UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Mobilizing Pain Research using Bluetooth Low Energy

Pain at the tips of your fingers

Master's Thesis
6th of July 2023

Author
ing. H. Slegers [BSS]

Committee

Chair:	dr. ir. J.R. Buitenweg [BSS]
Supervisor:	ir. F. Muijzer [BSS]
Extern:	dr. ir. R.A.R. van der Zee [ICD]

BSS	Biological Signals and Systems
ICD	Integrated Circuit Design

Summary

The NociTRACK system is a tool currently used in pain research in order to analyse the response of humans to electric stimuli at the lowest possible boundary of their pain perception. The system is under active development at the University of Twente, and it is intended to be used at a large scale to allow for analysis of a wide range of subjects, ranging from healthy people to those with chronic pain or other problems with pain sensations. In order to perform these tests at scale, the system requires that tests can be performed using a laptop or tablet at the general practitioner or even at home.

Bluetooth Low Energy provides a wireless protocol which allows multiple peripherals to connect simultaneously, and reduces the energy usage of wireless connections considerably. Additionally many mobile devices provide better support for Bluetooth Low Energy. As the NociTRACK system currently only supports Bluetooth Classic, this system must be overhauled to remain relevant to a wide range of devices and experiments in the foreseeable future.

At the same time, the applications used to run the experiments are only available on desktop computers. Many physicians, especially those that visit clients, prefer to use a mobile devices or tablets, as they are considerably more portable. Researchers and engineers on the other hand, prefer the desktop environment which is more versatile to develop net experiments and features. For these reasons, an application was developed that can be built to run both on a laptop and a mobile device.

To this end, a new Bluetooth module was needed on an existing stimulator, and based on the previous version, a new protocol was designed for with Bluetooth Low Energy. An application was conceptualized which can be used to perform nociceptive experiments by a broad range of operators on a broad range of devices. As developing the entire application was not feasible within the allotted time, a version was developed which can be used to communicate with stimulators through the Bluetooth Low Energy protocol. The application is also designed to provide a basis upon which future developers can add functionality, such as providing predetermined tests to perform at scale, as well as a way for researchers to develop new testing methodologies.

Test results indicate that the newly developed wireless protocol is able to fulfill the same functionality as its previous iteration, though performance is hindered through several factors, such as low-level configuration of the Bluetooth connection parameters. For tests not demanding fast performance, the current implementation suffices. The implementation of the application is not cross-platform due to fragmented support for Bluetooth Low Energy across platforms, though care has been taken such that creating an implementation for Android or iOS devices is quite feasible. The current application also successfully provides a workable basis upon which a fully featured test platform can be developed.

Based on the results, recommendation have been given for future step to improve the implementations, and what aspects of the system should be developed next in the process.

Acronyms

NRS	Numeric Rating Scale
VAS	Visual Analogue Scale
EEG	Electroencephalography
QST	Quantitative Sensory Testing
MTT	Multiple Threshold Tracking
BLE	Bluetooth Low Energy
SDK	Software Development Kit
GATT	Generic Attribute Profile
UUID	Universally Unique Identifier
MTU	Maximum Transmissible Unit
PDU	Protocol Data Unit
ISI	Inter Frame Space
UART	Universal asynchronous receiver/transmitter
RTT	Round-trip time
JSON	JavaScript Object Notation

Contents

Chapter 1

Introduction

This thesis will pertain to the conceptualization and development of a new iteration of the NociTRACK system, in order to scale up and include more subjects in nociceptive research, e.g. into chronic neuropathic pain.

1.1 NociTRACK

The NociTRACK system is a tool developed at the University of Twente in order to carry out psychophysical experiments on the nociceptive nervous system.

These experiments involve strapping stimulation electrodes to a patch of skin on the subject, and recording the response from the subject. This response can be in the form of a button press, EEG measurements, or whatever other sensors wish to be used.

This system has been used for research aiming to design an algorithm that can determine the detection threshold of the nociceptive nerves. In this algorithm the stimulus current is gradually increased until the subject can feel anything from the stimulus at all, not necessarily inflicting pain. These experiments are valuable for the general understanding of the processing of nociceptive stimuli, as well as identifying abnormal pain sensations such as allodynia, where someone feels pain from a sensation that should not be, or hyperalgesia, where pain sensations are amplified. These kinds of pain-threshold-tracking tests find many uses in identifying and characterising chronic pain in a quantitative way, as currently patient pain levels are self-reported through some kind of numeric or visual scale, which has many other influences from anything including the emotional state, to attention span, to general anxiety.

1.2 Moving research out of the lab

While the current version of the NociTRACK system has proved to be effective at performing threshold tracking experiments, these experiments are still bound to research lab and hospital environments. This means that testing is limited to subjects with the movement freedom to come to the research facilities, and requires much time and resources from the research team to carry out the experiments. A preferable system solution is usable at the point-of-care so that subjects can go to their usual physician, or at their own home by an at-home physician. This would greatly extend the reach of such experiments, allowing for the collection of far greater amounts of, and variations in, research data.

1.3 Limitations of the NociTRACK System

In order to scale up testing in pain research, the current implementation must overcome several hurdles. These are identified as follows:

Firstly, the software used by the system depends is built upon LabVIEW, which for research purposes is a good way to set up a program with a user interface that can be used for research experiments and easily gather and process data. However as experiments and programs grew larger and more complex, the visual programming style became unwieldy and creating new parts largely involved figuring out why what connections went where, and how to make an intuitive interface for a less intuitive system. In addition, the applications created in LabVIEW are exclusively available for personal computers, while for portable applications, mobile phones and tablets are a more fitting platform. For these reasons, it is imperative to move the development of the application to a more professional and cross-platform framework, before the application is distributed and testing capacity can effectively be scaled up.

Secondly, the protocol used to communicate between computers and stimulator hardware is named StimCom. In the current iteration of StimCom, the wireless connection is made using Bluetooth 2. This version is based on older Bluetooth Classic technology. Newer portable devices such as tables, mobile phones, and laptops have moved to newer versions of Bluetooth (4.0 and up), which are based on Bluetooth Low Energy (BLE). While Bluetooth Classic is supported by a decent range of products, this support is slowly dwindling. Supporting Bluetooth 4 or higher will guarantee that modern devices are able to connect to the NociTRACK system. In addition, Bluetooth Classic is a peer-to-peer technology, meaning networks that require multiple stimulators, or communication between stimulators are limited. Bluetooth Low Energy solves this issue by providing support for more complex network structures, such as mesh networks [1]. Therefore, in order to support all necessary devices for the application, stimulators must also be adapted to support BLE.

Thirdly, experiments carried out take a long time, sometimes up to two hours. Some experiments are simple reporting tasks, such as reporting whether a stimulus was detected. Tasks such as these, combined with lengthy or slowly converging threshold tracking algorithms can reduce the quality of the experiment. Maintaining focus becomes difficult, and the mind of the subject will wander more and more frequently the longer an experiment takes, causing noise and inconsistencies in the data. In addition, an experiment that takes a long time is more difficult to fit into the schedules of already busy physicians, while also increasing the hiring cost. Reducing the test time by improving the convergence rate of the algorithm is therefore a priority.

Lastly, the experiments are complex, and require educated researchers to configure and execute the application correctly. If the interface can be made in a way that experiments can be run without extensive knowledge of the background and technology of the system, the experiments can be carried out by more generally trained physicians.

This thesis will focus on the first and second issue, aiming to provide a protocol that transfers the functionality of StimCom to Bluetooth Low Energy, and implementing the protocol on the stimulator and application. It will also provide a design for an application design that is supported by as many devices as possible, and is extensible for future additions.

Chapter 2

Background

The background will briefly expand on the subject on pain research, as well as provide a very short overview of the functionality of Bluetooth Low Energy.

2.1 Research into Nociceptive Processing

In order to properly understand nociceptive processing, adequate methods must be found to evoke neural activity associated with pain, and appropriate biomarkers should be found that can be used to quantify when a person feels pain, and how this can be compared to other "kinds" of pain. This section will explore some of the results that have been found in the field of nociceptive processing.

2.1.1 Evoking Pain

Stimulation can take several forms. Some examples include chemical stimulation through capsaicin patches, mechanical stimulation through cutting or percussive probing, thermal stimulation through hot probes, or electrical intracutaneous or transcutaneous stimulation.

The ideal stimulus innervates the Aδ- and C-fibers exclusively and instantaneously. This means no other nerves should be generating action potentials due to the stimulus, and the stimulus should be able to quickly increase and decrease without a long ramp or lingering after the stimulus has ended.

These requirements have lead to intra-epidermal electrical stimulation to be a popular option for nociceptive research, as electrical pulses can be closely controlled in amplitude and in time [2]. By varying the pulse parameters, the behaviour of the nociceptive nervous system can be observed.

2.1.2 Observing Pain

As the experience of pain is completely subjective, the common method to report a stimulus is the perceived pain intensity. This can be done using a Numeric Rating Scale (NRS), a Visual Analogue Scale (VAS), or similar pain reporting questionnaires. A major disadvantage of these methods is that nothing about the inner workings of the underlying mechanisms can be observed. Even though they are useful to measure the general intensity and qualities of the pain, these methods are also prone to response bias.

Other methods utilize neuroimaging techniques, such as Electroencephalography (EEG), to measure evoked cortical activations as a result of given stimuli [3].

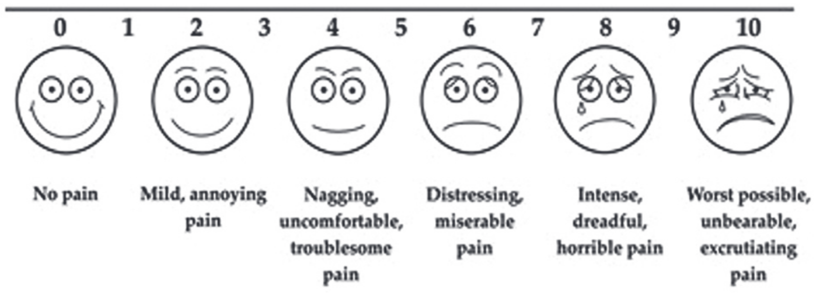

0
1
2
3
4
5
6
7
8
9
10
No pain
Mild, annoying pain
Nagging, uncomfortable, troublesome pain
Distressing, miserable pain
Intense, dreadful, horrible pain
Worst possible, unbearable, excrutiating pain

Figure 2.1: The Visual Analog Scale for pain research [4]

2.1.3 Psychophysics in Pain

Quantitative Sensory Testing (QST) is a psychophysical method in which the threshold of detection and magnitude of the perception with respect to the stimulus intensity are tracked. This is a useful diagnostic tool to identify the changes in sensory function due to symptoms such as allodynia and hyperalgesia [3], [5].

The concept of relating a physical stimulus to a subjective experience is an area of psychology known as psychophysics [6]. In this field, experiments are designed to create a model of an individual or group in order to predict their subjective experience of a known stimulus. In the field of nociceptive processing, this means devising experiments to find a threshold at which a stimulus can be perceived for a particular stimulus.

In Figure 2.2, a description of such a procedure is shown. Each given stimulus is paired with its corresponding response to form a pair. Based on previous stimulus-response pairs, the next stimulus can be determined. After enough data has been collected, these pairs are used to generate the final estimate of the system properties.

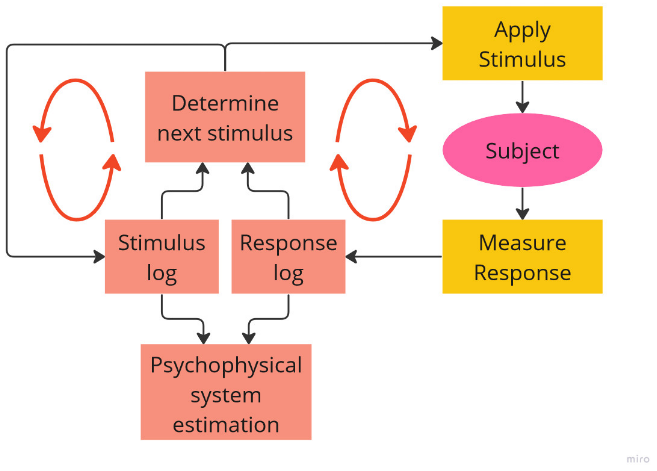


Figure 2.2: Psychophysical system estimation. Functionality fulfilled by software is colored red, and those fulfilled by actuators and sensors in yellow.

In nociceptive research performed at the University of Twente, these stimuli are given through a stimulation electrode, through which only nerves associated with the nociceptive nervous system are activated. The subject is then tasked to momentarily release a held button as quickly as possible after they think they felt a stimulus, and then hold it down again. Because of inherent noise in human senses, each stimulus does not always result in the same response. Sometimes a subject will miss a stimulus that they had felt before. This is especially true close to the threshold, where it effectively becomes a guess. In order to have enough confidence in this threshold, many stimuli close to the threshold are needed to give a reliable result.

One method for this is by using a randomised staircase procedure, based on the staircase method [7]. If the previous stimulus was not detected, the intensity of the next stimulus will become higher. If the stimulus was detected, the intensity will be lowered instead. Random amplitude noise is added to the stimuli to reduce the predictability of the procedure.

By determining the thresholds of multiple kinds of stimuli, known as Multiple Threshold Tracking (MTT), more parameters of the system can be determined. By employing an EEG measurement during stimulation additional response data can be gathered. These kinds of methods are employed during experiments known as NDT-EP [3], where the EEG signals, as well as the button response are used to correlate the button response with the EEG readings. The full experimental setup of such an experiment is given in 2.3. The stimulator and response button are part of the same handheld device, and the controller is a program running on a laptop. The various system blocks are indicated by the blue-green blocks. Interfaces with the subject are displayed with ovals, and sources of noise in the subject are indicated with red clouds.

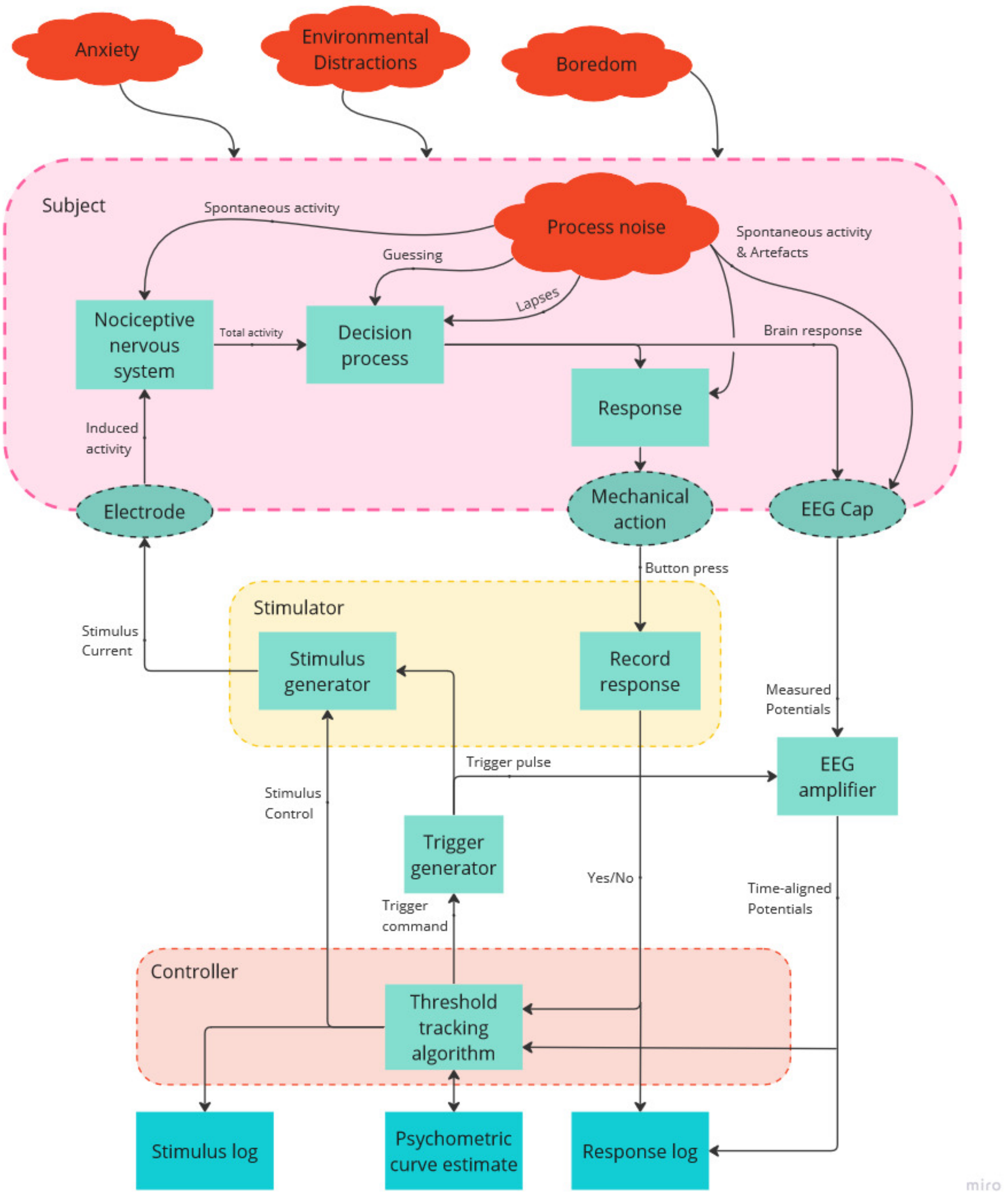


Figure 2.3: NociTRACK NDT-EP experiment setup. The various system blocks are indicated by the blue-green blocks. Interfaces with the subject are displayed with ovals, and sources of noise in the subject are indicated with red clouds.

2.2 StimCom 2.1

In order to facilitate the translation of the wireless communication protocol between the personal computer and stimulators to BLE, the current protocol must be altered. This section will briefly explain the technical details of the existing StimCom protocol, referred to as StimCom 2.1, or StimCom Serial.

2.2.1 Protocol

As Bluetooth Classic maintains a peer-to-peer connection, devices communicating over a StimCom 2.1 connection are able to use a command-response structure. After the stimulator has received the a command, it will check the validity of the parameters. Parameters are considered valid if the stimulator is able to execute the command with the given parameters. If all parameters are valid, it will respond with an identical command. Invalid parameters will be adjusted, and if so, the response will contain the adjusted parameters. The stimulator will then execute the command in the response packet.

Certain commands only request information from the stimulator. In these cases, the command packet will contain the same number of fields as are expected in the response packet, with the `0` character in place of the data. The response packet will then contain the appropriate data. Some commands also result in a secondary response packet, sent at a later point in time.

If a command is not recognized, or cannot be reasonably corrected by the stimulator, it will respond with an error packet.

2.2.2 Command Overview

A short overview of the available commands are given in Table 2.1.

2.2.3 Packet

StimCom packets are ASCII-encoded strings, structured with a header, payload and trailer.

1. The header determines the format of the payload. Payload length is dynamic, with a maximum packet length of 255 characters.
2. The payload is a series of ASCII-encoded value fields, preceded by comma `,` characters.
3. The trailer consists of the null character `\0`.

For example, the ASCII-encoded packet `C,1,1,0\0` has a header of `C`, followed by the values `1`, `1`, and `0`. The full documentation of StimCom 2.1 can be found in Appendix B: StimCom Serial Documentation.

Command	Short Description	Header (Hexcode)
Version Query	Requests the device serial number and software version.	V (0x56)
Feature Query	Requests the hardware features of the device.	F (0x46)
Interval Configuration	Sets the length of the interval between pulses.	I (0x49)
Pulse Channel Configuration	Sets the channels on which the pulses of the pulse train will occur.	P (0x50)
Positive Pulse Amplitude Configuration	Sets the pulse amplitudes of the positive pulses of the pulse train.	A (0x41)
Negative Pulse Amplitude Configuration	Sets the pulse amplitudes of the negative pulses of the pulse train.	a (0x61)
Positive Pulse Width Configuration	Sets the pulse widths of the positive pulses of the pulse train.	W (0x57)
Negative Pulse Width Configuration	Sets the pulse widths of the negative pulses of the pulse train.	w (0x77)
Channel Enable Configuration	Command used to disable or enable a channel on the stimulator.	C (0x43)
Power Configuration	Command used to disable or enable the high voltage on the stimulator.	M (0x4D)
Stimulation Command	Performs a stimulus, and returns the response.	S (0x53)
Check Response Command	Command used to check the state of the stimulator	R (0x52)

Table 2.1: Overview of StimCom commands

2.3 A Crash Course on Bluetooth Low Energy

This section aims to provide a (very) quick overview of the Bluetooth Low Energy (BLE) protocol, and points out some key differences with Bluetooth Classic.

2.3.1 Network Structure

Bluetooth Low Energy follows a client-server structure [1], where the central device is a client of the peripheral server. In the context of BLE, central refers to a device initiating a connection, such as a laptop or phone. Devices that can be connected to, such as smart devices or a wireless stimulator, are referred to as peripherals. This client-server structure allows BLE centrals to connect to multiple peripherals, and allows multiple centrals to connect to the same peripheral, as opposed to the one on one connection provided by Bluetooth Classic.

