measured [V]		Error [%]	
Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative
1	-1	24	-26	0	0	1,001	-1	-0,1	0,00
2	-2	48	-52	0	0	2,018	-1,992	-0,9	0,40
3	-3	71	-78	0	0	2,999	-2,992	0,03	0,27
4	-4	95	-104	0	0	4,003	-3,989	-0,075	0,28
5	-5	119	-130	0	0	5,015	-5,074	-0,3	-1,48
5	-5	119	-130	0	-7	5,018	-4,997	-0,36	0,06

Amplitude desired [V]		Magnitude set		Trim		Amplitude measured [V]		Error [%]	
Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative
1	-1	24	-26	0	0	1,048	-1,056	-4,8	-5,60
1	-1	24	-26	-24	-28	1	-0,98	0	2,00
2	-2	48	-52	0	0	2,128	-2,071	-6,4	-3,55
2	-2	48	-52	-32	-17	1,999	-1,995	0,05	0,25
3	-3	71	-78	0	0	3,152	-3,135	-5,07	-4,50
3	-3	71	-78	-25	-22	2,942	-2,987	1,93	0,43
4	-4	95	-104	0	0	4,172	-4,156	-4,3	-3,90
4	-4	95	-104	-21	-19	4,002	-3,999	-0,05	0,02
5	-5	119	-130	0	0	5,24	-5,329	-4,8	-6,58
5	-5	119	-130	-24	-32	4,999	-4,997	0,02	0,06

Figure 4.9: The table on the top contains calibration parameters employing a resistance R = 1,0048 MΩ and *StimStepSize* = 50 nA. The table on the bottom contains calibration parameters employing R = 513 kΩ and *StimStepSize* = 100 nA.

Below is reported an example of parameters values set for the biphasic waveform, followed by the plot of the samples acquired from the channel on which the stimulation occurred in Fig. 4.10. The figure also highlights the compliance with the time parameters values set by the operator. In this instance the target amplitude was 3 V, therefore the initial values of *amp_parameters.mag_pos* and *amp_parameters.mag_neg* were respectively 71 and -78.

```
    BiphasicWaveform biphasic_example = {
            .channel = 3,
            .trigger_source = 1,
            .num_pulses = 3,
            .pulse_train_period = 12000,
            .resting_amplitude = 2,
            .first_phase_amplitude = amp_parameters.mag_pos,
            .second_phase_amplitude = amp_parameters.mag_neg,
            .post_trigger_delay = 20,
            .first_phase_duration = 4000,
            .interphase_delay = 0,
            .second_phase_duration = 4000,
            .refractory_period = 4000,
            .enable_amp_settle = true,
            .pre_stim_amp_settle = 20,
            .post_stim_amp_settle = 4000,
            .maintain_amp_settle = true,
            .enable_charge_recovery = true,
            .post_stim_charge_recovery_on = 0,
            .post_stim_charge_recovery_off = 2000
    };
```