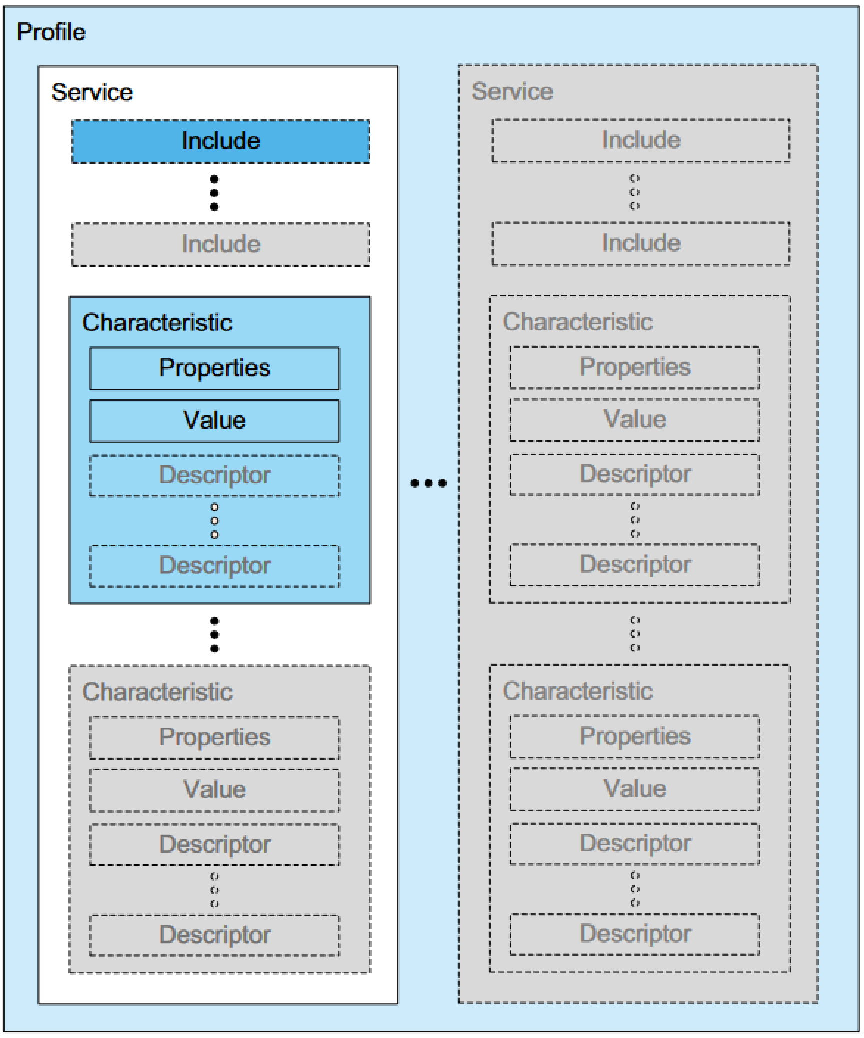


Figure 2.4: The Generic Attribute Profile (GATT)-based Profile hierarchy [1]

2.3.2 The Generic ATTribute Profile (GATT)

As Bluetooth was originally meant as a cable-replacement technology, it only facilitated a continuous stream of data between two devices. BLE took a different direction, and attempted to reduce the number of transmissions necessary for communication through the client-server structure. A BLE peripheral server maintains a set of generic attributes, which contain some value and are addressable by some unique ID, known as a Universally Unique Identifier (UUID). The complete collection of attributes maintained by a peripheral are known as the GATT Profile of the peripheral. As part of the Profile, the peripheral may offer any number of GATT Services, which are collections of related attributes. These Services then contain the GATT Characteristics, which are attributes that store a value, information about that value, and how the client may interact with them. These Characteristics can be accessed by clients, and can be seen as the containers for values on the peripheral. The full GATT-based Profile hierarchy is shown in Figure 2.4.

Through the GATT Profile, the server can request (read), or change (write) the value of a characteristic. If the value changes by an operation not initiated by the client, it can be notified by the server (indicate). This outlines the 3 fundamental operations that are possible through BLE: read, write, indicate. These operations are all acknowledged, though some distinction is made for similar operations that are not acknowledged.

2.3.3 Connection Intervals and Packet Structure

In general, Bluetooth connections are known as quite power hungry. This is because in Bluetooth Classic, each device is assigned alternating timeslots of 625 μs in which it must transmit a packet, which means the radio transmission hardware is activated continuously while the connection is active. As BLE is designed to use as little energy as possible, the radio modules of both peripheral and central devices tend are silent most of the time. Periodically, the central will tell the peripheral to wake up and give it the possibility to exchange data. During such a connection interval, data can be exchanged alternating bidirectionally, as long as the devices can respond to each other within the 150 μs Inter Frame Space (ISI) communication will continue. If the ISI has been exceeded without communication from either device, communication is halted until the next connection interval. Most consumer devices do not acknowledge write operations before the ISI is exceeded, meaning acknowledged operations tend to take at least 1 connection interval, and can only exchange 1 packet in either direction. As a direct consequence, much less data can be transmitted over BLE, compared to Bluetooth Classic.

During the interval, the payload length may not exceed the Maximum Transmissible Unit (MTU), which is the maximum amount of data that may be sent in a single GATT operation. If the MTU exceeds the Protocol Data Unit (PDU), the operation is split across multiple packets. The default MTU size is equal to the minimum of 23 bytes, but many implementations allow MTU sizes up to 512 bytes. The structure of a Bluetooth LE packet is shown in Figure 2.5.

2.4 Consequences for Design

As can be seen, the BLE protocol is fundamentally different from Bluetooth Classic. The central challenge will be to translate a serial connection protocol to non-serial client operations, while maintaining identical functionality. Additionally this protocol should be modular to accept more functionality as the system is expanded. The protocol must then be implemented in a cross-platform application. This program, however, does not exist yet, so the basis must be formed for the application in a way such that future higher software layers are easily able to implement existing and new experiments for nociceptive research.

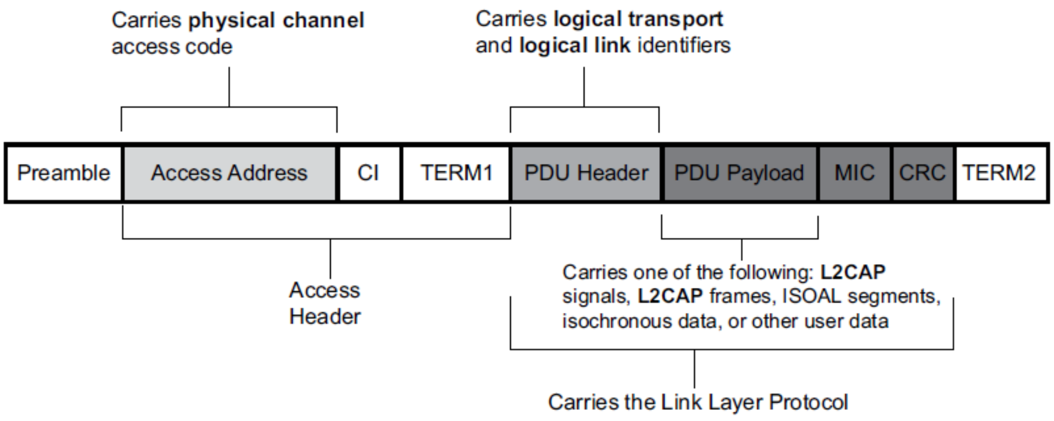

Carries physical channel access code
Carries logical transport and logical link identifiers
Preamble
Access Address
CI
TERM1
PDU Header
PDU Payload
MIC
CRC
TERM2
Access Header
Carries one of the following: L2CAP signals, L2CAP frames, ISOAL segments, isochronous data, or other user data
Carries the Link Layer Protocol

Figure 2.5: An BLE data packet, using encoding [1]

Chapter 3

Design

In order to create a cross-platform application, and translate the StimCom wireless protocol to BLE, the NociTRACK system requires a new iteration.

In this design chapter, first the global architecture of the system is explained, what functionalities are to be fulfilled by which subsystems, and which parts are not part of the system at all. Then the new StimCom protocol is explained, as well as the implementation of StimCom on the stimulator. Finally the software application will be discussed, as well as the application side implementation of the new StimCom protocol.

Due to a lack of documentation about the implementation of the existing system and its protocols, the requirements of the NociTRACK (sub)systems are documented for future design and verification efforts. These documents have been included in Appendix A,B, and C. As the documentation of the whole system is out of scope for this thesis, only relevant sections have been worked out.

3.1 System Design

This section will focus on the global design of the NociTRACK system, and the division of functionality across its subsystems.

3.1.1 System Goals

The ultimate objective of the NociTRACK system is to be able to perform research experiments on the perception of pain, such as the NDT-EP exeperiment in Section 2.1.3, through the nociceptive nervous system at scale. In order to accomplish this, the system must meet many requirements. As an exhaustive list is not productive, several key goals are listed here:

- In order to facilitate many different tests, the hardware and software of the system must be modular.
- The stimulation hardware must be only innervate relevant nociceptive nerve fibers, such as $A\delta$- and C-fibers.
- The stimulation hardware must be electrically isolated from any main voltage sources.
- The system must be suitable to quickly develop and test new experiments.
- Experiments using the system must be able to be performed quickly and easily by any physician, who does not necessarily have the medical or technical background to understand the system.

- External sensors must be able to synchronize in time with the stimuli given by the stimulation hardware.

3.1.2 Functional Breakdown

Using the goals from Section 3.1.1, and based on the existing NDT-EP experimental setup given in Figure 2.3, the NociTRACK system can be broken down into the following function components:

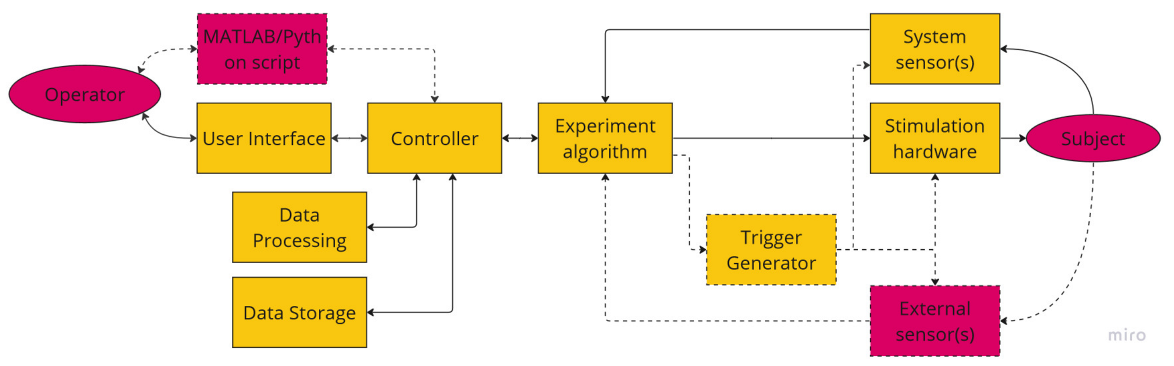


Figure 3.1: NociTRACK Functional Breakdown. Functions not critical to system operation are marked by dashed lines. Functionality external to the system are colored purple

- **Operator**
  Conductor of the experiment. Can be a researcher, or other medical personell.
- **User Interface**
  Presentation software layer through which the lower software layers can be controlled.
- **MATLAB/Python script**
  An external script that may interact with the controller through a software connection.
- **Controller**
  The top-level software interface with the experiment for the Operator. Can also interface with other external system not directly pertaining to the test algorithm, e.g. secure access, database access, data post-processing. Through the Controller, testing algorithms and parameters can be selected, tests can be started, paused, aborted, etc.
- **Data Storage**
  Long-term storage of experiment parameters, various experiment algorithms, experiment results, etc.
- **Data Processing**
  Computational power and algorithms used before and after an experiment.
- **Experiment algorithm**
  Direct controller of the Stimulator(s) and Sensor(s), and implements the selected test algorithm.
- **System sensors(s)**
  Sensors which directly or indirectly interface with the subject, and whose hardware is directly part of the system, e.g. an integrated button on the stimulator.

- **Stimulation hardware**
  Stimulators which can be configured to give various types of single or multimodal pulses, intended to innervate only a specific population of neurons.

- **Trigger Generator**
  Hardware which output a central trigger pulse to synchronise the time of stimulus across stimulators and sensors. If time constraints for stimulus-response pairs are loose for all sensors, it may be omitted, e.g. only a button press is required. This response may take a while to occur due to reaction and thinking times.

- **External sensor(s)**
  Sensors which directly or indirectly interface with the subject, and whose hardware is not directly part of the system, e.g. an EEG system. Necessity depends on testing algorithm.

- **Subject**
  Subject of the experiment, connected to the Stimulator(s) and Sensor(s)

3.1.3 System Architecture

This breakdown can then be used to generate the architecture of the total system architecture. This architecture is largely based on the experiment setup used in the NDT-EP method.

In general, the system design is divided into a software application, and stimulator device. These two subsystems then communicate between each other through the StimCom protocol. Most subsystems are identical to those described in Section 3.1.2. For the stimulator the system sensor takes the form of a feedback button, and the stimulation hardware is subdivided into a processor, pulse generator, and electrodes. The architecture is shown in Figure 3.2.

The next logical step in the design process is to generate requirements for every subsystem. For this thesis, however, only the StimCom protocol and software application are worked out, and can be found in Appendix A: System Design Document.

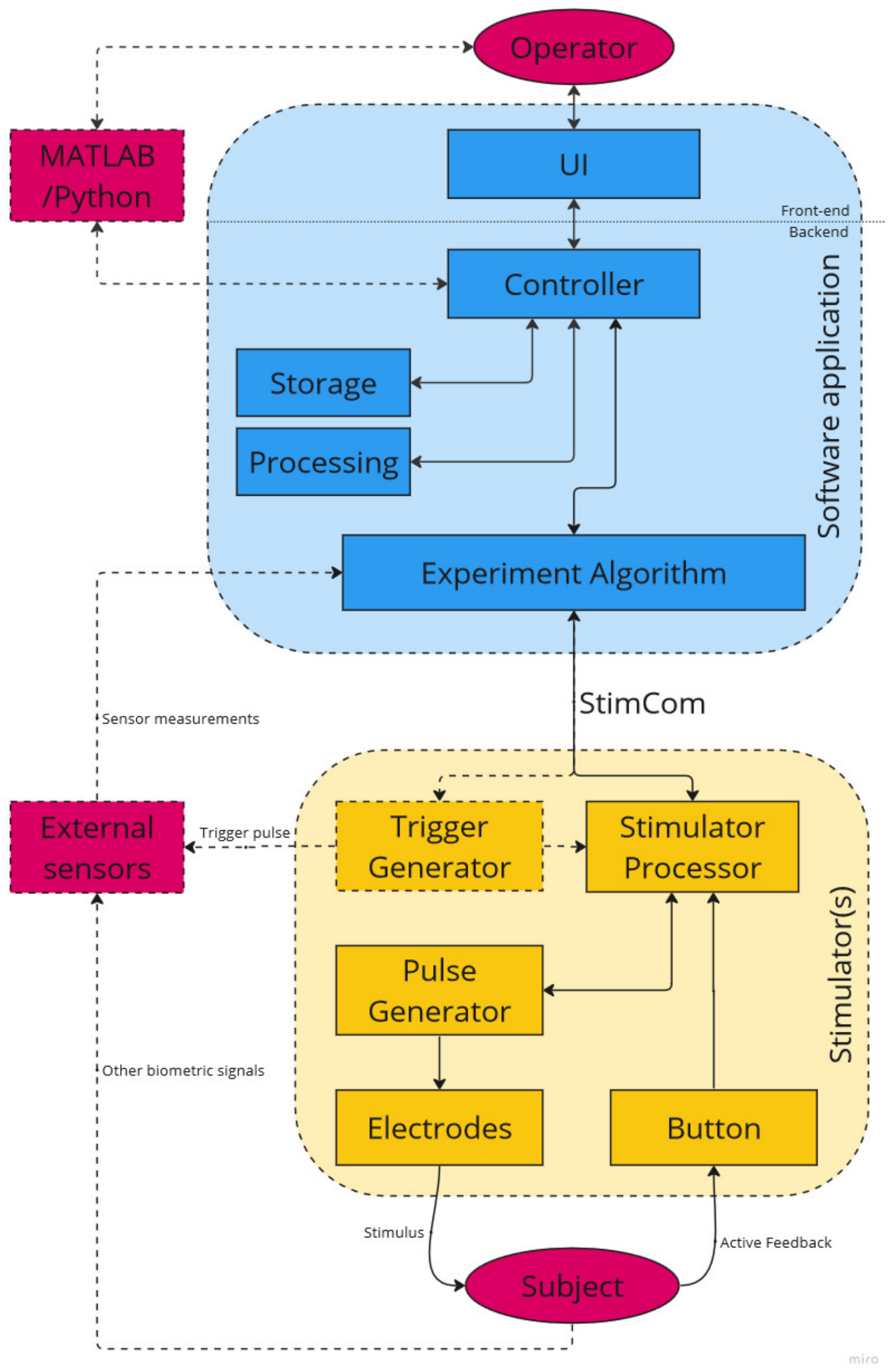


Figure 3.2: NociTRACK System Architecture. Software subsystems are marked in blue, hardware subsystems in yellow, external parts in purple. Some parts are optional depending on the desired experiment. These are marked by dotted lines. The StimCom protocol is explicitly marked in between the application and stimulator

3.2 StimCom

The communication protocol used to communicate between the controller and stimulator is called StimCom. Through this protocol, the stimulator can be configured, pulse trains can be programmed and applied, and responses can be recorded.

As mentioned, the current version of StimCom (StimCom 2.1) uses Bluetooth Classic, which is a serial protocol. As stimulators in the new iteration are required to support BLE, StimCom requires a new iteration that can fulfill the same functionality as before, this version will be referred to as Stimcom 3.0.

3.2.1 StimCom 3.0 Requirements

In general, the goal of StimCom 3.0 is to fulfill the same functionality as StimCom 2.1, through BLE.

The requirements of the StimCom protocol are defined in Appendix A: System Design Document, section 6.1.2, repeated here for convenience in Tables 3.1, 3.2, and 3.3. Note that the Application-Stimulator Interface that is mentioned, is implemented through the StimCom protocol. The design for StimCom 3.0 aims to implement these same StimCom functionalities though BLE instead of Bluetooth Classic.

Table 3.1: Application-Stimulator Interface Functional Requirements Table

ID	**Description**	**Reasoning**
ASI-F1	The interface must be wireless.	The stimulator may not be electrically coupled to other systems.
ASI-F2	The application is always completely aware over the actions of the stimulator.	The stimulator must report its actions to the application, and may not take independent action.
ASI-F3	The application must be able to configure the pulse train on the stimulator.	Pulse trains must be configurable.
ASI-F4	Pulse trains are applied when the application gives the appropriate command.	Stimulators may not perform stimuli autonomously.
ASI-F5	The application must be able to read the response of a stimulus.	After a stimulus, the response should be available.
ASI-F6	90% of pulse trains configurations must complete in no longer than 1 second.	To be able to deliver new pulses quickly enough for experiments, configuring a pulse may not take too long.
ASI-F7	100% of sent commands must be received by the stimulator.	The stimulator must reliably receive packets sent by the application.

Table 3.2: Application-Stimulator Interface Non-Functional Requirements Table

ID	**Description**	**Reasoning**
ASI-NF1	Establishing the connection from the application to the stimulator must complete with 4 seconds after initiation from the application.	A fast connection procedure results in smoother setup for experiments

Continued on next page →

→ Continued from previous page

ID	Description	Reasoning
ASI-NF2	Establishing the connection from the application to the stimulator must not require intimate knowledge of the system.	Minimal technical knowledge should be required from the operator

Table 3.3: Application-Stimulator Interface Other Requirements Table

ID	Description	Reasoning
ASI-O1	The interface must be modular, to allow functionality to be added or altered in future iterations.	The interface must be prepared to handle functional changes in both the application and stimulator, without needing a complete redesign.

3.2.2 StimCom 3.0 Protocol

GATT Service

The protocol is implemented under a GATT Service, with a predefined UUID, so that the application can learn which devices support the protocol by looking which Services are offered by a peripheral. To implement the various commands from StimCom 2.1, a Characteristic is added for every command. This eliminates the need for a header or trailer, as the payload is already encased by the GATT and physical protocol layers.

Payload

The payload will still contain the same ASCII-encoded data as in StimCom 2.1, excluding the redundant leading comma. Value fields are still separated by `,`-characters, this is done as the number of fields of some commands is variable, and the original implementation is functional. A more compact encoding is very much possible, but this was considered out of scope of this thesis, as the objective was to adapt the StimCom protocol for BLE. However, ASCII encoding of numeric data, as well as using a whole 8-bit character to separate fields, makes inefficient use of the limited space in BLE packets.

Operations

As reliability is valued over speed in this protocol we will only consider acknowledged operations. Read operations are largely unused, as most communication must be initiated by writing a command.

Some commands, such as Version Query and Feature Query, result in hardcoded parameters on the stimulator. For these commands, the command-response structure is not needed, as the read operation will suffice for them.

To implement the command-response structure, the write and indicate operations are needed. To execute a command, the controller performs a write operation on the appropriate characteristic. The stimulator will follow with an indication for the same characteristic with the response packet, which possibly changed parameters if they were invalid. For some commands, a secondary response packet sent at a later time is required. For these commands, another indication will be sent on the same characteristic. This message sequence is visualized in Figure 3.3.

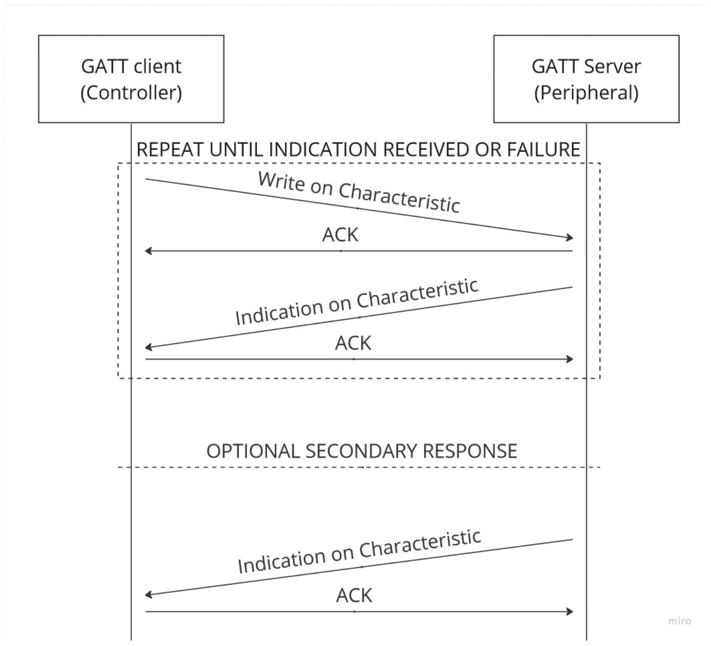

GATT client
(Controller)
GATT Server
(Peripheral)
REPEAT UNTIL INDICATION RECEIVED OR FAILURE
Write on Characteristic
ACK
Indication on Characteristic
ACK
OPTIONAL SECONDARY RESPONSE
Indication on Characteristic
ACK
miro

Figure 3.3: StimCom 3 Command-Response Message Chart

If any command has invalid parameters that cannot be reasonably corrected by the stimulator, it will respond with an error indication: `!`.

The full documentation of the documentation can be found in Appendix C: StimCom 3.0 Specification.

3.3 StimCom implementation - Stimulator

For the initial implementation of the StimCom 3.0 protocol, the AmbuStim stimulator is used. The embedded architecture of the AmbuStim Stimulator is shown in Figure 3.4. As the existing Bluetooth Module does not support BLE, it is replaced with a newer version that does support it. A particular challenge with the implementation was that the serial protocol with the processor uses the StimCom 2.1 protocol. The original implementation used this, as it was trivial to translate the protocol between Bluetooth Classic and UART. In this design, the serial communication continues using the StimCom 2.1 protocol, as changing the embedded code on the processor would require a major overhaul that could impact the performance of the firmware generating the stimulation currents. This was deemed more appropriate for a separate assignment. Instead, it was decided to implement a translation step inside the BLE module from StimCom 3.0 to StimCom 2.1. This will be implemented as a small embedded software program.

3.3.1 Hardware

The nRF52840 BLE module is used to replace the existing Bluetooth module. This module notably features a Bluetooth 5.3 transceiver, as well as a 32-bit 64MHz ARM M4 Cortex processor a, several serial interfaces including UART, and 48 general purpose IO pins. For the proof-of-concept implementation the Adafruit Feather nRF52840 Express module is used, as it integrates this module on a printed circuit board, and provides a USB interface that allows for onboard programming and debugging from a pc, using the Arduino Development Environment.

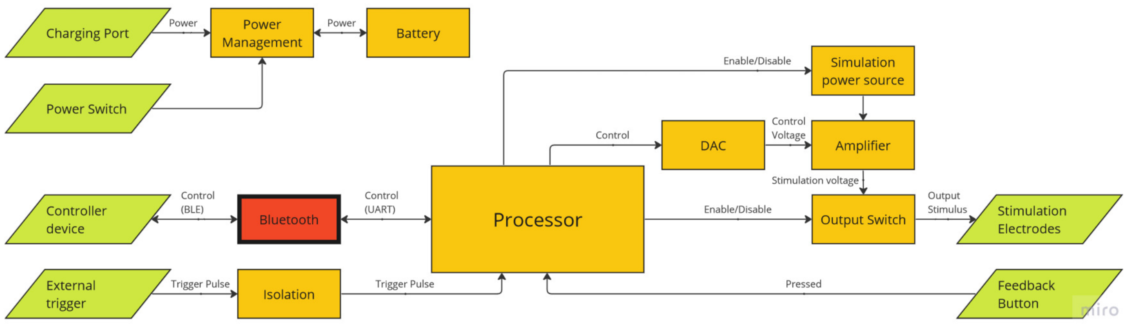


Figure 3.4: The embedded architecture of the AmbuStim Stimulator. Hardware subsystems are shown in yellow, software modules in blue, and external hardware interfaces in lightgreen. The Bluetooth module is highlighted in red.

The Bluetooth module uses a 2-wire Universal asynchronous receiver/transmitter (UART) serial interface for communication with the processor. The hardware interface is shown in Figure 3.5. The baudrate is set to 9600, with 8 data bits and a parity bit. Communication utilizes the StimCom 2.1 protocol, in the same way as discussed in Section 2.2: StimCom 2.1.

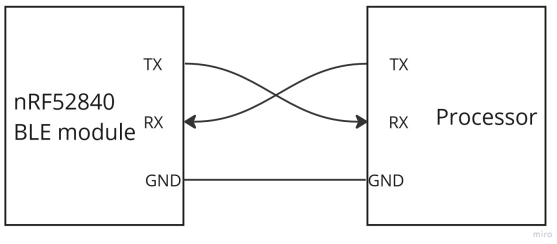


Figure 3.5: The UART serial interface on the AmbuStim.

3.3.2 Software Architecture

The Adafruit Feather nRF52840 Express includes an Arduino implementation of a BLE library that interfaces with the lower radio functions. As the Arduino libraries provides software libraries for interfacing with the nRF52840 processor functions, the Arduino framework was used to build the embedded application. It should be noted that, in general, using Arduino is not recommended for professional use, as library implementations are generally not as efficient, contain many unused parts which occupy memory space, and the Arduino IDE does not support debugging. For proof-of-concept and prototyping, however, the pre-built libraries save significant time implementing lower layer software.

In Figure 3.6, the architecture is shown. The StimCom 2.1, and StimCom 3 protocols are implemented on the Arduino and Adafruit libraries for UART and BLE, respectively. The top-level application will provide the logic needed to translate between the two protocols.

3.3.3 Software Design

The application is programmed in `C++`, using the Arduino environment, and Adafruit libraries to use the BLE functions. Before compilation, the StimCom GATT Service is defined in a `.XML` file. This file is shared between the stimulator and application implementations, to ensure the definition of the BLE Service is identical. From this definition, a `.H` and `.CPP`

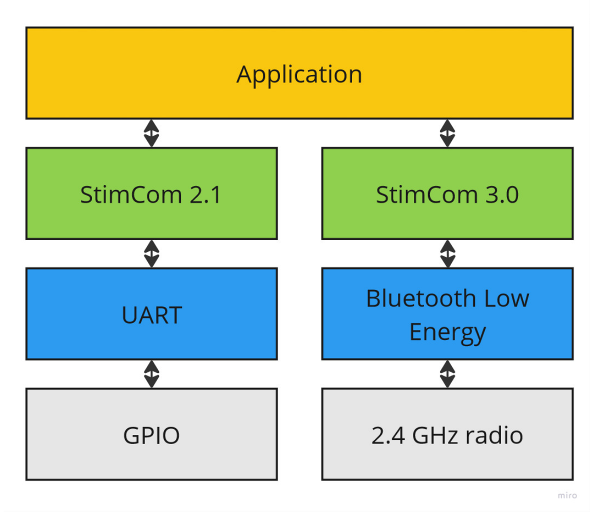


Figure 3.6: BLE module embedded software architecture

file are generated using a Python script. These files provide the `C++` classes and methods that can be called by the application to define the configuration of the StimCom 3.0 GATT Service.

After boot, the module first configures the UART and BLE interfaces, and fetches the device constants from the processor through the Version Query and Feature Query commands. It then starts broadcasting itself over BLE to surrounding devices and enters the event loop. The boot sequence is shown in Figures 3.7.

As mentioned, the UART connection is configured with 9600 baud, 8 data bits and a parity bit. The BLE device is configured with the StimCom 3.0 GATT Service. As peripherals cannot force a particular connection interval and MTU, only request that the master will provide something sufficient, peripherals must implement for a range of values that will work. For this application the device is configured to accept any MTU size, down to the minimum of 23 Bytes. The connection interval is accepted, as long as it is below 62.5ms, as for longer intervals some implementations of Bluetooth Low Energy will always refuse the connection.

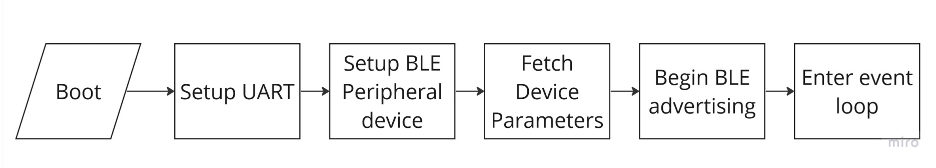


Figure 3.7: BLE module embedded software boot sequence

After the initial setup is complete the stimulator will wait for a connection to be established, after which it will enter an event loop. While no events occur, the processor will remain idle.

To act as an intermediary between the two protocols, two events are defined. The first is triggered by a write operation on a Characteristic, note that each Characteristic has its own event. This will convert the payload to a StimCom 2.1 packet, and send it over UART. It then awaits the response, and broadcasts the indication. If no response is received over UART, an error will be sent instead.

The second event is triggered by a message on the UART connection, note that this does not include messages received in response to messages from the Bluetooth module. The only command that is expected to be received this way is the secondary packet from the Stimulation Command. These packets are converted to StimCom 3 packets, and sent as an indication on the Stimulation Command Characteristic. If any other messages are received, they will be discarded.

The flowcharts of the events are shown in Figure 3.8.

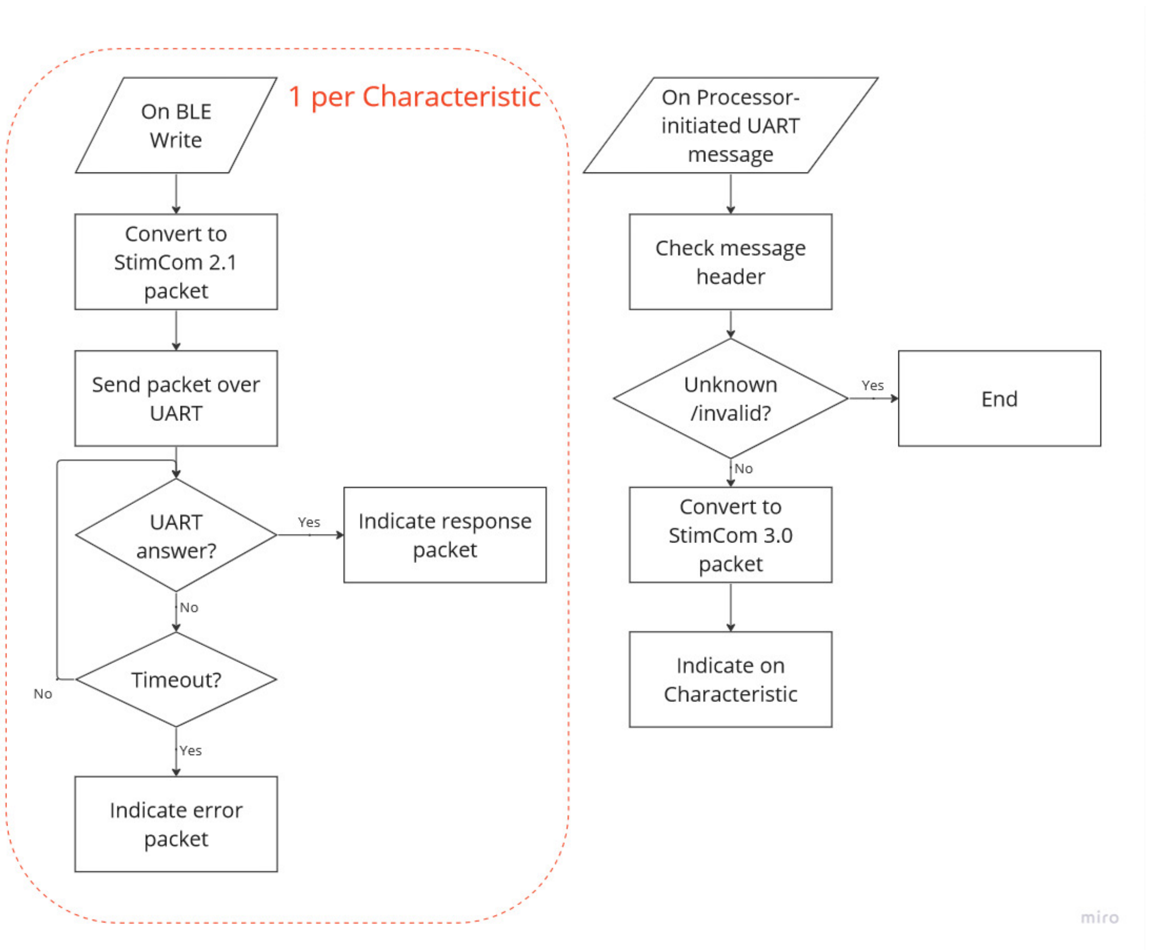


Figure 3.8: BLE module embedded software events

3.4 Application

The NociTRACK application is the software application that implements the user interface, experiment algorithm, and other software components needed to perform experiments. This section will work out the requirements and choice of development ecosystem.

3.4.1 Functional Description

Through the application, the operator is able to connect to one or more stimulators, and optionally other external sensing apparatus, and is able to configure a pre-programmed experiment to use these peripherals. The experiment controller then autonomously is able to conduct the experiment, as configured by the operator. Researchers are able to connect to the application from a MATLAB or Python script to try new experiments and configurations. These newly developed experiments can then be implemented into the application itself, for other operators to use.

3.4.2 Requirements

The full list of requirements can be found in Appendix A: System Design Document, section 5.3.2, repeated here in Tables 3.4, 3.5, and 3.6.

Table 3.4: Application Functional Requirements Table

ID	Description	Reasoning
A-F1	The application must be compatible with Windows desktops and Android and iOS devices.	These platforms cover a large part of operating systems on devices used by medical personell.
A-F2	The application could be compatible with Linux and MacOS desktops.	These platforms, while not mandatory to have, would expand the number of devices that are able to run the application.
A-F3	The application must be able to utilize Bluetooth Low Energy in order to communicate with stimulators.	BLE is mandatory in order to connect to stimulators and conduct experiments.
A-F4	The application must be able to configure a pulse train on stimulators, and be able to send the command to trigger the stimulation.	Configuring and giving stimuli is mandatory in order to conduct experiments.
A-F5	The application must be able to register the subject response after a stimulus.	Recording subject responses is mandatory in order to conduct experiments.
A-F6	The application must be able to interface with other types of sensors, not part of the NociTRACK system.	By providing interfaces to external sensor systems, many other types of experiments are able to be conducted. The sensors with which the system should be able to connect with, depends on the desired experiments.

Continued on next page →

→ Continued from previous page

ID	Description	Reasoning
A-F7	The application must provide an intuitive interface with other applications, such as MATLAB and Python, in order to allow control over the stimulators.	MATLAB and Python provide many functions which allow for rapid prototyping of experiments. This functionality is essential for researchers developing new algorithms and experiments.
A-F8	The application must be able to load pre-programmed experiment algorithms and parameters for experiments.	Developed experiments should be able to be permanently registered by the application, and reproduced, without need for third party applications, such as MATLAB.
A-F9	The application must be able to import and export relevant subject parameters for experiments.	Experiment parameters with differ based on the type of experiment and between subjects. These should be easily stored and recalled for the operator.
A-F10	Experiments should be able to be aborted at any point by the operator	In case an experiment must be cancelled, e.g. in case of an emergency, the program should be able to be aborted.

Table 3.5: Application Non-Functional Requirements Table

ID	Description	Reasoning
A-NF1	The application must have an intuitive interface, and enough autonomy to provide operators with little technical knowledge with the ability to perform any experiment with the system.	The enables as many people as be able to conduct experiments, thus enabling large scale testing.
A-NF2	The connection procedure used to connect to stimulators in a fast and safe way.	Fast connecting makes setting up an experiment less laborious, and is generally considered safer.
A-NF3	The connection procedure used to connect to stimulators should leave no doubt the operator is connecting to the desired device.	When many stimulators are present, each should be distiguishable in the application to prevent accidental wrong connections.

Table 3.6: Application Other Requirements Table

ID	Description	Reasoning
A-O1	The BLE connection should be encrypted.	Connections with stimulators should be secure.
A-O2	The application should be easily extensible, to allow new functionality and support for other operating systems	The development team of the Noci-TRACK system is frequently changing. The application should be developed in such a way that incremental development is easily possible for new developers.

3.4.3 Software Development Kit (SDK)

For the SDK, the intuitive preference goes to an open-source environment, as open-source software tends to be more widely supported, and have more public resources available. As the application must be operable on several operating systems, environments that are able to compile their code cross-platform are preferred, as this means high-level code only needs to be developed once. As the mantra for cross-platform goes: "write once, run anywhere" [8]. As platforms differ in implementation at the lowest level, depending on requirements, some platform-specific programming for BLE is very likely still necessary. It is preferred to have the software front- and back-end unified in a single environment, as this simplifies the development cycle significantly. Some options meeting these preferences for the SDK are listed below.

React Native
React Native is a free open-source cross-platform framework, officially released in 2015 by Meta (formerly Facebook) [9]. Application in React Native are developed in JavaScript and JSX. As JavaScript applications must use bridges to communicate with native code, performance can be impacted if the application intensively interfaces with lower layers. React Native has an active community-driven collection of libraries, providing a wide breadth of functionality to developers.

Flutter
Flutter is a free open-source cross-platform framework for application development, officially released in 2017 by Google [10]. Flutter applications are developed in the Dart language, which is inspired by C++ (pronounced *see-plus-plus*). Unlike many high-level languages, Dart code can be compiled to platform-specific machine code, which results in excellent native-like performance. Flutter prides itself on its shallow learning curve and minimal template code for its source files (known as "boilerplate"). As both Flutter and Android are Google products, Android implementations tend to enjoy some favoritism in Flutter.

Xamarin
Xamarin is a free open-source cross-platform development framework, created in 2011 by Microsoft, built off of the .NET (pronounced *dot-net*) framework [11]. Applications in Xamarin are programmed in any of the ".NET languages": C#, F# (pronounced *see-sharp* and *eff-sharp* respectively), and Visual Basic. Xamarin is considered to be quite optimized, as part of the effort to approach native-like performance. Being developed by Microsoft, Xamarin inherits ecosystem of tools, such as .NET and Microsoft Visual Studio. It also provides extensions to create platform-specific implementations.

Comparison

As all presented options are able to meet the requirements, this becomes a choice of preference. React Native, Flutter and Xamarin are compared on several categories:

Maturity
Xamarin, being one of the first endeavours into cross-platform development, is easily the most mature environment. This has the advantage that plenty of examples can be found online, and a wide array of tools are available. Flutter is the youngest of the options, which means it has more limited tool options for developers, though it is gaining more support through its growing community.

Development Language
JavaScript and JSX are dynamically typed by nature. This means the language is very

versatile, but can create odd quirks, as any type of data is accepted in functions without explicit checks. C# is a statically typed language, meaning it is not as versatile, but is generally considered safer and more robust. Dart offers static typing, but also provides a dynamic memory type, allowing the choice up to the developer.
The main targets for JavaScript applications are in web development, while C# is a general-purpose programming language, used to develop a wide array of programs and applications. Dart is used for web and general application development, predominantly using the Flutter framework, and is exportable to JavaScript for web applications.
JavaScript and Dart are single-threaded, meaning waiting code and time-consuming processes will block execution. Dart circumvents the blocking problem, by employing an asynchronous execution of code. When a function is awaiting a return value, Dart will start executing a different part of code, so the program will not halt. C# fully supports multi-threading, and therefore does not have these issues.
None of the languages are particularly complex, and can be learned without difficulty with only basic programming experience. Dart, however, has shown a strong orientation towards beginners, and provides excellent tutorials, tools and documentation.

Performance
React Native requires a JavaScript bridge to native code of the device, which slows down performance, especially if many calls are made to parts native to the device. Xamarin is very optimized due to its maturity, achieving near-native performance. Dart is also very optimized, and can achieve C-like performance across platforms due to ahead-of-time compilation.

Ecosystem
Xamarin and React Native currently boast the most expansive development ecosystem, due to their maturity and active community, respectively. Xamarin, however, has shown a recent decline in popularity, with developers moving to younger alternatives. Ever since its release, Flutter has shown a steady increase in available plugins and popularity as it matures. All SDKs are supported by large companies, and are still widely used, so support and updates will very likely continue for a long time. Additionally their respective communities are very active, and provide additional support.

Price
React Native, Xamarin and Flutter are all available for free for commercial use.

Choice

React Native becomes less attractive as it is focused on high-level web-like applications. As the program will likely extensively communicate with local modules, such as BLE, which will limit performance. Xamarin and Flutter are both preferable for their good performance, and statically-typed languages. While Xamarin has a more mature array of libraries, it is declining in popularity, and Flutter already boasts a decent catalogue of libraries which is actively growing and improving. As the development of the system is expected to take several years still, it is expected that support and reliability will also grow in that time. On top of this, Flutter has excellent documentation and tutorials. It is expected that the development team will change continuously as new students work on the application, and it is assumed each of them will be required to learn the associated programming languages. Good documentation will aid in better understanding of the language, more robust code, and less time spent on implementation.

It is for these reasons that Flutter was selected as the SDK of the application.