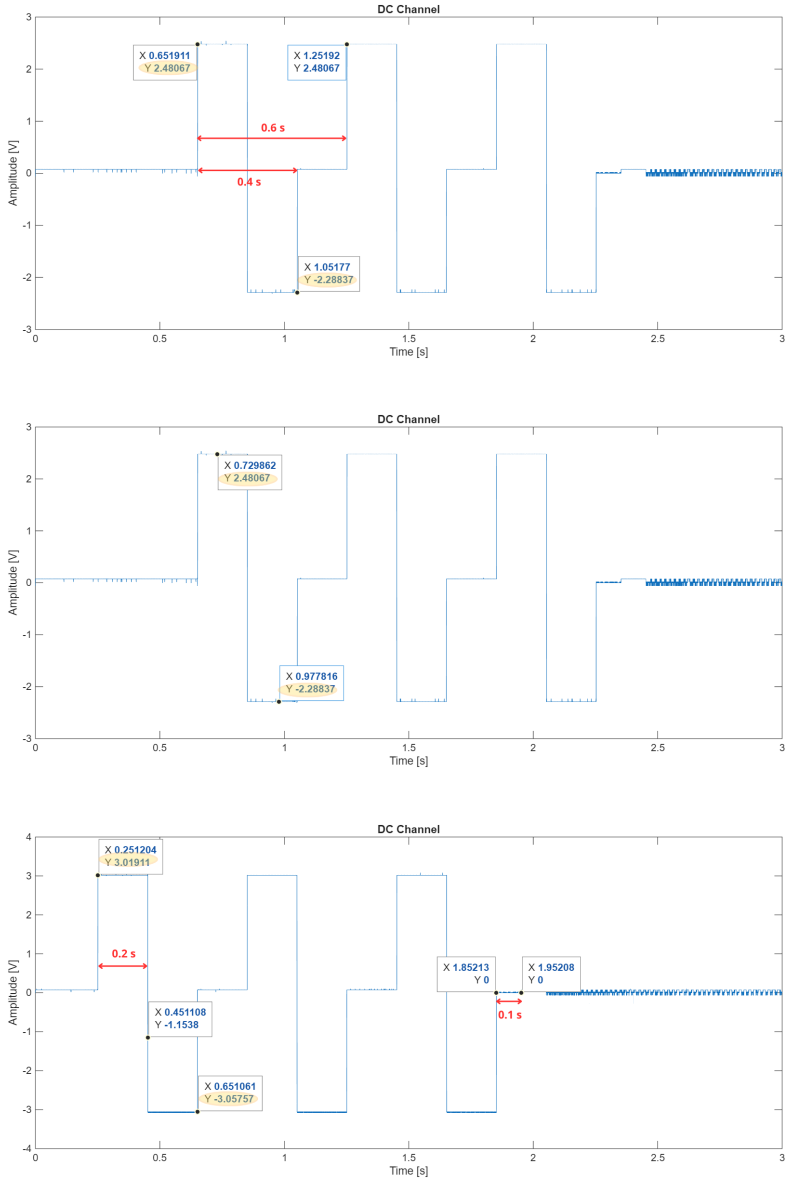


Figure 4.10: On the top is plotted the stimulation waveform corresponding to a target of 3 V before the correction procedure, in the middle an intermediate stage, on the bottom is plotted the waveform after applying the correction.

4.3 BLE Communication

The proper functioning of the BLE communication between the NUCLEO WBA65RI and the STBLEToolbox application, as described in section 3.2, was tested by selecting each option in the menu and inserting a value for the corresponding parameter, where required. In Fig. 4.12 is reported an instance of values entered in the appropriate field of the application and the corresponding printing on MATLAB Command Window. As can be observed, the blue led turns on when the user has to insert a value and turns off when the value has been entered.