3.5 Application Implementation

3.5.1 Concept

In order to implement the requirements from Section 3.4.2, the following application is conceptualized:

The application implements two concurrent processes, a background service, which implements the communication with the stimulators through StimCom, as well as storage, and data processing required by controlling algorithms, as well as a user-facing process, which will act as a client of the background process.

The background process has no user interface, rather it exposes a WebSocket server on the device. This server can then be connected to through a WebSocket client process. The user-facing process, which contains the user interface and onboard algorithms, is one such process. It will initially connect to the background process and provides a way to start pre-programmed experiments with customized parameters. A user may also choose to disconnect from the background process and expose it to other processes. The user-facing process will then only provide rudimentary operations, such as shutting down the application. Additionally, basic monitoring functionality of the stimulators may be present on the user-process, though direct control should only be possible from only one client.

If the background process is not connected to the user-facing process, it may be connected to by an external process, such as a MATLAB script. Control of the experiment is then relinquished to the connected process.

This method of operation allows any one software process to utilize the full functionality of the background process. This is useful for implementing new algorithms and experiments, or integrating the system as a subsystem into a larger system. As WebSockets operate over TCP/UDP, it is possible to implement the same security mechanisms as for any web service. The server may also be opened on the localhost IP-address. This means only processes running on the same machine as the background process can interact with it.

A suggestion for the WebSocket communication is by transferring JavaScript Object Notation (JSON) strings between the processes. JSON is a versatile way of encoding a hierarchical object as a string, for which many implementations already exist. String encoding is far less of an issue for inter-process communication, as these messages travel far faster than BLE, and do not suffer strict payload size limitations. A lightweight protocol would be wrapped around it to define the NociTRACK WebSocket server/client protocol. This protocol would then communicate basic high-level commands and events that occur on either process. A multi-stage process, such as configuring the stimulus parameters or performing a stimulus with an external trigger, would be condensed into a single command to be handled by the server. Note that the client and server do not share memory, so all relevant value changes between server and client should be communicated.

As the whole application as described is too much to complete in this thesis assignment, it was decided to implement only a test application utilizing the StimCom 3.0 protocol. This application only needs to provide a workable basis that can later be expanded upon to add functionality, as well as methods to test the functionality and performance of the protocol. The architecture of the implementation is given in Figure 3.9. It shows the dependencies of the application on the StimCom library in the back-end and Flutter SDK, which is used to create the user interface.

3.5.2 Bluetooth Low Energy

In an ideal situation, the BLE implementation are already available in one cross-platform available for at least Windows, Android and iOS. However, at time of writing this is not

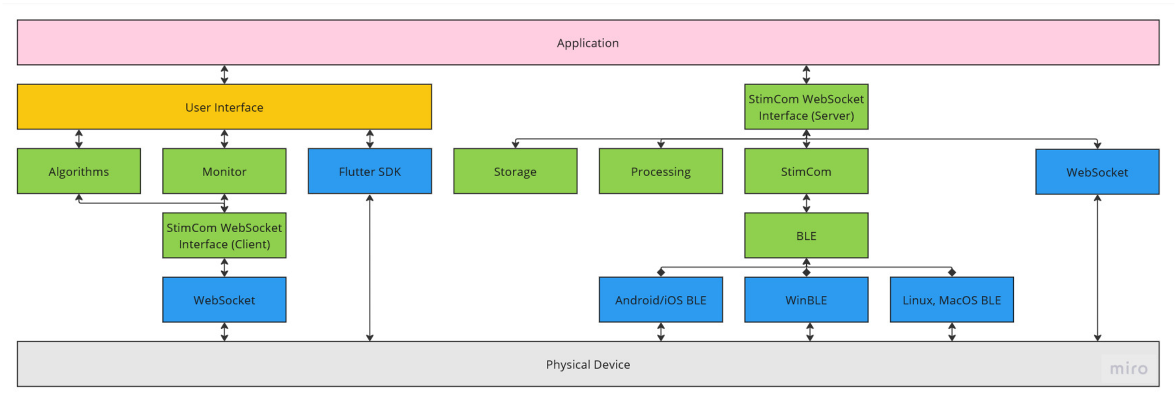


Figure 3.9: Application Software Architecture

the case, and while there are libraries available that have cross platform implementations for Android and iOS, Windows (as well as macOS and Linux) requires a separate library.

For this initial version, Windows was chosen as the platform over Android or iOS, as this platform covers the most use cases for researchers, as most initial tests will occur on laptops and personal computers. It also allows easier debugging and testing in conjunction with the embedded stimulator firmware.

In order to make the implementation in the application platform-agnostic, an additional layer is added between the StimCom library and the platform-specific libraries. This "BLE manager" layer is a functional abstraction layer that can be linked to one of the platform-specific implementations. This means that the StimCom 3.0 library and all higher dependencies, will work regardless of the underlying Bluetooth Low Energy implementation. The available features and functions may, however, differ depending on the platform. Higher level layers are expected to be able to handle these limitations. In Figure 3.9, the BLE layer contains the manager. The BLE layer can be modified in the future to implement other platforms, by using different libraries for various platforms. The higher software layer, such as the StimCom library then do not have to be altered.

The manager implements the following functionalities:

- Scanning for devices.
- Connecting and Disconnecting.
- Pairing and unpairing.
- Subscribing to Characteristics (allowing indications to be sent on it).
- Reading from Characteristics.
- Writing to Characteristics with and without peripheral acknowledgements.
- Receiving Indications and Notifications on Characteristics.
- Storing discovered and connected devices during the session, as well as the configuration of parameters on all devices.

3.5.3 StimCom

The StimCom protocol is implemented as described in Section 3.2.2, and Appendix C: StimCom 3.0 Specification. The full BLE Service table is taken from the specification. It is built on the base 128-bit address `e9ef0001-9644-424f-a318-bf065e5efc6`. All Characteristic UUIDs are based off of this address.

UUID	Description
`e9ef0001-9644-424f-a318-bf065e5efc6`	StimCom Service Base Address
`e9ef0002-9644-424f-a318-bf065e5efc6`	Version Query
`e9ef0003-9644-424f-a318-bf065e5efc6`	Feature Query
`e9ef0004-9644-424f-a318-bf065e5efc6`	Interval Configuration
`e9ef0005-9644-424f-a318-bf065e5efc6`	Pulse Channel Configuration
`e9ef0006-9644-424f-a318-bf065e5efc6`	Positive Pulse Amplitude Configuration
`e9ef0007-9644-424f-a318-bf065e5efc6`	Negative Pulse Amplitude Configuration
`e9ef0008-9644-424f-a318-bf065e5efc6`	Positive Pulse Width Configuration
`e9ef0009-9644-424f-a318-bf065e5efc6`	Negative Pulse Width Configuration
`e9ef000A-9644-424f-a318-bf065e5efc6`	Enabled Channel Configuration
`e9ef000B-9644-424f-a318-bf065e5efc6`	Power Configuration
`e9ef000C-9644-424f-a318-bf065e5efc6`	Stimulation Command
`e9ef000D-9644-424f-a318-bf065e5efc6`	Check Response Command

Table 3.7: StimCom Service UUID table

The message chart for commands sent through StimCom 3.0, is given in Figure 3.10

Packet loss

As is frequently the case with wireless communication, sometimes a packet is lost, or as could be the case with StimCom, the response packet is missed. In these cases, the application cannot be certain the command was executed successfully. Once a command has been initiated, and a timeout is reached, the command is sent again. This repeats until either a response is received, or it runs out of retries and fails completely.

Interface functions

The StimCom library exposes the following functionalities to the upper layers:

- Scanning for StimCom devices.
- Connecting and Disconnecting to StimCom devices.
- Fetching the stimulator version and features.
- Pairing and unpairing.
- Enabling/disabling the high voltage output.
- Configuring a stimulation pulse train.
- Executing individual StimCom functions.
- Callback functions for when a stimulus has been completed.
- Data structures to easily configure pulse trains and other parameters.

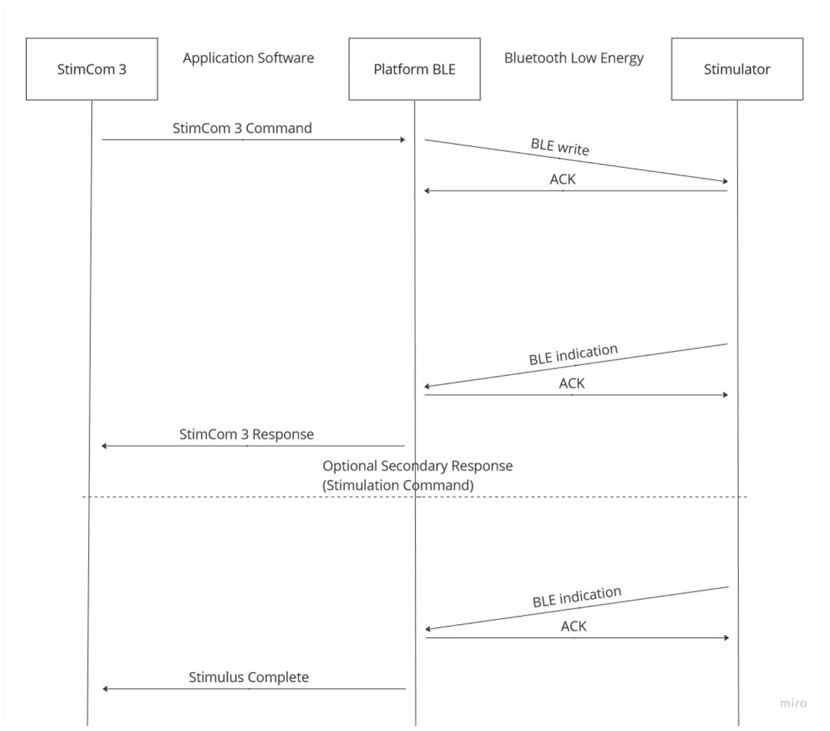


Figure 3.10: The implemented StimCom 3 protocol message chart for all commands except the Version and Feature Query commands, which only support read operations. All commands start with an acknowledged write operation. The response is sent back to the application through an indication, which also receives an acknowledgement. The stimulation command also receives a secondary packet, indicating the stimulus has been completed.

- Memory for storing discovered and connected stimulators during the session.
- Exceptions for when an error packet has been received, desynchonization occurs, or when initialization errors occur.

These functions are all abstracted from the underlying packets and encoding structure of the StimCom protocol. For example, in order to enable the high power on a stimulator, instead of writing `"1,1"` to the Power Configuration Characteristic and waiting on an indication containing the same, a programmer may simply write:
`bool x = await stimulator.setHighPower(true);` which fills the return value that has been set on the stimulator once the operation has been completed.

Similarly, pulse trains can be configured by simply calling the function:

```
List<Pulse> x = await stimulator.setPattern([
    Pulse(...),
    Pulse(...)
]);
```

This command will configure the pulse train to be a series of 2 pulses. Naturally these pulses can be individually configured to have any valid amplitude or duration. The return value will be the pulse train that has been configured on the stimulator.

Values such as pulse amplitude and durations are configured in ADunits and Timerunits by StimCom. After connecting to a stimulator, the calibration values for the timer and

DAC are automatically fetched from the device. These are then used to perform the unit conversions implicitly. This means that amplitudes can be simply set to a double floating point value (with a unit of milliampere), and durations such as pulse widths can be set using the `Duration` data type in Dart.

The Dart library implements several exceptions that should be used by higher software layers to handle any issues that occur on the connection, such as desynchronization between the central and peripheral, and command timeouts.

A note about asynchronous code execution

In the above examples, the `await` keyword is essential for the execution of the asynchronous StimCom functions. As write operations can take several connection intervals, most StimCom functions are implemented with Dart's `Future`s. A `Future` is a value that may not be evaluable at that point, but might be at a later point in time. in the `setPower` example, the return value is of the type `Future<bool>`, which means at some point it will have the value `true` or `false`. The `await` keyword tells the code to wait at that line until the `Future` has completed to is assigned data type, in this case, a boolean value. While the Dart code is waiting in this line, code in other locations can continue running unobstructed.

This is also the mechanism through which the order of operations is preserved. A StimCom function should not be called if one of its `Future`'s has not returned a value or `Exception`. Note that this behaviour is not explicitly enforced by the StimCom library.

3.5.4 User Interface

Flutter applications are built using use Routes and Widgets. Routes are Flutter's interpretation of different "screens" or "pages" like one would think of a login screen, main page, settings page. Widgets are any and all UI elements that can appear on Routes, which includes everything from text to buttons, graphs, and integrated video players.

As the StimCom protocol is maintained by the entire application, and must be persistent across Routes, the StimCom library is instantiated at the root level of the application. From this root instance the Widget tree built up, depending on the current Route to build the user interface.

In order to change a Widget's appearance, e.g. changing text or a color, the entire Widget must be rebuilt. This means that if any change would occur in the StimCom instance, the change would need to propagate up the entire tree. All this while only a small subset of widgets require access to the StimCom instance. Flutter, instead of propagating the instance up through the tree, has a dynamic way of exposing it to a single Widget without needing to rebuild the entire Widget tree, through Providers and Consumers. By creating a Provider of the StimCom instance and state, it is possible to "broadcast" these throughout the application. At any point in the Widget tree, a Widget can mark itself as a Consumer of StimCom, and gain access to its state and functions. The when the StimCom instance experiences some change, only those Widgets that have marked themselves an Consumer will be rebuilt. This way of updating Widgets is visualized in Figure 3.11. This structure allows the StimCom library to run in the background as a Provider, while the UI can still interact with it, and reflect live changes.

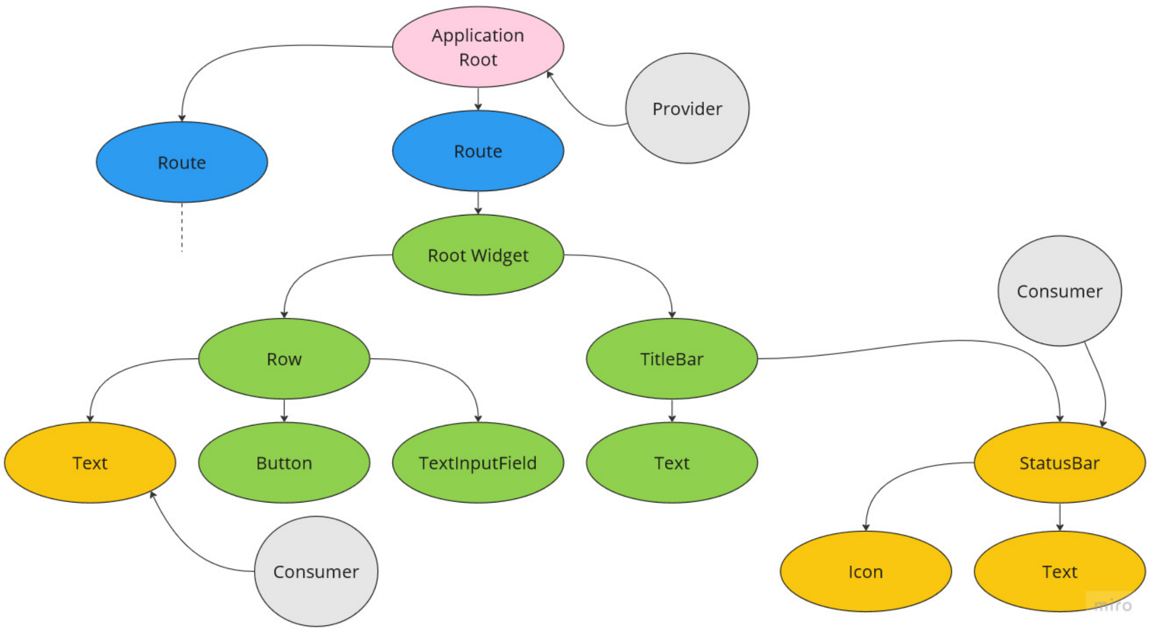


Figure 3.11: Example Flutter UI Widget tree with a Provider and Consumers. The Widgets that are changed after an update from the Provider, are marked in yellow.

3.5.5 Test Application

Using the above mechanism, the application framework was developed. The general usage flow is simply as follows:

The user interface will feature a startup screen, which will check whether BLE is available on the device. If Bluetooth Low Energy (BLE) cannot be used for whatever reason, the user will be informed. If it can be used a button becomes available to start the application. After that, the user will enter a connection screen, through which active StimCom-supporting devices can be discovered and connected to. The user can connect and disconnect to and from a device by tapping on it on the list. While connected to a device, the user can swipe right to enter a screen on which several functional and networking tests can be performed. These tests will be described further in Chapter 4.

The architecture of the software can be viewed in Figure 3.12. The test functions have been compiled in a separate test library for ease of use and modification.

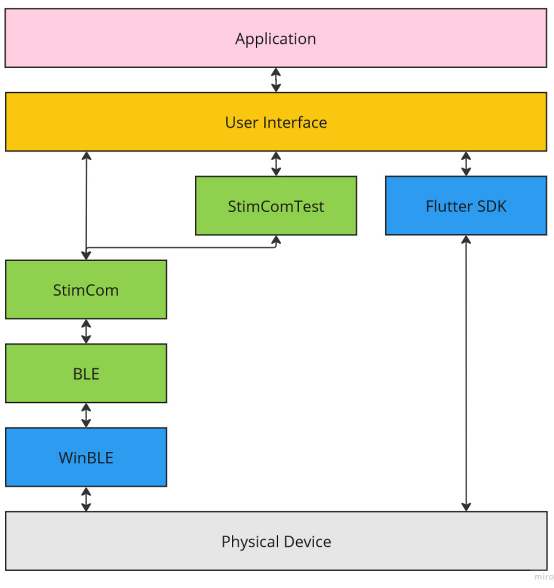


Figure 3.12: Implemented Software Architecture.

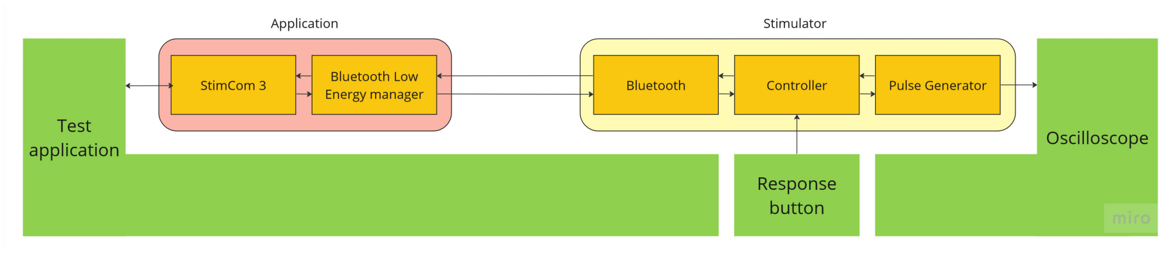
Chapter 4

Testing and Analysis

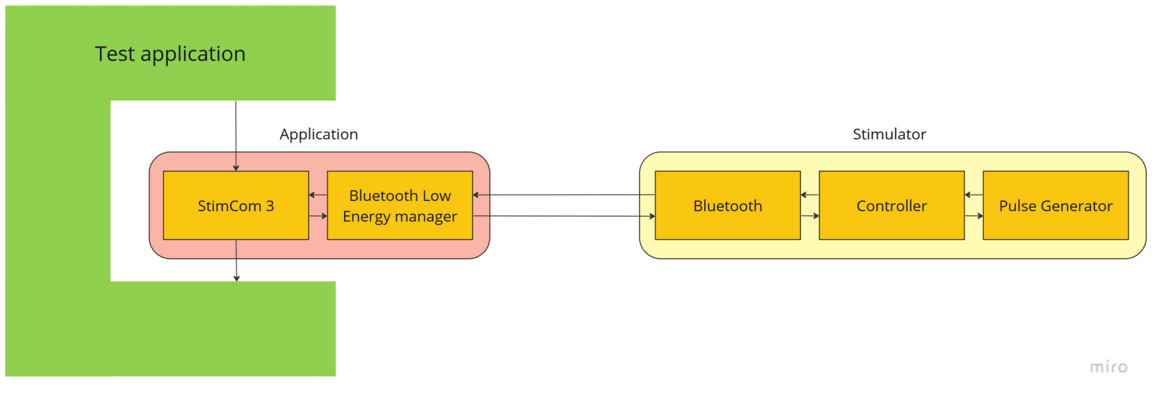
In order to verify the the implementation and performance of the protocol, two testing setups were devised. Both setups involved a test application running on a laptop featuring Bluetooth Low Energy, that provides an interface to conduct the tests, as well as the AmbuStim stimulator, adapted with a BLE module. In order to perform stimuli, the stimulator must be left floating, i.e. not be grounded. As the stimulator is connected to a laptop during the tests through a USB port, the laptop itself must also be left floating. Therefore, it is important to not have a charger connected during testing. As the input impedance of the oscilloscope is very high, it may be connected to the mains voltage without impeding the stimulator.

Networking tests could be conducted using only the test application and the stimulator. For functional tests, an oscilloscope was needed to verify configurations of the pulse parameters. The response button was used to test the button reading functionality. The schematic representation of these tests are given Figure 4.1a and 4.1b. A visualization is displayed in Figure 4.2.

The Bluetooth Low Energy parameters were left at their default, which may differ per device. These parameters were found to be configured to a connection interval of 60 ms, and an MTU of 23 bytes. The same test laptop and stimulator were used in each test mentioned in this chapter.



(a) Functional test



(b) Network test

Figure 4.1: Testing schematic setup

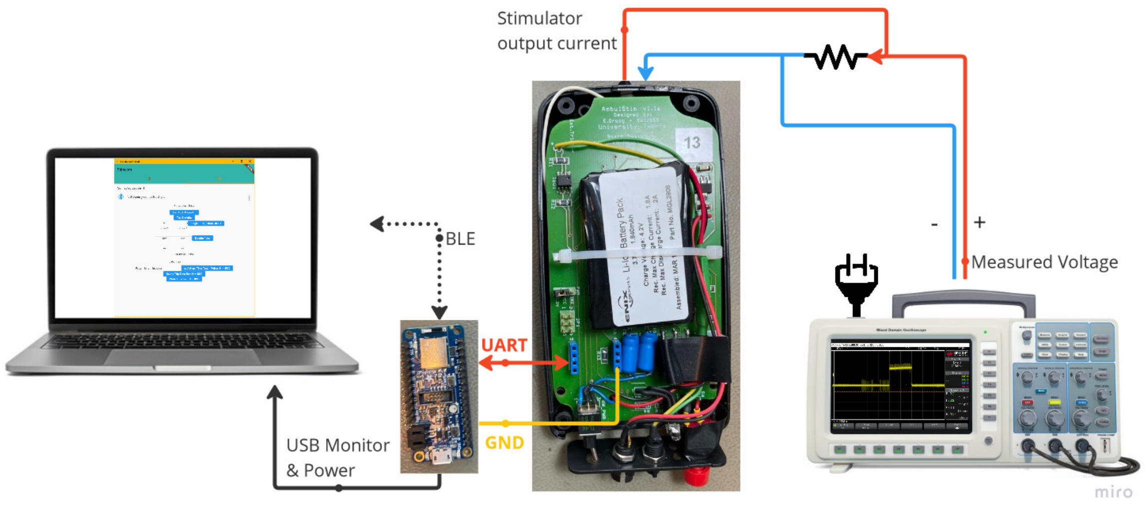


Figure 4.2: Testing Visualization. For functional tests, for network tests, the output resistor and oscilloscope are disconnected.

4.1 Functional Tests

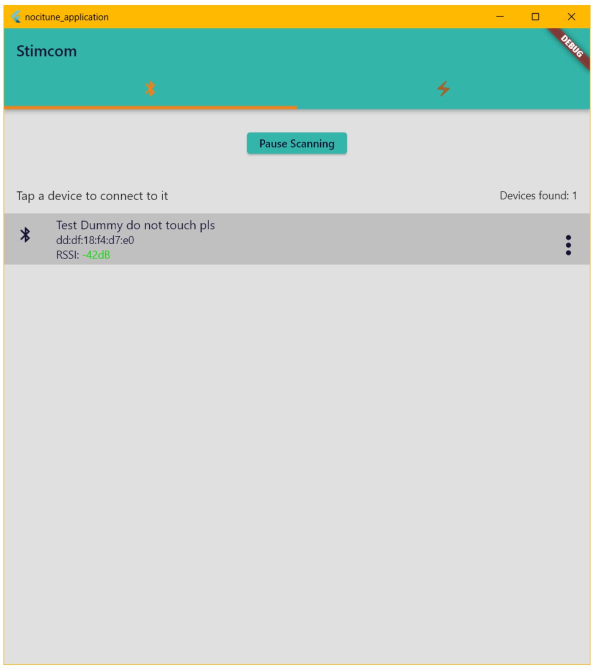
Configuration of a single pulse was tested by entering the desired parameters in the application, and pressing the button to start it. The application will then wait for the response button to be held. Once the response button is pressed the stimulus is applied. The output current of the pulse was then measured. The delay between giving the stimulation command and the detection of the stimulus was not measured. The pulses were measured by loading the stimulator output with a 100 Ohm resistor, and measuring the output voltage. In order to analyse the output current, the conversion simply becomes: $I = V/100$.

The pulse trains that were configured are given in Table 4.1. The resulting scope output images are shown in Figure 4.4.

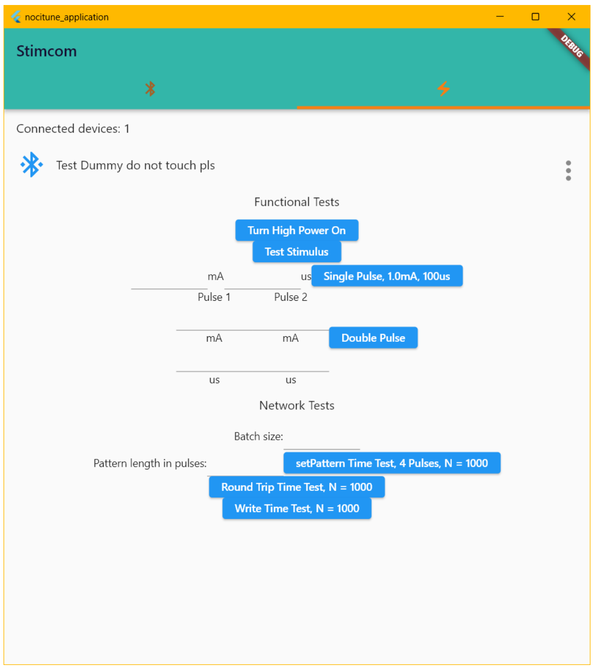
Type	Pulse Amplitude(s) [mA]	Width(s) [us]	Measured Amplitude(s) [mA]	Measured Width(s) [us]
Single	1	1000	1	1000
Single	0.5	1000	0.5	1000
Single	1	2000	1	2000
Double	[1, 0.5]	[1000, 1000]	[1, 0.5]	[1000,1000]

Table 4.1: Tested Pulse trains

As can be seen from the scope images and the table, the measured output currents and pulse widths are identical to those configured in application.

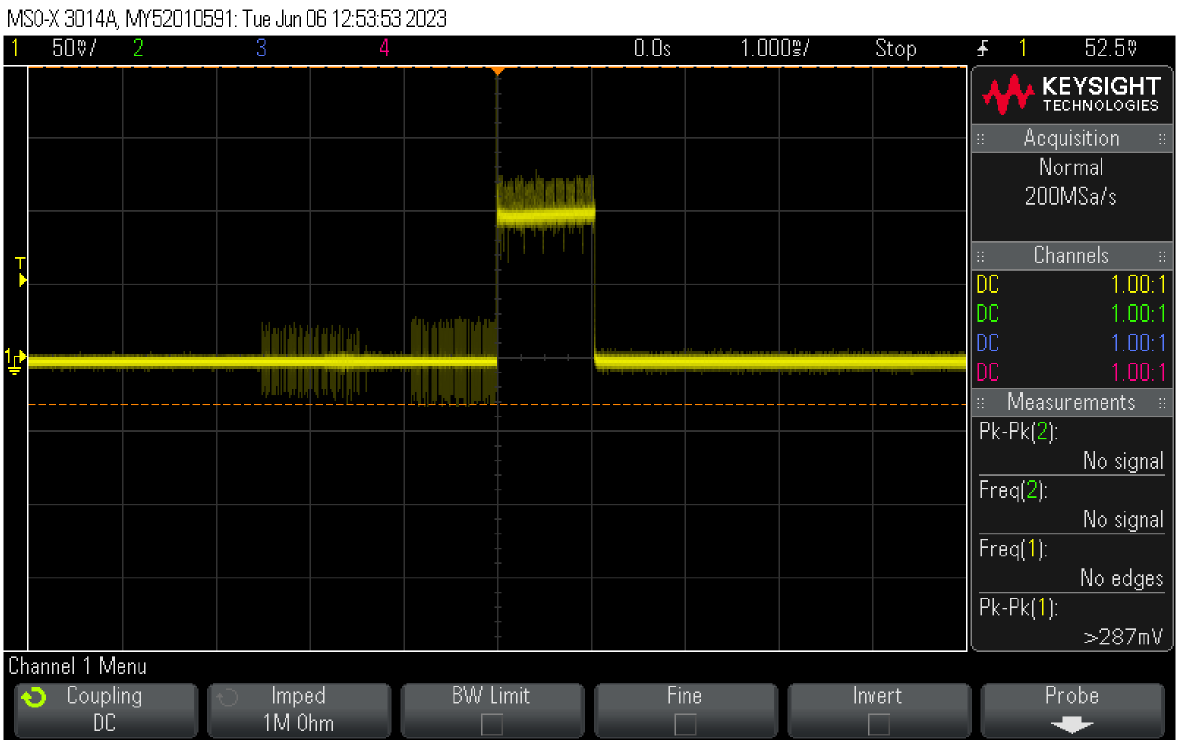


(a)

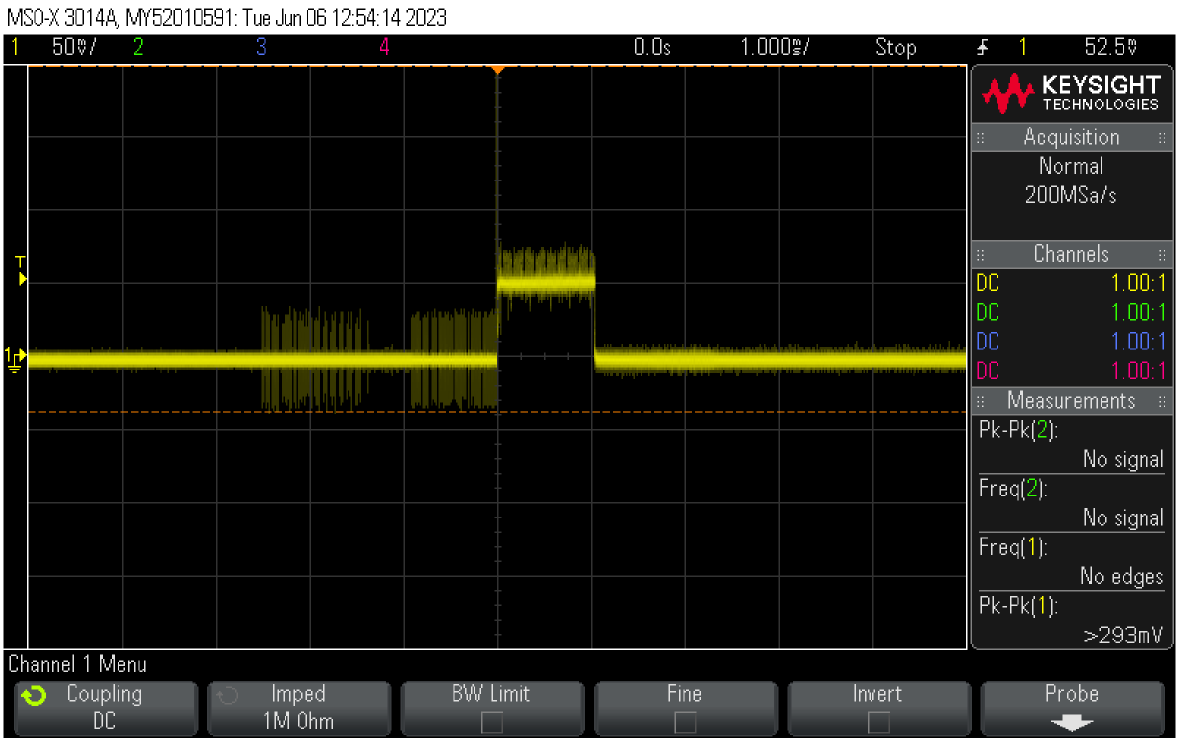

nocitune_application
Stimcom
DEBUG
Connected devices: 1
Test Dummy do not touch pls
Functional Tests
Turn High Power On
Test Stimulus
mA
us
Single Pulse, 1.0mA, 100us
Pulse 1
Pulse 2
mA
mA
Double Pulse
us
us
Network Tests
Batch size:
Pattern length in pulses:
setPattern Time Test, 4 Pulses, N = 1000
Round Trip Time Test, N = 1000
Write Time Test, N = 1000

(b)

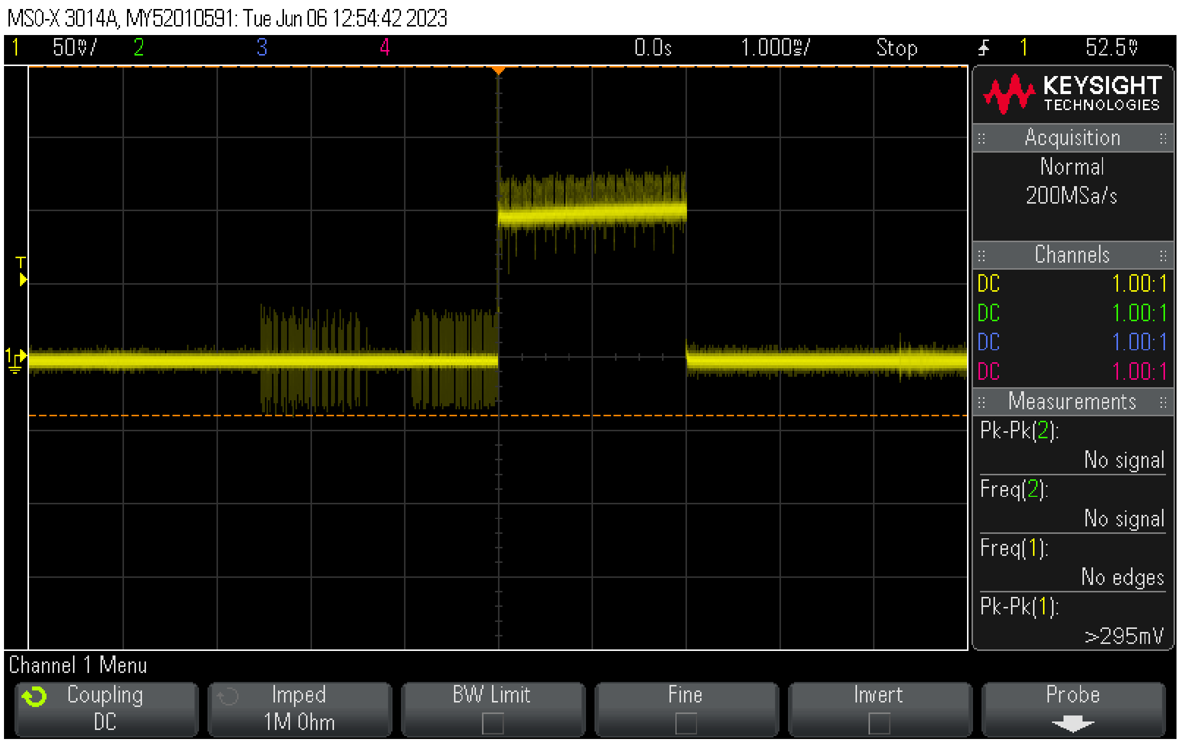
Figure 4.3: The test application used to perform the functional and networking tests. (a) The connection screen, through which StimCom-supporting devices can be discovered an connections can be established. (b) The test page from which all tests can be configured and executed.



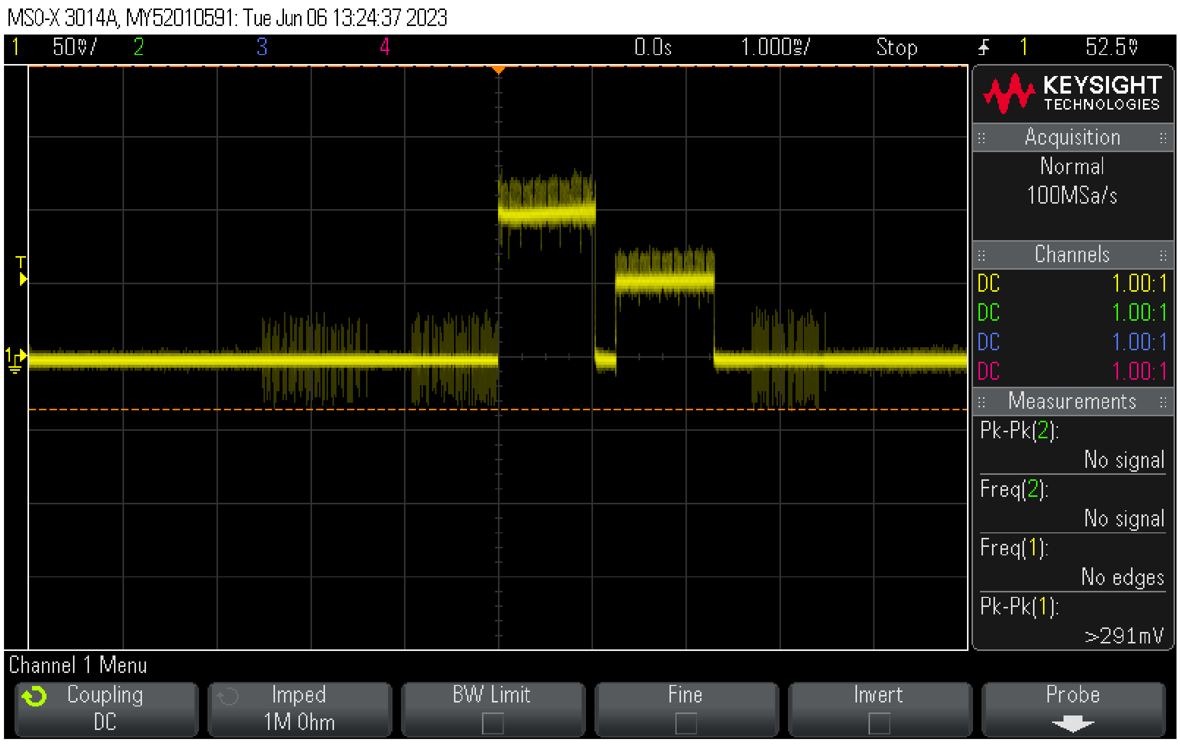
(a) Single pulse, 1 mA, 1000 us



(b) Single pulse, 0.5 mA, 1000 us



(c) Single pulse, 1 mA, 2000 us



(d) Double pulse, 1 mA, 0.5 mA, 1000 us, 1000 us

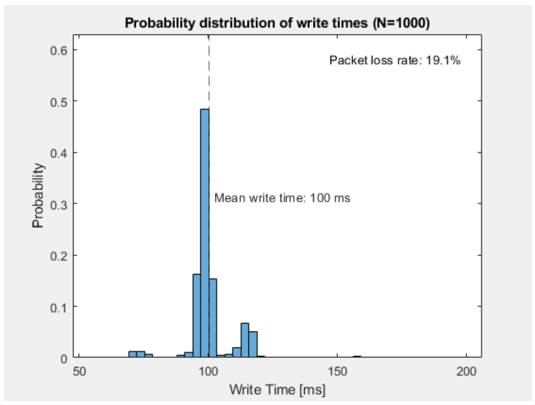
Figure 4.4: Stimulator output scope images. The output amplitude scale was set to 50mV/div, which corresponds with 0.5mA/div. The output time scale was set to 1ms/div.

4.2 Network Tests

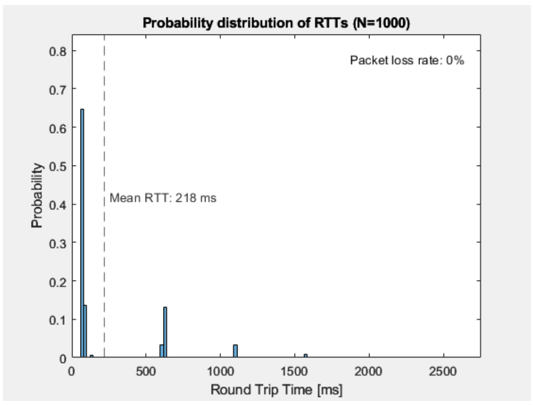
In all network tests, each operation was performed a large number times, and measured using a software timer within the test application. These results were then exported to a `.csv` file and processed and plotted in MATLAB.

The performance of writing operations was measured by initiating a writing operation on the application, and measuring the time for an acknowledgement to be received. Note that this does not include the returning indication. If the time exceeds 500 ms, it is counted as a lost packet.

The performance of the StimCom protocol was evaluated as the Round-trip time (RTT), measured from sending a command to successfully receiving the response indication from the stimulator. The data in the command is irrelevant, as longer messages spanning multiple packets will still arrive within a single connection interval. If after 500 ms no indication has been received, the protocol will send a retry packet. This process repeats up to 10 retries, after which the packet is counted as definitively lost.



(a)



(b)

Figure 4.5: (a) Write time analysis, (b) Round-trip time analysis

In Figure 4.5a, it can be seen that 80.9% of write operations complete successfully, almost all of which arrive between 75 and 120 ms. This is expected, as the time for an acknowledgement to arrive typically takes 1 connection interval, which in this case is 60 ms. The additional delay is likely a semi-random delay due to the messaging system of the operating system, causing a large peak at one particular value. The mean time to complete a write operation is 100 ms, which coincides with the mode, as 49% of operations complete between 97 and 100 ms.

In Figure 4.5b, 100% of operations are successfully completed. The retry mechanism recognisably appears in the distribution, leaving gaps of just over 500 ms in between when operations complete. 79% of operations do not need a retry, 95% of operations complete within 2 attempts, and after 3 attempts 99% of operation have completed. In total, the mean operation time becomes 218 ms.

The main functionality to be tested is the ability to configure and apply pulses through the protocol, so a test was configured in which an array of pulses was configured repeatedly. For single channel stimuli, this results in an queue of 6 commands. For each additional channel used, another command is needed to enable it. In some implementations, only one or two parameters are changed at a time, so not all commands need to be called, leading to better performance. This test gives a good indication of a worst-case scenario, where en entirely new stimulus pattern is configured for every pulse.