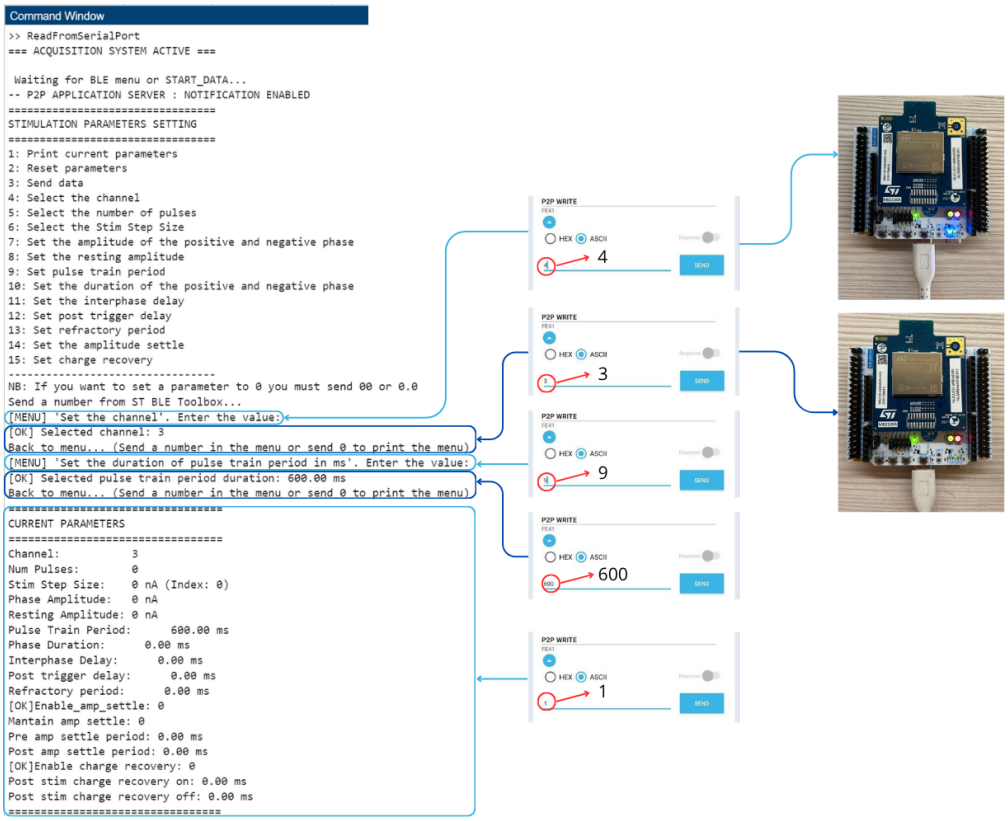


Figure 4.11: BLE communication between NUCLEO WBA65RI and STBLEToolbox application.

When the option '3' was selected, the application disconnected from the NUCLEO board and the impedance check started, followed by the calibration procedure.

Introduction of nRF52840 DK

To evaluate BLE communication between the NUCLEO WBA65RI and the Nordic nRF52840 DK in both directions, explained in section 3.3, several tests were run. To test the data exchange from the Nordic development kit to the NUCLEO board, Button 1 or Button 2 on the Nordic board was pushed and then the printings on MATLAB command window were checked to verify that the waveform parameters have been correctly received by the NUCLEO WBA65RI.

The execution time of the BLE transmission function for the 29-byte parameter structure was approximately 0.03 ms.

After data transmission from the nRF52840 DK to the STM32 board, impedance check was initiated, followed by acquisition from two channels, while simultaneously delivering stimulation through one of them or another.

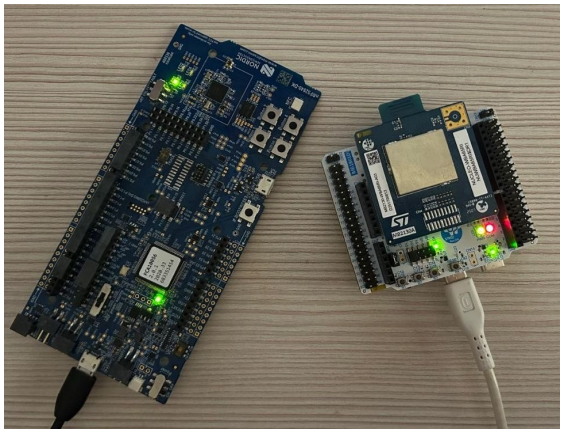


Figure 4.12: BLE communication between NUCLEO WBA65RI and nRF52840 DK.

After these trials, BLE transmission from the WBA65RI to the nRF52840 DK was evaluated, replacing the transmission via UART to MATLAB with transmission via BLE to the Nordic board, which in turn forwarded the data to MATLAB for visual inspection of the received samples. The delay between consecutive BLE packet transmissions, set in the *transmit_data_ble()* function, was changed to find the minimum value that allowed the correct receipt of all the samples forwarded from the nRF52840 DK to MATLAB.

With no delay, the BLE throughput measured from the Nordic board to the NUCLEO board was approximately 1500 kbps. However, some samples were lost during the UART transmission, resulting in meaningless results in MATLAB.

The minimum delay value that ensured a successful transfer of all samples to MATLAB was 3 ms. In this case the throughput measured was around 950 kbps, while the recorded UART throughput was about 200 kbps.

4.4 Discussion of the results

As can be seen in Fig. 4.1 and Fig. 4.2, a signal within the range of ± 5 mV that emulates the electrophysiological one can be correctly acquired with the setup implemented.