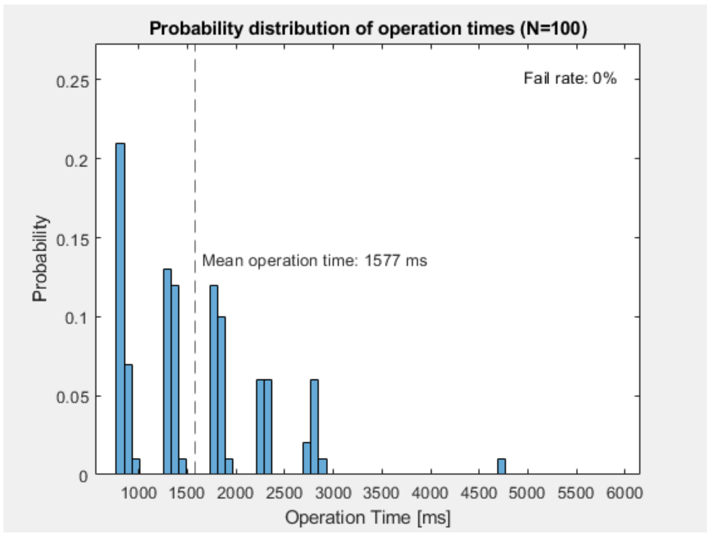


Figure 4.6: Pattern Change time analysis

As can be seen from Figure 4.6, 29% of the time, the pattern is changed within 1 second, 55% of the time within 1.5 seconds, 78% within 2 seconds, 90% within 2.5 seconds, 99% within 3 seconds. Though most operations will finish in this interval, the theoretical maximum amount of time the change pattern operation could take, based on the worst-case results in Figure 4.5b, would be 9.6 seconds.

4.2.1 Performance Analysis

Bandwidth and throughput

Using only write operations, the bandwidth was determined to be 1.84 kbps. If using the command-response protocol, the effective bandwidth is reduced to 0.85 kbps. This extremely limited bandwidth is a direct consequence of the connection parameters. The theoretical maximum bandwidth for write operations, assuming a write takes 1 connection interval, is reached with a connection interval of 7.5 ms and an MTU size of 512 bytes. This configuration reaches a bandwidth of 546.14 kbps, or 68.27 kBps. Another limiting factor may be because the BLE operations are queued through the operating system, which may send messages and call interrupt functions at a delayed time. It is impossible to know how long this delay is in reality, because the lower software layers are completely invisible to the application, as are the underlying mechanisms responsible for the delay. In spite of this, it is reasonable to assume that these delays will never be longer than a few milliseconds.

For the throughput, we can calculate the best-case scenario, where each parameter field has a maximum value of $2^{32} - 1 = 4294967295$ (assuming an 32-bit unsigned integer), which each take up 10 bytes when ASCII-encoded. This means with an MTU size of 23 bytes, 2 parameters of maximum value can be sent in a single operation (2 times 10 bytes per parameter, plus one character in between each parameter), resulting in 20 bytes of data, and a throughput of 87% of the total bandwidth. In the worst case scenario, where each parameter field has a value of 0 and takes up 1 byte when encoded, this results in a throughput of 52%. This means the effective throughput will range anywhere from 0.73 to 0.44 kbps, with an average of 0.59 kbps, not accounting for packet retries.

As demonstrated, the number of pulses than can be configured through StimCom depend directly on the MTU. The maximum number of pulses N_{max} is equal to $N_{max} = \texttt{MTU}/2$. If all pulses are allocated the same amount of space, the value range of each parameter is directly limited by $V_{max} = 10^{\texttt{floor}((\texttt{MTU}-N)/N)} - 1$. If the values of the pulses are significantly different, however, this becomes far more complex. The rule of thumb here becomes: Larger parameter values, less pulses.

As can be seen, the protocol itself is inefficient due to character encoding, and the bandwidth not high. For this application, however, a very high bandwidth is not needed to configure the pulses. The encoding, in conjunction with a limited MTU size, places some seemingly arbitrary limits on the values that can be transferred, which is not ideal.

Latency

The delays measured are not insignificant at 218 ms per operation, resulting in a pattern change delay that frequently takes over 2 seconds. In experiments where only a single parameter needs to be changed, this might not be a problem, as then less commands are needed. While for some experiments this is acceptable but not ideal, some cases where the pulse train is frequently replaced this will be a major bottleneck and a source of inconsistency in the experiment. While it should be expected that wireless connections will have a hiccup from time to time, the severity should be reduced wherever possible. A relatively simple solution would be to reduce the connection interval, meaning operations can occur in faster succession. This might bring some difficulty with implementation as some devices, notably mobile devices, as the operating system will automatically try to optimise the connection parameters to reduce energy consumption. Another would be to decrease the time taken before a retry takes place, though experimentally this has also been found to increase the amount of retries needed.

4.3 Requirements Validation

The implemented StimCom protocol and application only implement a subset of the requirements as specified in Appendix A, System Design Document, sections 5.3.2 and 6.1.2. This validation can be seen in Tables 4.2, 4.3, and 4.4 for the Application-Stimulator Interface, and in Tables 4.5, 4.6, and 4.7 for the Application.

The application interface with the stimulator was fully implemented, of which only ASI-F6 was not met due to poor performance. Of the application, only fundamental requirements, such as a cross-platform implementation, as well as requirements pertaining to the stimulator interface were implemented.

Table 4.2: Application-Stimulator Interface Functional Requirements Comparison

ID	**Description**	✓	**Result**
ASI-F1	The interface must be wireless.	Y	StimCom 3.0 uses Bluetooth Low Energy (BLE), though trigger inputs are stil wired.
ASI-F2	The application is always completely aware over the actions of the stimulator.	Y	All StimCom 3.0 operations are acknowledged and guaranteed to arrive. The stimulator reports back any action it undertakes.
ASI-F3	The application must be able to configure the pulse train on the stimulator.	Y	StimCom 3.0 implements functions which can be used to configure the pulse train on the stimulator.
ASI-F4	Pulse trains are applied when the application gives the appropriate command.	Y	StimCom 3.0 implements a function to either: immediately apply the pulse train, or to wait for an external trigger.
ASI-F5	The application must be able to read the response of a stimulus.	Y	StimCom 3.0 communicates the stimulus response to the application.
ASI-F6	90% of pulse trains configurations must complete in no longer than 1 second.	N	Only 29% of the pulse train configurations complete within 1 second. This requires 0 retries on StimCom commands.
ASI-F7	100% of sent commands must be received by the stimulator.	Y	Through a retry mechanism all commands will eventually be received by the stimulator.

Table 4.3: Application-Stimulator Interface Non-Functional Requirements Comparison

ID	**Description**	✓	**Result**
ASI-NF1	Establishing the connection from the application to the stimulator must complete with 4 seconds after initiation from the application.	Y	With security procedure in place, the connection is established quickly.

Continued on next page →

→ Continued from previous page

ID	Description	✓	Result
ASI-NF2	Using the connection from the application to the stimulator must not require intimate knowledge of the system.	Y	The StimCom library provides a simple call-response interface through which the functionality of the stimulator is made available

Table 4.4: Application-Stimulator Interface Other Requirements Comparison

ID	Description	✓	Result
ASI-O1	The interface must be modular, to allow functionality to be added or altered in future iterations.	Y	The StimCom library underlying logic is simple to adapt to additional functionality.

Table 4.5: Application Functional Requirements Comparison

ID	Description	✓	Result
A-F1	The application must be compatible with Windows desktops and Android and iOS devices.	N	The current application implementation is only available on Windows, but is extensible to Android and iOS.
A-F2	The application could be compatible with Linux and MacOS desktops.	N	The current application implementation is only available on Windows, but could also be extended to support Linux and MacOS.
A-F3	The application must be able to utilize Bluetooth Low Energy in order to communicate with stimulators.	Y	The StimCom protocol is able to use BLE to communicate with stimulators.
A-F4	The application must be able to configure a pulse train on stimulators, and be able to send the command to trigger the stimulation.	Y	Configuring pulse trains and applying stimuli through StimCom is fully possible.
A-F5	The application must be able to register the subject response after a stimulus.	Y	Recording subject responses through StimCom is fully possible.
A-F6	The application must be able to interface with other types of sensors, not part of the NociTRACK system.	N	Not Implemented
A-F7	The application must provide an intuitive interface with other applications, such as MATLAB and Python, in order to allow control over the stimulators.	N	WebSocket interfaces have been tested, but not implemented.

Continued on next page →

→ Continued from previous page

ID	Description	✓	Result
A-F8	The application must be able to load pre-programmed experiment algorithms and parameters for experiments.	N	Not Implemented
A-F9	The application must be able to import and export relevant subject parameters for experiments.	N	Not Implemented
A-F10	Experiments should be able to be aborted at any point by the operator	N	Not Implemented

Table 4.6: Application Non-Functional Requirements Comparison

ID	Description	✓	Result
A-NF1	The application must have an intuitive interface, and enough autonomy to provide operators with little technical knowledge with the ability to perform any experiment with the system.	N	Not Implemented
A-NF2	The connection procedure used to connect to stimulators in a fast and safe way.	N	The connection procedure currently has no security measure taken.
A-NF3	The connection procedure used to connect to stimulators should leave no doubt the operator is connecting to the desired device.	N	The connection procedure currently does not include a way to definitely connect to the correct device.

Table 4.7: Application Other Requirements Comparison

ID	Description	✓	Result
A-O1	The BLE connection should be encrypted.	N	Not Implemented
A-O2	The application should be easily extensible, to allow new functionality and support for other operating systems	Y	The StimCom 3.0 library is dependent on a platform-specific implementation of BLE. This platform-specific implementation was abstracted to unify the interface used by StimCom across platforms. The StimCom 3.0 library itself provides an intuitive and simple interface for higher software layers.

Chapter 5

Conclusion

During this thesis, the objective was to enable a software application to wirelessly communicate with stimulators in a cross-platform way, as a step in the process of enabling experiments to move out of the lab. In order to do this, it was deemed necessary to adapt the existing Bluetooth Classic protocol to Bluetooth Low Energy. To this end, StimCom 3.0 was designed and implemented as a replacement for StimCom 2.1. The protocol implements the same functionality as its predecessor, but opens up new possibilities, such as maintaining a connection with multiple stimulators and lower energy usage. Additionally, a Flutter application with the purpose of controlling stimulators through the StimCom 3.0 protocol, written in Dart, was designed and implemented to work on Windows operating systems. The framework of the application is able to compile cross-platform, though the BLE implementation is still platform specific. Also a vision was formulated for the future functionality of the application for use in research and clinical settings, by creating a background process implementing the essential functionality, and exposing this process to other software processes through WebSockets.

Bluetooth Low Energy was implemented on a stimulator by replacing the module Bluetooth Classic with one supporting BLE. The serial interface with the processor still uses StimCom 2.1, so the module was programmed to convert between the two protocols.

StimCom 3.0 was tested and verified using the developed application. The functionality was fulfilled, though the protocol performance was deemed too slow to be desirable. Individual StimCom operations take 218 ms on average, which resulted in the time to change a pattern to frequently take over 2 seconds. While this might be acceptable for some experiments, for most applications it is too slow. The likely cause for this was the long connection interval, as well as the retry rate of protocol packets. While connecting to multiple peripherals is possible, performing stimuli with multiple stimulators has not been tested.

Chapter 6

Discussion

The connection interval is the greatest contributor to the poor performance of the the protocol. These parameters need to be configured by the central device, as it acts as an absolute authority operations occurring on the connection, as well as the technical parameters that are used while a connection is established. Access to these parameters is highly platform dependent, and they not always easily accessible to developers. The Dart software library used to create the BLE library does not have access to the connection interval and MTU size of the connection. Instead these parameters are left to the default of the Windows operating system, which was found to be 60 ms. If these connection parameters need to be accessed, some way to access these parameters must be made available. Alternatively, the peripheral can be configured to not support connection intervals above a certain value. The connection will then only by accepted if the central also supports these values (Bluetooth Core Specification, v5.3 [1], Vol 3, Part C, Section 9.3.9)

Some indications sent by the stimulator receive an acknowledgement from the application, but they do not reach the application layer. Missing these indications triggers the retry mechanism, though the operation was carried out successfully. Aside from slowing down the protocol, this has a side-effect of unintentionally executing a command an additional time. For most commands this only degrades performance, but for the stimulation command it might trigger an additional stimulus.

Currently all data in BLE packets is encoded as ASCII characters, which in every cases requires more bytes than the original data, and has a variable length depending on the original data. This means that depending on the values to configure, less data can be transmitted. On top of being highly inefficient, variable payload length is more efficiently implemented through a length field, than explicit separators.

In the current implementation, mesh networking is not part of the protocol. Because of this, communication between stimulators, or with multiple simulators, is not possible yet. The functional performance between a controller and the stimulator is identical to the StimCom 2.1 protocol, which is also a bottleneck for adding additional functionality. This is inherent to the system, as currently the BLE module is an interface between the StimCom 2.1 and 3.0 protocols.

It is also important also to mention that the write-indication structure used is another limiting factor on performance, as the controller is forced to wait until the stimulator sends what is essentially a secondary acknowledgement of the original message. In the original serial Bluetooth Classic protocol this was necessary to verify data integrity, as Bluetooth Classic does not have an integrated CRC check. BLE however, has a CRC check on each packet. This means that when the acknowledgement is received, the data is already guaranteed to be intact. This means the only function of waiting on the indication is to verify that the sent parameters are valid for the stimulator. It would speed up the connection significantly to initially trust that the stimulator can handle the parameters, and continue sending commands. Then when a command is invalid on the stimulator, the stimulator can request or suggest that the controller change the parameter to its preferred value. If all commands have been sent, and no indications have been received within some time frame, e.g. 2 connection intervals, or some explicit verification command, the procedure is considered complete. This allows for faster batch-processing of commands. If order of commands is relevant to StimCom 2.1, this is expected to be handled by the BLE module.

With StimCom 2.1, there were sometimes issues that the connection between the central and peripheral would randomly drop out during usage. These problems have not been experienced with StimCom 3.0. This could be due to the change in Bluetooth module, or the change to Bluetooth Low Energy. Another cause could be the use of fully acknowledged operations, as in StimCom 2.1, the connection is rejected by the protocol when the stimulator does not respond to a message, though Bluetooth still accepts this. In BLE, this is left to the platform-implementation, as not acknowledging an operation is automatic reason for a disconnect by the Low Energy Link Layer (Bluetooth Core Specification, v5.3 [1], Vol 6, Part B, Section 4.5.2).

The application currently only has a build available for Windows, as the BLE layer for other operating systems is not implemented. It is not ideal to require platform-specific implementation for each platform. There are, however, large differences in the Bluetooth implementations between Windows, iOS and Android in what permissions are required, how various subsystems are accessed, and how connections are managed. The lack of an already available cross-platform BLE library in Dart means a custom implementation is necessary, at least until such a library becomes available. As PC-mobile cross-platform is becoming a more desired, and Flutter gains popularity, this is expected to happen within the coming years.

Chapter 7

Recommendations

Future development efforts with respect to the application should be put towards improving the performance and reliability on the existing Windows implementation. On expanding support to include Android and iOS, the advice is to wait some time and see if a cross-platform BLE library becomes available, at least across Windows, Android and iOS, as this would save much development effort.

The next logical step for the software application would be to implement the WebSocket interface on on a background process. The WebSocket interface already has a small working demo setup using a Dart Isolate (a sort of green thread) available for establishing the connection, and has been confirmed to work with Python and MATLAB on the same machine. This would involve working out the interface suggested in Section 3.5.1.

After the WebSocket interface is completed, more functionality can be added to the application, such as implementing various algorithms on the Dart WebSocket client, and making data processing and storage logic available on the WebSocket server.

In order to improve the performance of the StimCom 3.0 protocol the advice is to try and decrease the connection interval to a more practical value. This is subject to the connection parameter requirements posed by the platform the peripheral is connecting to. For instance, Apple requires BLE peripheral devices to have a connection interval of a multiple of 15 ms [12]. These might also affect procedures, such as time synchronization between stimulators and the application.

While the ASCII-encoding of the data and dynamically separating parameters with commas might make verifying and converting commands easier, it is highly inefficient. Transferring the data as predefined raw data structures would remove much overhead and allow much more data to be transferred more compactly. This would, however change much of the protocol interface libraries as they are currently built on the assumption that the data can be processed using string libraries, instead of raw data. Because most data still fits inside BLE packets, this should not pose much of a problem until it becomes desired to transmit larger amounts of data per packet.

The current implementation has programmed over a USB connection using the Arduino environment. In the new hardware implementation this USB interface is not available, and it should be programmed over a J-Link interface. The compiled Arduino binary files should also be suitable to upload through this J-Link, though this has not been tested.

It is therefore highly recommended to rewrite the firmware on the nRF52840 to its native functions, instead of Arduino is an arduous task. but will greatly reduce the overhead and bloat included in the program binaries due to Arduino and enable debugging, which greatly enhances the development process.

The last recommendation pertains to the documentation that was generated during this thesis. Much of the requirements and implementation detail of the other subsystems have remained fuzzy. It is strongly advised to iterate over these documents as more details, requirements, and design documents are generated. Maintaining a complete database of the latest documentation, as well as an archive of previous versions will greatly improve designers and engineers to work with the existing system.

References

[1] Bluetooth Special Interest Group, "Bluetooth Core 5.3 Specification," 2021.

[2] A. H. Poulsen, J. Tigerholm, S. Meijs, O. K. Andersen, and C. D. Mørch, "Comparison of existing electrode designs for preferential activation of cutaneous nociceptors," *Journal of Neural Engineering*, vol. 17, no. 3, 6 2020.

[3] B. v. d. Berg, "Combined Psychophysical and Neurophysiological Tools for Mechanism-Based Observation of Impaired Nociceptive Processing," Ph.D. dissertation, University of Twente, Enschede, The Netherlands, 4 2022.

[4] L. Vieira, L. Nissen, G. Sela, Y. Amara, and V. Fonseca, "Reducing Postoperative Pain from Tonsillectomy Using Monopolar Electrocautery by Cooling the Oropharynx," *International Archives of Otorhinolaryngology*, vol. 18, no. 02, pp. 155–158, 2014.

[5] N. B. Finnerup, R. Kuner, and T. S. Jensen, "Neuropathic pain: From mechanisms to treatment," *Physiological Reviews*, vol. 101, no. 1, pp. 259–301, 1 2021.

[6] F. A. Kingdom and N. Prins, *Psychophysics: A Practical Introduction.* Elsevier, 1 2016.

[7] T. N. Cornsweet, "The Staircase-Method in Psychophysics," *Source: The American Journal of Psychology*, vol. 75, no. 3, pp. 485–491, 1962.

[8] "Write once, run anywhere?" [Online]. Available: https://web.archive.org/web/20120728221058/https://www.computerweekly.com/feature/Write-once-run-anywhere

[9] "React Native - Learn once, write everywhere." [Online]. Available: https://reactnative.dev/

[10] "Flutter - build apps for any screen." [Online]. Available: https://flutter.dev/

[11] "Xamarin — Open-source mobile app platform for .NET." [Online]. Available: https://dotnet.microsoft.com/en-us/apps/xamarin

[12] A. Inc, "Accessory Design Guidelines for Apple Devices Release R20," 2022.

Appendix A

System Design Document

NociTrack
SYSTEM DESIGN DOCUMENT

Version 0.3

Prepared by :
ing. Henry Slegers
ir. Frodo Muijzer
dr. ir. Jan Buitenweg

Submitted :
July 6, 2023

Contents

1. Version Control

Version	Date	Notes
0.1	2023/04/18	Initial Version
0.2	2023/06/23	Changed Application Terminology
0.3	2023/07/06	Updated System Figures

2. Introduction

2.1. Purpose

This System Design Document (SDD) describes how the functional and nonfunctional requirements recorded in the System Requirements Document (SRD) are transformed into more technical design specifications from which the system will be built. The SDD will document the high-level functional breakdown, connections and total architecture of the system. Of each component, the intended purpose, behaviour, and requirements will be given. This document serves as a point of reference, and a resource to developers and testers to aid in the design and verification of the various subsystems of the NociTrack system.

2.2. Intended Audience

This SDD is intended for project managers, developers and testers.

3. System Overview

A functional description of the system.

The NociTrack system can be used to determine nociceptive detection thresholds for the purpose of chronic pain research, and general understanding of the nociceptive nervous system.

It does this by applying a current to the nociceptive nerve cluster in a subject, and recognising the presence of, or lackthereof, detection of this stimulus in the subject.

4. Design considerations

Design issues that should be solved before the design can be completed.

4.1. Design constraints

Global limitations that strongly impact the design of any hard-/software within the system. E.g. processing hardware, user environment, standards/certifications, performance, compatibility.

- Must meet the Medical Device Regulation (MDR).
- Must comply with Class IIa medical device regulation.
- Device must be operable in a medical laboratory.
- Device must be operable in a household by medical staff without needing expertise of the nociceptive nervous system.
- Experiments must take an appropriate amount of time for subjects with heightened sensitivity, to not put unnecessary strain on them.
- The system may not locally store personal information about subjects.
- The device must be easily stored in an office drawer when not in usage.
- The device must be easily transported.
- External devices must be able to synchronise with applied stimuli.

4.2. Goals and Guidelines

Design goals and methods that should be employed in the design. E.g. coding/design conventions, version control, COTS product usage, custom produced components, etc.

- Wherever possible, version control should be used, e.g. through Git.
- For sensor/actuator hardware, custom designs are needed. Other systems preferably use already available systems.
- The system should preferably not be dependent on other paid licensing, such as MATLAB.

5. System Architecture

On a global functional level, show the subsystems below here and how they relate to each other.

5.1. Functional overview

On a functional level, the Nocitrack system can be broken up into a controller, algorithm, data processing/storage, and stimulator. Optionally a trigger generator and (external) sensors can be used, but are not essential for system operation.

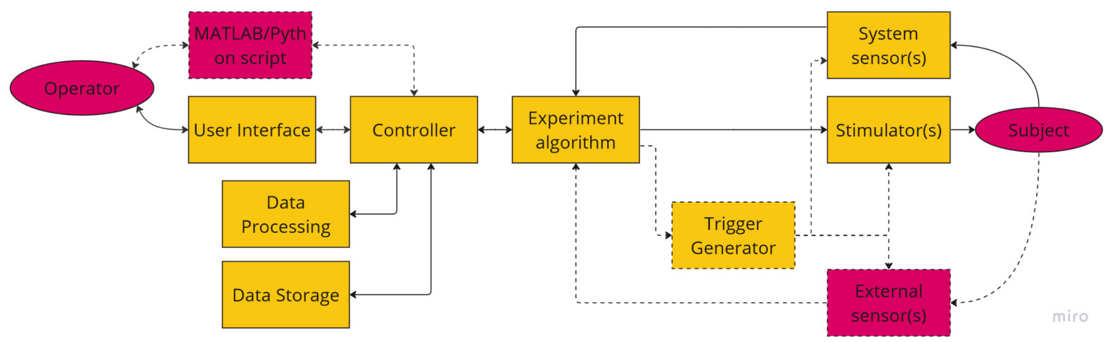


Figure 5.1.: Functional Breakdown

5.1.1. Controller

The controller provides the operator with an interface to control and monitor the experiment from. It provides the operator with the ability to:

1. implement and configure algorithms for experiments.
2. connect to one or more stimulators
3. monitor an ongoing experiment.
4. fetch and store results and parameters of an experiment.
5. analyse the results of one or more experiments after the fact.
6. the ability to expose these functions for use by an external MATLAB/LabVIEW/Python program.

5.1.2. User Interface

Frontend software that allows an operator to utilise the underlying software.

5.1.3. Data processing/storage

Data storage contains the parameters needed to run an experiment, and provides space to store the results of an experiment. Data processing provides methods to analyse the results of one or multiple experiments.

5.1.4. Experiment algorithm

The algorithm is the prominent system element during an ongoing experiment. First an experiment must be configured and started. Afterwards, the algorithm will configure the stimulator, using feedback from the subject, to generate the correct pulses. It will terminate when a condition has been met, or if terminated manually. During the experiment it will inform the controller of the status of the experiment, and feed back the experimental results.

5.1.5. Stimulator

The stimulator can be configured by the experiment algorithm. It can generate the stimulus either when given the command, or when a trigger pulse has been received. The stimulus is then applied to the subject.

5.1.6. System sensors

These are sensors that are part of the NociTrack system. Measurements are fed into the experiment algorithm.

5.1.7. External sensors

These are sensors that are not part of the NociTrack system, but may optionally be connected if required for a particular experiment. Measurements are fed into the experiment algorithm.

5.2. Subsystems

In this section, the subsystems are compiled from the functional breakdown.

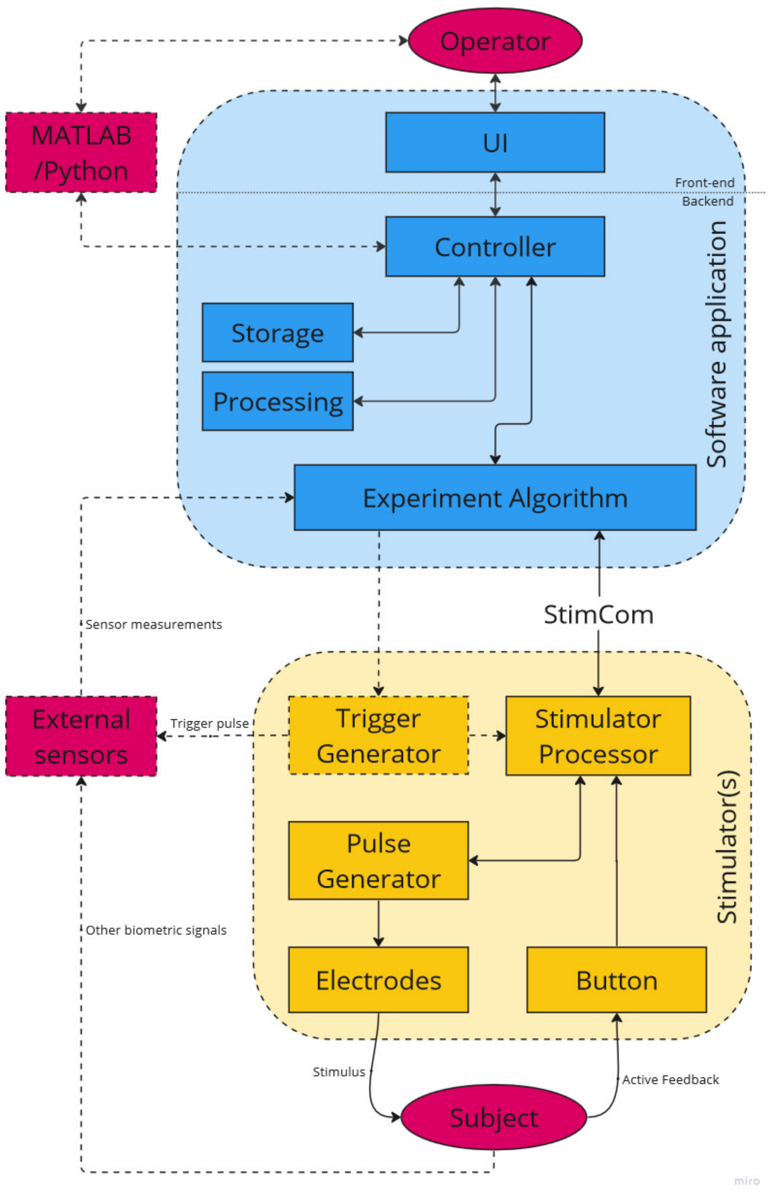


Figure 5.2.: System Architecture

The controller and experiment algorithm functionality are merged together into a single controller subsystem, as this functionality will likely exist on the same hardware. The data processing/storage may also be integrated, as this functionality does not always have to be a separate subsystem, e.g. in portable applications, when a network connection is not available, or in simple experiments. Otherwise this capability is offloaded to a server. The stimulator is split into the stimulator proper, and its electrodes. This is useful because different stimuli patterns and location might require different electrodes, and multiple electrodes may be connected to a single stimulator.

5.3. Application

5.3.1. Purpose and Behaviour

The application is a computer program that may run on a personal computer, a mobile phone, or tablet. The application can connect to one or more stimulators, as well as a trigger generator. It can also connect to a server with data storage and processing power over the internet. From the application, the operator can select an experiment, which is compatible with the connected devices. Through these experiments, stimuli can be applied using the connected stimulators. Responses are recorded using the active feedback from stimulators, or from other external sensors. The application feeds the status of the experiment back to the operator, as well as providing them the ability to interact with the experiment, e.g. to manually terminate the experiment.

The application must be electrically isolated from the stimulators and sensors. For this purpose, the communication between the application and stimulators is wireless.

5.3.2. Requirements

Table 5.1.: Application Functional Requirements Table

ID	Description	Reasoning
A-F1	The application must be compatible with Windows desktops and Android and iOS devices.	These platforms cover a large part of operating systems on devices used by medical personell.
A-F2	The application could be compatible with Linux and MacOS desktops.	These platforms, while not mandatory to have, would expand the number of devices that are able to run the application.
A-F3	The application must be able to utilize Bluetooth Low Energy in order to communicate with stimulators.	BLE is mandatory in order to connect to stimulators and conduct experiments.

Continued on next page →

→ Continued from previous page

ID	**Description**	**Reasoning**
A-F4	The application must be able to configure a pulse train on stimulators, and be able to send the command to trigger the stimulation.	Configuring and giving stimuli is mandatory in order to conduct experiments.
A-F5	The application must be able to register the subject response after a stimulus.	Recording subject responses is mandatory in order to conduct experiments.
A-F6	The application must be able to interface with other types of sensors, not part of the NociTRACK system.	By providing interfaces to external sensor systems, many other types of experiments are able to be conducted. The sensors with which the system should be able to connect with, depends on the desired experiments.
A-F7	The application must provide an intuitive interface with other applications, such as MATLAB and Python, in order to allow control over the stimulators.	MATLAB and Python provide many functions which allow for rapid prototyping of experiments. This functionality is essential for researchers developing new algorithms and experiments.
A-F8	The application must be able to load pre-programmed experiment algorithms and parameters for experiments.	Developed experiments should be able to be permanently registered by the application, and reproduced, without need for third party applications, such as MATLAB.
A-F9	The application must be able to import and export relevant subject parameters for experiments.	Experiment parameters with differ based on the type of experiment and between subjects. These should be easily stored and recalled for the operator.
A-F10	Experiments should be able to be aborted at any point by the operator	In case an experiment must be cancelled, e.g. in case of an emergency, the program should be able to be aborted.

Table 5.2.: Application Non-Functional Requirements Table

ID	Description	Reasoning
A-NF1	The application must have an intuitive interface, and enough autonomy to provide operators with little technical knowledge with the ability to perform any experiment with the system.	The enables as many people as be able to conduct experiments, thus enabling large scale testing.
A-NF2	The connection procedure used to connect to stimulators in a fast and safe way.	Fast connecting makes setting up an experiment less laborious, and is generally considered safer.
A-NF3	The connection procedure used to connect to stimulators should leave no doubt the operator is connecting to the desired device.	When many stimulators are present, each should be distiguishable in the application to prevent accidental wrong connections.

Table 5.3.: Application Other Requirements Table

ID	Description	Reasoning
A-O1	The BLE connection should be encrypted.	Connections with stimulators should be secure.
A-O2	The application should be easily extensible, to allow new functionality and support for other operating systems	The development team of the NociTRACK system is frequently changing. The application should be developed in such a way that incremental development is easily possible for new developers.

5.4. Stimulator

5.4.1. Purpose and Behaviour

The stimulator is a portable embedded device which can apply a stimulus using stimulation electrodes. The desired stimulus pattern can be configured wirelessly. The stimulus is applied either when the command is received, or when a wired trigger pulse has been received. Incoming trigger pulses are electrically isolated from the stimulation hardware.

5.4.2. Requirements

Table 5.4.: Stimulator Functional Requirements Table

ID	**Description**	**Reasoning**
S-F1	During continuous usage, the stimulator must remain operable without maintenance for at least 4 hours.	The stimulator should be operable for at least half a working day before needing minor maintenance, such as recharging or replacing single-use components.
S-F2	The maximum positive output current should be less or equal than 50 mA.	Overstimulation may lead to skin burn, muscle contraction, and uncontrolled movements.
S-F3	The minimum negative output current should be more or equal than -50 mA.	See S-F2.
S-F4	When not delivering a stimulus, the output current must be 0 mA.	The stimulus output may not be active stimulation is inactive.
S-F5	The stimulator must be electrically isolated from any external electrical power source.	There may never be the option of the subject to be connected to a live wire due to a system failure.
S-F6	The stimuli that can be given must be configurable from an external device.	The stimulation parameters must be controllable for researchers and experimenters.
S-F7	The configured stimulus will be applied to the output when a trigger input signal or command is given.	The moment of stimulation must be controllable for researchers and experimenters.
S-F8	A trigger output signal will be generated when a stimulus has been applied.	External sensing systems must be able to synchronize with the moment of stimulus.
S-F9	The stimulation must be generated by an internal current source.	External power sources are forbidden.
S-F10	A stimulator may have one or multiple output stimulation channels	Depending on the stimulator design, multiple outputs may be desired, though one at least one output is required.
S-F11	Stimulation channels may not experience cross-talk from other simulation channels higher than 1%	Stimulation through one channel may not significantly impact other channels.

Table 5.5.: Stimulator Non-Functional Requirements Table

ID	**Description**	**Reasoning**
S-NF1	On the outside of the stimulator, there must be an indication of whether the device is turned on or off.	The device must inform the users when it is turned on.
S-NF2	On the outside of the stimulator, there must be an indication of the state of the Bluetooth connection.	To verify the state of the wireless connection, an indication is required
S-NF3	While the battery of the stimulator is charging, the device must electrically disconnect from the battery.	The subject may never be directly or indirectly in contact with a mains voltage.

Table 5.6.: Stimulator Other Requirements Table

ID	**Description**	**Reasoning**
S-O1	...	...

5.5. Trigger Generator

5.6. Stimulator Electrodes

5.7. Server

6. Subystem Interfaces

Interfaces between subsystems of the system. E.g. communication protocols, connector pinouts, mechanical interactions, etc.

6.1. Application-Stimulator

6.1.1. Description

The Application-Stimulator is the channel through which the application establishes a connection, configures and activates pulse trains, and reads the active feedback from the subject. This interface is implemented through the StimCom protocol.

6.1.2. Requirements

Table 6.1.: Application-Stimulator Interface Functional Requirements Table

ID	**Description**	**Reasoning**
ASI-F1	The interface must be wireless.	The stimulator may not be electrically coupled to other systems.
ASI-F2	The application is always completely aware over the actions of the stimulator.	The stimulator must report its actions to the application, and may not take independent action.
ASI-F3	The application must be able to configure the pulse train on the stimulator.	Pulse trains must be configurable.
ASI-F4	Pulse trains are applied when the application gives the appropriate command.	Stimulators may not perform stimuli autonomously.
ASI-F5	The application must be able to read the response of a stimulus.	After a stimulus, the response should be available.
ASI-F6	90% of pulse trains configurations must complete in no longer than 1 second.	To be able to deliver new pulses quickly enough for experiments, configuring a pulse may not take too long.
ASI-F7	100% of sent commands must be received by the stimulator.	The stimulator must reliably receive packets sent by the application.

Table 6.2.: Application-Stimulator Interface Non-Functional Requirements Table

ID	Description	Reasoning
ASI-NF1	Establishing the connection from the application to the stimulator must complete with 4 seconds after initiation from the application.	A fast connection procedure results in smoother setup for experiments
ASI-NF2	Using the connection from the application to the stimulator must not require intimate knowledge of the system.	Minimal technical knowledge should be required from the operator

Table 6.3.: Application-Stimulator Interface Other Requirements Table

ID	Description	Reasoning
ASI-O1	The interface must be modular, to allow functionality to be added or altered in future iterations.	The interface must be prepared to handle functional changes in both the application and stimulator, without needing a complete redesign.

7. System Interfaces

Interfaces with external entities that interact with the system

7.1. Human-Machine Interfaces

Buttons, stimulators, sensors, screens, microphones, etc. What is their role, and what are expected interactions.

7.2. External Interfaces

Hard- or software (power connections, ethernet, laptops, internet servers, wifi)

8. Operational Scenarios

Describe projected real-world scenarios, and detail system usage and intended behaviour.

9. References

A. Abbreviations

Abbr.	Abbreviation
SRD	System Requirements Document
SDD	System Design Document

B. Requirement Tracing

Relate requirements in SRD to requirements in SDD

Appendix B

StimCom Serial Documentation

StimCom Serial DOCUMENTATION

Version 1.0
by :
ing. Henry Slegers

Submitted :
July 6, 2023

Contents

1 Version Control

Version	Date	Notes
0.1	2023/03/23	Initial Version
1.0	2023/07/06	Updated command descriptions

2 Introduction

2.1 Purpose

The StimCom Serial Documentation describes the UART serial communication protocol between a Bluetooth chipset (or other controlling peripheral) and the central processor of a stimulator.

2.2 Intended Audience

This documentation is intended for developers and testers.

3 Protocol

3.1 Introduction

The protocol consists of a set of string commands called from the controlling unit. Each command is sent as a package with a single character header. The command header and arguments are separated by comma's, and the command is terminated by the `\0` character.

After the stimulator has received the a command from the controller, it will check the validity of the parameters. Parameters are considered valid if the stimulator is able to execute the command with the given parameters. If all parameters are valid, it will respond with an identical command. Invalid parameters will be adjusted, and the response will contain the adjusted parameters. The stimulator will then execute the adjusted command.

The data in the fields is formatted as ASCII character strings. e.g. an unsigned integer with the value `100` is encoded as the ASCII characters `{ 0x31 , 0x30 , 0x30 }`

3.2 Commands

3.2.1 Version Query

Requests the version and serial number of the embedded software on the device.

Field	Format	Description
Header	character	V (0x56)
1	unsigned integer	0 (0x30)
2	unsigned integer	0 (0x30)
3	unsigned integer	0 (0x30)

Table 3.1: Command

Example: `V,0,0,0\0`

Field	Format	Description
Header	character	V (0x56)
1	unsigned integer	Major Version
2	unsigned integer	Minor Version
3	unsigned integer	Serial Number

Table 3.2: Response

Example: `V,1,0,27\0` denotes that the device with serial number 27 has an embedded software version of 1.0.

3.2.2 Feature Query

Requests the hardware features of the device.

Field	Format	Description
Header	character	F (0x46)
1	unsigned integer	0 (0x30)
2	unsigned integer	0 (0x30)
3	unsigned integer	0 (0x30)
4	unsigned integer	0 (0x30)

Table 3.3: Command

Example: `F,0,0,0,0\0`

Field	Format	Description
Header	character	F (0x46)
1	unsigned integer	Number of Channels
2	unsigned integer	Maximum Pattern Length
3	unsigned integer	DAC Calibration value [ADunits-to-mA]
4	unsigned integer	Timer Calibration value [Timerunits-to-ms]

Table 3.4: Response

Example: `F,1,20,80,35\0` denotes that the device has 1 stimulation channel, can store stimulation patterns up to 20 stimuli, has a DAC Calibration value of `80 ADunits/mA`, and has a timer calibration value of `35 Timerunits/ms`

3.2.3 Interval Configuration

Contains the length of interval between pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Field	Format	Description
Header	character	I (0x49)
n	unsigned integer	Interval Length [Timerunits]

Table 3.5: Command

Example: `I,40,40,10,10\0` refers to a pulse train consisting of 4 pulses, after the first there is a delay of 40 Timerunits, after the second, 40 Timerunits, after the third, 10 Timerunits, and after the fourth, 10 Timerunits.

Field	Format	Description
Header	character	I (0x49)
n	unsigned integer	Interval Length [Timerunits]

Table 3.6: Response

Example: `I,40,40,10,10\0` refers to a pulse train consisting of 4 pulses, after the first there is a delay of 40 Timerunits, after the second, 40 Timerunits, after the third, 10 Timerunits, and after the fourth, 10 Timerunits.

3.2.4 Pulse Channel Configuration

Contains the channels on which the pulses of the pulse train will occur, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Field	Format	Description
Header	character	P (0x50)
n	unsigned integer	Pulse Amplitude [ADunits]

Table 3.7: Command

Example: `P,1,1,2\0` refers to a pulse train consisting of 3 pulses, the first and second of which will occur on channel 1 of the stimulator. The third occurs on channel 2.

Field	Format	Description
Header	character	P (0x50)
n	unsigned integer	Pulse Amplitude [ADunits]

Table 3.8: Response

Example: `P,1,1,2\0` refers to a pulse train consisting of 3 pulses, the first and second of which will occur on channel 1 of the stimulator. The third occurs on channel 2.

3.2.5 Positive Pulse Amplitude Configuration

Contains the pulse amplitudes of the positive pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Field	Format	Description
Header	character	A (0x41)
n	unsigned integer	Pulse Amplitude [ADunits]

Table 3.9: Command

Example: `A,40,30,20,10\0` refers to a pulse train consisting of 4 pulses, the first of which has an amplitude of 40 ADunits, the second of 30 ADunits, third of 20 ADunits, and the fourth of 10 ADunits.

Field	Format	Description
Header	character	A (0x41)
n	unsigned integer	Pulse Amplitude [ADunits]

Table 3.10: Response

Example: `A,40,30,20,10\0` refers to a pulse train consisting of 4 pulses, the first of which has an amplitude of 40 ADunits, the second of 30 ADunits, third of 20 ADunits, and the fourth of 10 ADunits.

3.2.6 Negative Pulse Amplitude Configuration

Contains the pulse amplitudes of the negative pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Field	Format	Description
Header	character	a (0x61)
n	unsigned integer	Pulse Amplitude [ADunits]

Table 3.11: Command

Example: `a,40,30,20,10\0` refers to a pulse train consisting of 4 pulses, the first of which has an amplitude of -40 ADunits, the second of -30 ADunits, third of -20 ADunits, and the fourth of -10 ADunits.

Field	Format	Description
Header	character	a (0x61)
n	unsigned integer	Pulse Amplitude [ADunits]

Table 3.12: Response

Example: `a,40,30,20,10\0` refers to a pulse train consisting of 4 pulses, the first of which has an amplitude of -40 ADunits, the second of -30 ADunits, third of -20 ADunits, and the fourth of -10 ADunits.

3.2.7 Positive Pulse Width Configuration

Contains the pulse widths of the positive pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Field	Format	Description
Header	character	W (0x57)
n	unsigned integer	Pulse Width [Timerunits]

Table 3.13: Command

Example: `W,10,20\0` refers to a pulse train consisting of 2 pulses, the first of which has an positive width of 10 Timerunits, the second of 20 Timerunits.

Field	Format	Description
Header	character	W (0x57)
n	unsigned integer	Pulse Width [Timerunits]

Table 3.14: Response

Example: `W,10,20\0` refers to a pulse train consisting of 2 pulses, the first of which has an positive width of 10 Timerunits, the second of 20 Timerunits.