Regarding stimulation, the first tests described in Section 4.2, in which the default biphasic stimulation waveform provided by the Intan Firmware Framework was employed, showed full compliance with the specified time parameters in the recorded samples, but a significant error in the measured amplitude. As indicated in Table 4.1 the percentage of error was around 16% for the positive phase and around 23% for the negative phase.

Moreover, a spike was detected at the beginning of the pulse, coinciding with a null resting amplitude. As explained by Intan Technologies, this is a common and unavoidable hardware effect that occurs when the stimulators are powered on. Therefore, a non-zero resting amplitude was used in the following tests to mitigate this effect. However, the potential implications of introducing a non-zero resting current, particularly in terms of charge balance, should be carefully considered.

Furthermore, enabling amplifier fast settle enhances the signal quality on DC channel reducing the small continuous variations during the pulse, as can be observed by comparing Fig. 4.5 with Fig. 4.3.

The developed calibration procedure, starting from the value obtained from the equations derived from the linear fits shown in Fig. 4.8 and combining amplitude and trim correction, proved effective in achieving a voltage amplitude differing by no more than 0.01% from the value corresponding to the required current, as demonstrated in Fig. 4.10. An average of 3-4 cycles was required to reach the target. In contrast, the trim correction alone was found to be ineffective.

Wireless communication via BLE was successfully established first between the NUCLEO WBA65RI and the STBLEToolbox application and then between the NUCLEO board and the nRF52849 DK. In both cases, the waveform obtained corresponded to the parameters sent and the calibration was efficiently accomplished.

The BLE transmission of the 29-byte stimulation waveform parameters from the Nordic board to the NUCLEO board required 0.03 ms, while the BLE throughput achieved in the opposite direction, while ensuring the transfer of all samples to MATLAB, was approximately 950 kbps.

Chapter 5

Conclusion

This work led to the development of custom firmware in STM32CubeIDE capable of automatically tuning the current amplitude parameters of a biphasic stimulation waveform. The implemented closed-loop system relies on voltage samples acquired through the Intan RHS2116 headstage to accurately reach a user-defined target. The NUCLEO WBA65RI (simulating the implantable module) successfully controls the Intan microchip via SPI to simultaneously perform neural stimulation and data acquisition.

Furthermore, a reliable BLE communication link can be established with an nRF52840 DK (acting as the external mechatronic structure). The wireless interface allows the initial transmission of the required biphasic parameters and also enables the user to dynamically update them during data streaming at a rate of 950 kbps from the Nordic DK to the NUCLEO.

The calibration routine iteratively corrects the parameters—or resets the calibration state to restart the process upon receiving new waveform configurations—and successfully terminates only when the measured voltage converges to within 0.01% of the required target. Once the calibration is completed, the NUCLEO WBA65RI returns to the waiting state, ready to receive a new waveform configuration.

In Fig. 5.1 a diagram summarizing the whole process is shown.