3.2.8 Negative Pulse Width Configuration

Contains the pulse widths of the negative pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Field	Format	Description
Header	character	w (0x77)
n	unsigned integer	Pulse Width [Timerunits]

Table 3.15: Command

Example: `w,10,20\0` refers to a pulse train consisting of 2 pulses, the first of which has an positive width of 10 Timerunits, the second of 20 Timerunits.

Field	Format	Description
Header	character	w (0x77)
n	unsigned integer	Pulse Width [Timerunits]

Table 3.16: Response

Example: `w,10,20\0` refers to a pulse train consisting of 2 pulses, the first of which has an positive width of 10 Timerunits, the second of 20 Timerunits.

3.2.9 Channel Enable Configuration

Command used to disable or enable a channel on the stimulator. Positive and negative pulses can be disabled independently.

Field	Format	Description
Header	character	C (0x43)
1	unsigned integer	Channel number
2	boolean (unsigned integer)	Enable positive pulse of channel
3	boolean (unsigned integer)	Enable negative pulse of channel

Table 3.17: Command

Example: `C,1,1,0\0` enables positive pulses on channel 1, and disables negative pulses.

Field	Format	Description
Header	character	C (0x43)
1	unsigned integer	Channel number
2	boolean (unsigned integer)	Enable positive pulse of channel
3	boolean (unsigned integer)	Enable negative pulse of channel

Table 3.18: Response

Example: `C,1,1,0\0` enables positive pulses on channel 1, and disables negative pulses.

3.2.10 Power Configuration

Command used to disable or enable the high voltage on the stimulator.

Field	Format	Description
Header	character	M (0x4D)
1	boolean (unsigned integer)	Power enabled
2	-	Reserved

Table 3.19: Command

Example: `M,1,1\0` enables the high voltage on the stimulator.

Field	Format	Description
Header	character	M (0x4D)
1	boolean (unsigned integer)	Power enabled
2	-	Reserved

Table 3.20: Response

Example: `M,1,1\0` enables the high voltage on the stimulator.

3.2.11 Stimulation Command

Command used to perform a stimulus. The command includes the number of patterns included in a stimulus, as well as a maximum response time for the subject. If no trigger input is given (Number of Triggers == 0), the stimulus is applied as soon as the command is received. When all stimuli have been given, another message is sent with Number of Triggers = 0, the same Number of Patterns per Trigger, and the response time of the subject. If the subject did not respond, command times out, and the response time is set to the maximum value.

Field	Format	Description
Header	character	S (0x53)
1	unsigned integer	Number of Triggers
2	unsigned integer	Number of Patterns per Trigger
3	unsigned integer [Timerunits]	Maximum response time

Table 3.21: Command

Example: `S,0,1,1000\0` performs a stimulus, not waiting for an external trigger. The stimulus consists of 1 pattern. The stimulator will respond `S,0,1,1000\0` to signal that the command has been received. The stimulator will wait 1000 Timerunits for a response from the subject.

Field	Format	Description
Header	character	S (0x53)
1	unsigned integer	Number of Triggers
2	unsigned integer	Number of Patterns per Trigger
3	unsigned integer [Timerunits]	Maximum response time

Table 3.22: Response

Example: `S,0,1,1000\0` performs a stimulus, not waiting for an external trigger. The stimulus consists of 1 pattern. The stimulator will respond `S,0,1,1000\0` to signal that the command has been received. The stimulator will wait 1000 Timerunits for a response from the subject.

Field	Format	Description
Header	character	S (0x53)
1	unsigned integer	Number of Triggers
2	unsigned integer	Number of Patterns per Trigger
3	unsigned integer [Timerunits]	Response time

Table 3.23: Secondary packet

Example: The given command is `S,0,1,1000\0`, indicating that the maximum response time is `1000 Timerunits`. The subject responds after `500 Timerunits`. The stimulator will send `S,0,1,500\0` to the controller.

3.2.12 Check Response Command

Command used to check the state of the stimulator. The sent command is always `R,0,0,0\0`. The stimulator responds using an indication to inform the master of the result. The first field contains the response signal, second contains whether the stimulus was triggered by an external signal. The IOK field is unused, and reserved for future use.

Field	Format	Description
Header	character	R (0x52)
1	boolean (unsigned integer)	0 (0x30)
2	boolean (unsigned integer)	0 (0x30)
3	boolean (unsigned integer)	0 (0x30)

Table 3.24: Command

Example: `R,0,0,0\0`

Field	Format	Description
Header	character	R (0x52)
1	boolean (unsigned integer)	Response Button is held
2	boolean (unsigned integer)	External Trigger is high
3	boolean (unsigned integer)	Battery voltage/Compliance voltage is ok (depending on device)

Table 3.25: Response

Example: `R,1,1,0\0` indicates that the button is currently held, and the stimulus was triggered externally.

3.2.13 Stimulus-response series [DEPRECATED]

Command used to ramp up the amplitude of 1 or all of the pulses in the pulse train. The configured pulse train is used to perform the ramp. The pulse train is repeated with increasing amplitude until the stop parameter is reached.

The ramp is started at the start parameter. All parameters are set in milliamperes. Why is this one not set in ADUnits, like the other ones, you ask? That's a great question.

The index parameter is used to configure which pulse in the train is modulated with the ramp. If the index is set to 0, all pulses in the train are modulated. Other values correspond with the index of the pulse in the train. e.g. 1 refers to the first pulse in the train, 2 to the second, etc.

If the crit parameter is set, the pattern is ignored, and the ramp stimulus is performed. The ramp is a sequence of block pulse stimuli, with each pulse one step stronger than the previous.

The ramp will continue until the stop parameter is reached, or until the button is let go.

Field	Format	Description
Header	character	Q (0x51)
1	unsigned integer	Ramp start current [mA]
2	unsigned integer	Ramp step current [mA]
3	unsigned integer	Ramp stop current [mA]
4	boolean (unsigned integer)	crit parameter
5	unsigned integer	Ramp pulse index

Table 3.26: Command

Example: `Q,2,1,7,0,0\0`

Field	Format	Description
Header	character	Q (0x51)
1	unsigned integer	Ramp start current [mA]
2	unsigned integer	Ramp step current [mA]
3	unsigned integer	Ramp stop current [mA]
4	boolean (unsigned integer)	crit parameter
5	unsigned integer	Ramp pulse index

Table 3.27: Response

Example: `Q,2,1,7,0,0\0`

3.3 Error codes

If a command is unknown, the processor will respond with the following package:

Table 3.28: Command

Field	Format	Description
Header	character	! (0x21)

Example:
`b,\0` is an unknown command, so the processor will respond with `!\0`

Appendix C

StimCom 3.0 Specification

StimCom 3.0 PROTOCOL SPECIFICATION

Version 1.0

Prepared by :
ing. Henry Slegers
ir. Frodo Muijzer

Submitted :
July 6, 2023

Contents

1 Version Control

Version	Date	Notes
0.1	2023/03/28	Initial Version
0.2	2023/04/04	Added Command descriptions
0.3	2023/06/08	Completed Message Charts and Protocol Description
1.0	2023/07/06	No Changes

2 Introduction

2.1 Purpose

The StimCom Protocol Specification defines the wireless Bluetooth Low Energy interface between one or more NociTrack devices, such as the AmbuStim, and an external device. It is a reference for developers, project managers, users and testers to evaluate and analyse the protocol in development and verfication.

2.2 Intended Audience

This Protocol Specification is for developers, project managers, users and testers.

3 Protocol

3.1 Introduction

The Bluetooth Low Energy, Core specification 5.3 (2021-07-13), radio and protocol is used as the basis for the StimCom 3.0 protocol. The protocol is backwards compatible with previous version of Bluetooth Low Energy, down to Bluetooth Low Energy, Core specification 4.0 (2010-6-30).

The StimCom protocol provides the StimCom Bluetooth Low Energy Service to connected devices, through which all wireless interactions with an active NociTrack device can be done. It also provides the procedure through which an external device can connect to the device, as well as procedures through which active NociTrack devices communicate and synchronise with each other.

This version of the StimCom protocol must at all times be backwards compatible with the previous version of StimCom 2.1, or StimCom Serial. The protocol can expand the functionality of the serial protocol, but may not discard or alter deprecated functionality.

3.2 Description

The StimCom 3.0 is intended as a way of direct communication between a controller and a stimulator using Bluetooth Low Energy. The protocol is meant to be expandable to mesh networks of multiple stimulators and a single controller, and perform accurate time-synchronized pulses.

Static parameters, such as the device software version, serial number, calibration parameters, etc. are only accessible through a read operation.

To perform a StimCom operation, the controller performs an acknowledged write operation on the GATT Characteristic associated with the command. The stimulator receives the command from a controller, upon which it will first verify the validity of the parameters. If the parameters are valid, the command will be executed as expected. If they are deemed invalid, the stimulator will attempt to alter the command, and the execute the altered command. The stimulator will then respond with the altered command. This response comes in the form of an acknowledged indication operation. This indication will always follow within at most 2 connection intervals. For example, if the stimulator has a maximum positive current configuration of 1000 ADunits, and the value `1100,900` is written to the Positive Pulse Amplitude Configuration, as described in 3.3.5, the response indication will be `1000,900`. Because both the write and indicate operation are acknowledged operations, this creates a robust synchronisation between connected devices.

Should the indication not be received within the allotted time, the operation is considered a failure, and the device is considered desynchronized. The operation should be retried until it is successfully completed, or until definitive failure is reached.

Some commands, currently only the Stimulation Command, take more time to complete. These commands send a second indication after they have completed.

3.3 StimCom Bluetooth Low Energy Service

The StimCom Service provides characteristics which are used to configure device and pulse parameters on the NociTrack device. Each characteristic is an ASCII-encoded character array containing one or more data fields separated by comma characters (`","`), and provides the necessary operations (read/write/notify/indicate) to fulfill its specific purpose.

3.3.1 Version Query

Contains the version and serial number of the embedded software on the device.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	Y	N	N	N	N

Field	Format	Description
1	unsigned integer	Major Version
2	unsigned integer	Minor Version
3	unsigned integer	Serial Number

Example: `1,0,27` denotes that the device with serial number 27 has an embedded software version of 1.0.

3.3.2 Feature Query

Contains the hardware features of the device.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	Y	N	N	N	N

Field	Format	Description
1	unsigned integer	Number of Channels
2	unsigned integer	Maximum Pattern Length
3	unsigned integer	DAC Calibration value [ADunits-to-mA]
4	unsigned integer	Timer Calibration value [Timerunits-to-ms]

Example: `1,20,80,35` denotes that the device has 1 stimulation channel, can store stimulation patterns up to 20 stimuli, has a DAC Calibration value of `80 ADunits/mA`, and has a timer calibration value of `35 Timerunits/ms`

3.3.3 Interval Configuration

Contains the pulse intervals between pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
n	unsigned integer	Interval Length [Timerunits]

Example: `40,40,10,10` refers to a pulse train consisting of 4 pulses, after the first there is a delay of 40 Timerunits, after the second, 40 Timerunits, after the third, 10 Timerunits, and after the fourth, 10 Timerunits.

3.3.4 Pulse Channel Configuration

Contains the channels on which the pulses of the pulse train will occur, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
n	unsigned integer	Pulse Channel

Example: `1,1,2` refers to a pulse train consisting of 3 pulses, the first and second of which will occur on channel 1 of the stimulator. The third occurs on channel 2.

3.3.5 Positive Pulse Amplitude Configuration

Contains the pulse amplitudes of the positive pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
n	unsigned integer	Pulse Amplitude [ADunits]

Example: `40,30,20,10` refers to a pulse train consisting of 4 pulses, the first of which has an amplitude of 40 ADunits, the second of 30 ADunits, third of 20 ADunits, and the fourth of 10 ADunits.

3.3.6 Negative Pulse Amplitude Configuration

Contains the pulse amplitudes of the negative pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
n	unsigned integer	Pulse Amplitude [ADunits]

Example: `40,30,20,10` refers to a pulse train consisting of 4 pulses, the first of which has an amplitude of -40 ADunits, the second of -30 ADunits, third of -20 ADunits, and the fourth of -10 ADunits.

3.3.7 Positive Pulse Width Configuration

Contains the pulse widths of the positive pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
n	unsigned integer	Pulse Width [Timerunits]

Example: `10,20` refers to a pulse train consisting of 2 pulses, the first of which has an positive width of 10 Timerunits, the second of 20 Timerunits.

3.3.8 Negative Pulse Width Configuration

Contains the pulse widths of the pegative pulses of the pulse train, with a number of fields equal to the number of pulses. Pulses are sorted in the order they appear in the pulse train. May contain up to 20 pulses.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
n	unsigned integer	Pulse Width [Timerunits]

Example: `10,20` refers to a pulse train consisting of 2 pulses, the first of which has an negative width of 10 Timerunits, the second of 20 Timerunits.

3.3.9 Channel Enable Configuration

Command used to disable or enable a channel on the stimulator. Positive and negative pulses can be disabled independently.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
1	unsigned integer	Channel number
2	boolean (unsigned integer)	Enable positive pulse of channel
3	boolean (unsigned integer)	Enable negative pulse of channel

Example: `1,1,0` tells the stimulator to enable the positive pulse, and to disable the negative pulse of channel 1.

3.3.10 Power Configuration

Command used to enable or disable the power output on the stimulator.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
1	boolean (unsigned integer)	Power enabled
2	-	Reserved

Example: `0` tells the stimulator to disable the power output.

3.3.11 Stimulation Command

Command used to perform a stimulus. The command includes the number of patterns included in a stimulus, as well as a maximum response time for the subject. If no trigger input is given (Number of Triggers == 0), the stimulus is applied as soon as the command is received. After the command is given, an indication is returned immediately. When all stimuli have been given, the master device is sent another indication with Number of Triggers = 0, the same Number of Patterns per Trigger, and the response time of the subject. If the subject did not respond, command times out, and the response time is set to the maximum value.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
1	unsigned integer	Number of Triggers
2	unsigned integer	Number of Patterns per Trigger
3	unsigned integer [Timerunits]	[Write, First Indication] Maximum response time [Second Indication] True response time

Example 1: `0,1,1000` performs a stimulus, not waiting for an external trigger. The stimulus consists of 1 pattern. The stimulator will indicate `0,1,1000` to signal that the command has been received. The stimulator will wait 1000 Timerunits for a response from the subject. The subject responds 500 Timerunits. The stimulator will indicate `0,1,500` to the master.

Example 2: `2,4,1000` waits for an external trigger before stimulation begins. The stimulator will indicate `2,4,1000` to signal that the command has been received. When a trigger is received, the stimulator immediately applies the stimulus. The stimulus consists of 4 patterns. When 2 trigger pulses have been received and executed the command has been completed, the stimulator will indicate `0,1,1000` to signal that all stimuli have been performed.

3.3.12 Check Response Command

Command used to check the state of the stimulator. The sent command is always `0,0,0`. The stimulator responds using an indication to inform the master of the result. The first field contains the response signal, second contains whether the stimulus was triggered by an external signal. The IOK field is unused, and reserved for future use.

Operation	Read	Write	Write no Ack	Notify	Indicate
Supported	N	Y	N	N	Y

Field	Format	Description
1	boolean (unsigned integer)	Response Button is held
2	boolean (unsigned integer)	External Trigger is high
3	boolean (unsigned integer)	Battery voltage/Compliance voltage is ok (depending on device)

Example: After `0,0,0` has been sent, the stimulator responds with `1,1,0`. This indicates that the button is currently held, and the stimulus was triggered externally.

3.3.13 Error code

If any command or parameter can not be handled by the stimulator, the stimulator will not execute the command, the response indication on the characteristic will always be

Field	Format	Description
1	Character	!

No further information is given, and the master device is left to decide the best course of action.

3.4 Security and Encryption

In order to use a stimulator using StimCom 3, the connected device must be paired to it. Paired devices must used some way of encryption to prevent other malicious devices from listening and/or injecting messages.

3.5 Connection procedure

Each stimulator must have a qr code sticker on it that contains all information necessary for a master device to connect to the stimulator (e.g. device address, passcodes, encryption keys, etc.). In this way, in a room with many other potential valid devices, scanning the qr code will ensure that the master connects securely with the desired device.

4 Example message sequence chart

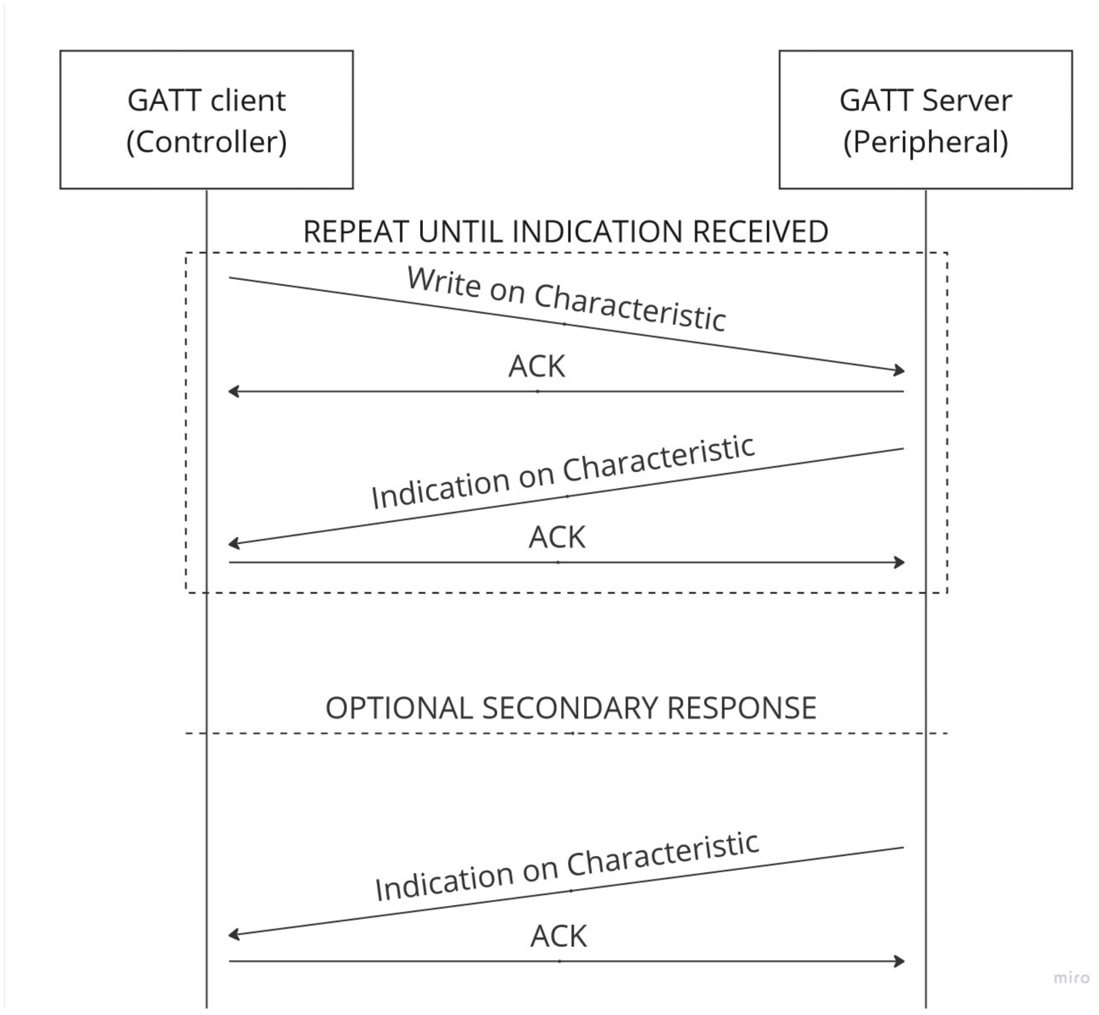

GATT client
(Controller)
GATT Server
(Peripheral)
REPEAT UNTIL INDICATION RECEIVED
Write on Characteristic
ACK
Indication on Characteristic
ACK
OPTIONAL SECONDARY RESPONSE
Indication on Characteristic
ACK
miro

Figure 4.1: Message chart for StimCom operations

5 StimCom Bluetooth Low Energy Service UUID table

The StimCom Service uses a Service Address UUID (128 bits) of
`e9ef0001-9644-424f-a318-bf065e5efc6`

UUID	Description
`e9ef0001-9644-424f-a318-bf065e5efc6`	StimCom Service Base Address
`e9ef0002-9644-424f-a318-bf065e5efc6`	Version Query
`e9ef0003-9644-424f-a318-bf065e5efc6`	Feature Query
`e9ef0004-9644-424f-a318-bf065e5efc6`	Interval Configuration
`e9ef0005-9644-424f-a318-bf065e5efc6`	Pulse Channel Configuration
`e9ef0006-9644-424f-a318-bf065e5efc6`	Positive Pulse Amplitude Configuration
`e9ef0007-9644-424f-a318-bf065e5efc6`	Negative Pulse Amplitude Configuration
`e9ef0008-9644-424f-a318-bf065e5efc6`	Positive Pulse Width Configuration
`e9ef0009-9644-424f-a318-bf065e5efc6`	Negative Pulse Width Configuration
`e9ef000A-9644-424f-a318-bf065e5efc6`	Enabled Channel Configuration
`e9ef000B-9644-424f-a318-bf065e5efc6`	Power Configuration
`e9ef000C-9644-424f-a318-bf065e5efc6`	Stimulation Command
`e9ef000D-9644-424f-a318-bf065e5efc6`	Check Response Command

Table 5.1: StimCom Service UUID table

The addresses, characteristic names and permissions for the StimCom 3 Bluetooth Low Energy Service are recorded in the associated XML file `stimcom_bleservice.xml`