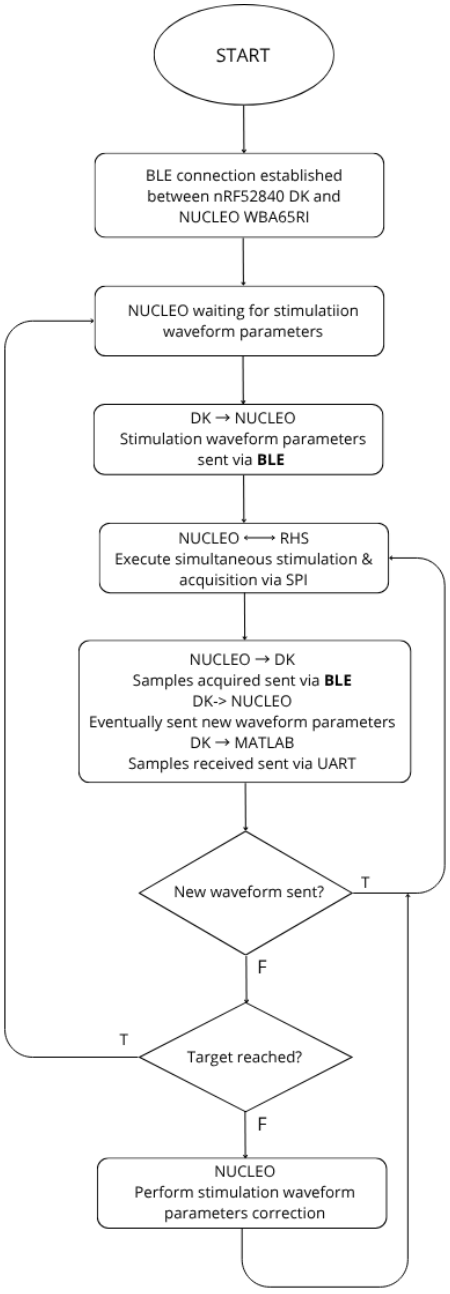


Figure 5.1: Summary diagram of the project

5.1 Limitations

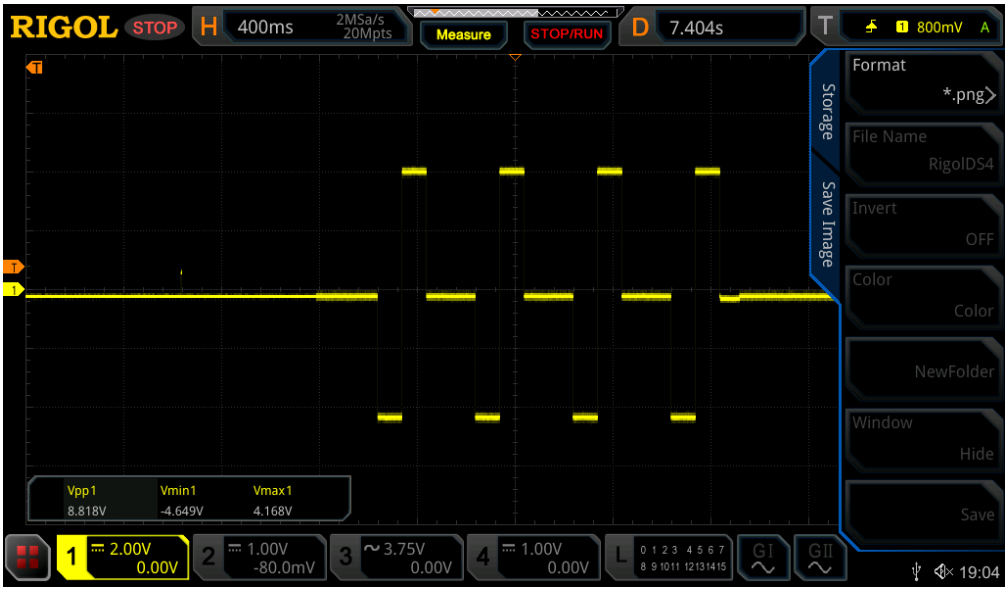
A requirement for the calibration performance obtained is given by Eq. 4.3. According to this equation, the stimulation step size value depends on the resistance connected to the channel on which the stimulation occurs and must therefore be evaluated in advance. In a real application, an estimation of the load impedance value could be necessary to find a stimulation step size suitable both for the current target value and for satisfying the equation. If the relationship 4.3 is not met, the correction step size employed in the calibration procedure may correspond to a voltage value different from 0.1 V, and the calibration could require a different number of cycles or may not be effective.

Another limitation is the low accuracy of the RHS DC amplifiers, which can detect an absolute minimum difference of 19.2 mV, variable because of inherent or external noise, or asymmetry between ADC steps. ntan Technologies reported that, experimentally, a typical error of around 25% was observed in the DC amplifiers.

In this project, a comparison between the waveforms plotted in MATLAB and those measured using the oscilloscope revealed a difference of approximately 0.4 V in the negative phase of the biphasic waveform, while the voltage of the positive phase was almost identical in the two cases. Some tests were therefore conducted using two separate target values for the positive and negative phases, with 0.4 V subtracted from the absolute target value of the negative phase. An example of the voltage reported by the Intan DC amplifier and that measured by the oscilloscope before applying the aforementioned correction the improvement resulting from this correction can be observed in Fig. 5.2.

```
trim_pos = 17, trim_neg = -26
peak_pos = 4120, peak_neg = -4103
mag_pos = 101, mag_neg = -107
amplitude desired = 4.10 V
```

```
trim_pos = 20, trim_neg = 22
peak_pos = 4076, peak_neg = -3677
mag_pos = 101, mag_neg = -88
amplitude desired = 4.10 V
```



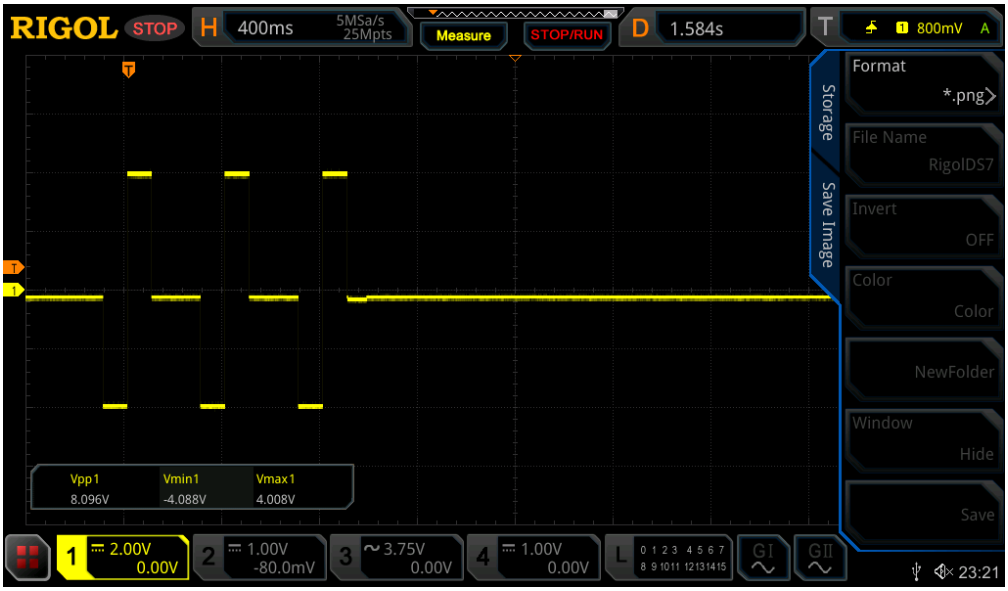


Figure 5.2: At the top, the positive (*peak_pos*) and negative (*peak_neg*) phase values recorded by the Intan DC amplifier are displayed. No correction is applied in the left plot, whereas the correction is applied in the right plot. At the bottom, the oscilloscope screen is reported first for the case without correction and then for the case with correction.

5.2 Possible enhancements

In this section, insights for a future development and improvement of this work are presented.

Greater flexibility in the selection and independent updating of stimulation waveform parameters, as well as in enabling or disabling stimulation, could be provided to the user by creating a menu similar to the one implemented in the ST BLE Toolbox application, but integrated into the Nordic board. The impedance check should also be improved due to a lack of precision in the measured data.

Another significant improvement would be the introduction of real-time acquisition, allowing the stimulation parameters to be updated simultaneously. The number of acquisition and stimulation channels should also be increased to meet the requirements of the NerveRepack project. To achieve these objectives, several considerations must be made regarding whether the microcontrollers used are suitable for the required performance. Managing continuous high-throughput SPI DMA transfers, executing closed-loop control algorithms, and servicing time-critical radio interrupts on a single-core architecture would inevitably lead to buffer overruns and packet loss.

Bibliography

[1] National Academies of Sciences, Engineering, and Medicine; Health and Medicine Division; Board on Health Care Services; Committee on the Use of Selected Assistive Products and Technologies in Eliminating or Reducing the Effects of Impairments. «Upper-Extremity Prostheses». In: *The Promise of Assistive Technology to Enhance Activity and Work Participation*. Ed. by J. L. Flaubert, C. M. Spicer, and A. M. Jette. Available from: https://www.ncbi.nlm.nih.gov/books/NBK453290/. Washington (DC): National Academies Press (US), May 2017. Chap. 4. URL: https://www.ncbi.nlm.nih.gov/books/NBK453290/ (cit. on p. 2).

[2] Aimee E. Schultz and Todd A. Kuiken. «Neural Interfaces for Control of Upper Limb Prostheses: The State of the Art and Future Possibilities». In: *PM&R* 3.1 (2011), pp. 55–67. DOI: https://doi.org/10.1016/j.pmrj.2010.06.016. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1016/j.pmrj.2010.06.016. URL: https://onlinelibrary.wiley.com/doi/abs/10.1016/j.pmrj.2010.06.016 (cit. on p. 2).

[3] Ho Shing Lo and Sheng Quan Xie. «Exoskeleton robots for upper-limb rehabilitation: State of the art and future prospects». In: *Medical Engineering Physics* 34.3 (2012), pp. 261–268. ISSN: 1350-4533. DOI: https://doi.org/10.1016/j.medengphy.2011.10.004. URL: https://www.sciencedirect.com/science/article/pii/S1350453311002694 (cit. on p. 3).

[4] IEEE Xplore. «A Review on Neural Interfaces and Neuromodulation Systems». In: *IEEE Access* (2024). IEEE Document No. 10601179. DOI: 10.1109/ACCESS.2024.10601179. URL: https://ieeexplore.ieee.org/document/10601179 (cit. on p. 3).

[5] Xavier Navarro, Thilo B. Krueger, Natalia Lago, Silvestro Micera, Thomas Stieglitz, and Paolo Dario. «A critical review of interfaces with the peripheral nervous system for the control of neuroprostheses and hybrid bionic systems». In: *Journal of the Peripheral Nervous System* 10.3 (Sept. 2005), pp. 229–258. DOI: 10.1111/j.1085-9489.2005.10303.x (cit. on p. 4).

[6] D. R. Merrill, M. Bikson, and J. G. R. Jefferys. «Electrical stimulation of excitable tissue: design of efficacious and safe protocols». In: *Journal of Neuroscience Methods* 141.2 (Feb. 2005), pp. 171–198. DOI: 10.1016/j.jneumeth.2004.10.020 (cit. on pp. 4, 5).

[7] Stuart F. Cogan. «Neural stimulation and recording electrodes.» In: *Annual review of biomedical engineering* (2008). DOI: 10.1146/ANNUREV.BIOENG.10.061807.160518 (cit. on p. 5).

[8] Cameron C. McIntyre and Warren M. Grill. «Extracellular stimulation of central neurons: influence of stimulus waveform and frequency on neuronal output». In: *Journal of Neurophysiology* 88.4 (Oct. 2002), pp. 1592–1604. DOI: 10.1152/jn.2002.88.4.1592 (cit. on p. 5).

[9] Stuart F. Cogan, Philip R. Troyk, J. Ehrlich, Timothy D. Plante, and D. E. Detlefsen. «Potential-biased, asymmetric waveforms for charge-injection with activated iridium oxide (AIROF) neural stimulation electrodes». In: *IEEE Transactions on Biomedical Engineering* 53.2 (Feb. 2006), pp. 327–332. DOI: 10.1109/TBME.2005.862531 (cit. on p. 5).

[10] W. F. Agnew and D. B. McCreery. «Considerations for safety with chronically implanted nerve electrodes». In: *Epilepsia* 31.s2 (1990), s27–s32. DOI: 10.1111/j.1528-1157.1990.tb05856.x (cit. on p. 5).

[11] Dustin W. Tan, Matthew A. Schiefer, Michael W. Keith, J. Robert Anderson, Jennifer Tyler, and Dustin J. Tyler. «A neural interface provides long-term stable natural touch perception». In: *Science Translational Medicine* 6.257 (Oct. 2014), 257ra138. DOI: 10.1126/scitranslmed.3008669 (cit. on pp. 8, 9).

[12] Francesco Clemente et al. «Intraneural sensory feedback restores grip force control and motor coordination while using a prosthetic hand». In: *Journal of Neural Engineering* 16.2 (Feb. 2019), p. 026034. DOI: 10.1088/1741-2552/ab059b. URL: https://doi.org/10.1088/1741-2552/ab059b (cit. on pp. 9, 10).

[13] Shinnosuke Idogawa, Koji Yamashita, Rioki Sanda, Rika Numano, Kowa Koida, and Takeshi Kawano. «A lightweight, wireless Bluetooth-low-energy neuronal recording system for mice». In: *Sensors and Actuators B: Chemical* 330 (2021), p. 129381. DOI: 10.1016/j.snb.2020.129381 (cit. on p. 11).

[14] Dai Jiang, Fangqi Liu, Henry T. Lancashire, Timothy A. Perkins, Matthew Schormans, Anne Vanhoestenberghe, Nicholas De N. Donaldson, and Andreas Demosthenous. «A Versatile Hermetically Sealed Microelectronic Implant for Peripheral Nerve Stimulation Applications». In: *Frontiers in Neuroscience* 15 (2021), p. 681021. DOI: 10.1016/10.3389/fnins.2021.681021 (cit. on p. 12).

[15] *NerveRepack-Context and Innovation*. URL: https://www.nerverepack.eu/copy-context-innovation (cit. on p. 13).

[16] Intan Technologies. *RHS2116 stim/amplifier chip*. URL: https://intantech.com/products_RHS2000.html (cit. on p. 15).

[17] Intan Technologies. *LVDS Adapter Board*. URL: https://intantech.com/LVDS_adapter_board.htm (cit. on p. 16).

[18] Intan Technologies. *18-pin Electrode Adapter Board*. URL: https://intantech.com/electrode_adapters.htm (cit. on p. 16).

[19] STMicroelectronics. *NUCLEO-H723ZG product overview*. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32h723zg.html (cit. on p. 17).

[20] STMicroelectronics. *NUCLEO-WBA65RI product overview*. URL: https://www.st.com/en/evaluation-tools/nucleo-wba65ri.html (cit. on p. 17).

[21] NORDIC SEMICONDUCTOR. *nRF52840 DK*. URL: https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK (cit. on p. 17).

[22] RIGOL. *MSO5104*. URL: https://rigolshop.eu/product-oscilloscope-mso5000-mso5104.html (cit. on p. 18).

[23] RIGOL. *DP832A*. URL: https://rigolshop.eu/products/power-supply/dp800/dp832a.html (cit. on p. 18).

[24] SGLabs. *HP Agilent 33120A*. URL: https://www.sglabs.it/en/product.php?s=hp-hewlett-packard-agilent-33120a&id=8 (cit. on p. 18).

[25] STMicroelectronics. *STM32CubeIDE*. URL: https://www.st.com/en/development-tools/stm32cubeide.html (cit. on p. 19).

[26] STMicroelectronics. *STBLEToolbox overview*. URL: https://www.st.com/en/embedded-software/stbletoolbox.html (cit. on p. 19).

[27] MathWorks. *MATLAB overview*. URL: https://www.mathworks.com/products/matlab.html (cit. on p. 19).

[28] SEGGER. *SEGGER Embedded Studio overview*. URL: https://www.segger.com/products/development-tools/embedded-studio/ (cit. on p. 20).

[29] *RHD/RHS STM32 Firmware Framework*. URL: https://intantech.com/downloads.html?tabSelect=Firmware&yPos=141.3333282470703 (cit. on p. 20).

[30] *RHS2116 digital stimulator/amplifier chip datasheet*. URL: https://intantech.com/files/Intan_RHS2116_datasheet.pdf (cit. on pp. 36, 37, 49).

[31] *STM32WBA Bluetooth® LE - Peer 2 Peer Applications*. URL: https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32WBA_Peer_To_Peer#STM32WBA_Peer_to_Peer_Server_application (cit. on p. 40).

[32] *Principles of lock-in detection*. URL: https://cms.zhinst.com/sites/default/files/documents/2025-10/zi_whitepaper_principles_of_lock-in_detection.pdf (cit. on p. 49).