FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Wireless CAN bus bridge

Pedro Miguel Marinho Almeida

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Miguel Pinho de Almeida (FEUP)

Co-Supervisor: José Ricardo de Sousa Soares (AddVolt)

June 24, 2019

Abstract

As a society, one of the main challenges for the upcoming century is the demand to reduce energy consumption and global warming. In order to do so, there is a urge to embrace the use of technologies that are environment friendly. The need by food industries to transport their refrigerated goods around the world and ensure that these remain suitable for consumption coupled with the fact that they are usually powered by a complementary diesel engine, led AddVolt to create and design from scratch a plug-in electric system that replaces the current dedicated diesel engine.

This product and similar ones are usually installed on trucks and trailers in which the drivers cabine is detachable from the refrigeration unit. In order to get rid of situations where cabling is not desirable, the use of an wireless gateway is a crucial feature. Transmission of information between the Electronic Control Units of these vehicles is usually done via a CAN bus. This wireless gateway formed by two transceivers must be able to forward all the information that arrives on each transceiver respective CAN network to the common wireless link.

The solution this dissertation proposes is dependent on a complete study of various wireless technologies and its main features. Key features like range and speed are crucial for the well functioning of said wireless gateway. After the gathering of the main requirements of the system and a full knowledge of the available hardware, a coherent choice must be made regarding the wireless technology, as well as the development of the firmware in C. It is crucial that the architecture and the system design as a whole are coherent with the application demands that will directly make use of the wireless gateway due to the fact that we are dealing with asynchronous communication.

It is also this dissertation goal to fully describe the implementation of this wireless CAN bridge to be fully integrated with AddVolt's network. We also aim to define metrics that will allow to measure and define the quality and robustness of the finished prototype. The challenges of the process, along with the testing methodologies and results will all be presented and discussed.

Acknowledgements

First and foremost I would like to thank the two superheroes that stood by me in this long journey that was my Master's degree. My mom Maria Amélia and my dad José Maria. Without their support, their love, their example and their resilience this degree would just not be possible. I would also like to thank myself for my bravery and for ultimately believing in me when things got more complicated. To my grandmother Ana for her unconditional love and wisdom. To the rest of my family for the good moments when we all got together.

I am thankful to have had the opportunity to be supervised and advised throughout the whole dissertation by Professor Luís Almeida. There is nothing as too much sympathy. In every meeting I learned something new.

I am also very blessed to have worked in such a wonderful and professional environment at AddVolt's office. I don't remember a single day where everybody was not in a good mood and ready to help. I thank Ricardo Soares for his supervision and Ricardo Gonçalves for all the time invested in me.

To all my friends, especially Michael, for giving me headaches and laughs at the same time and for making me feel appreciated even when I wasn't physically present.

Last but not least to my girlfriend. Thank you Lucia for you love, for making me a better person and for believing in me. You made a lot of days better just by being present.

Pedro

If everything I did failed,
just the fact that I'm willing to fail is an inspiration.
People are so scared to lose that they don't even try.

Kanye

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ATT	Attribute Protocol
BLE	Bluetooth Low Energy
CAN	Controller Area Network
DCE	Data Communication Equipment
DTE	Data Terminal Equipment
ECU	Electronic Control Unit
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
HGV	Heavy Goods Vehicle
IoT	Internet of Things
ISO	International Organization for Standardization
Kbit/s	Kilobit per second
LLC	Logical Link Control
MAC	Medium Access Control
Mbit/s	Megabit per second
MITM	Man In The Middle
MCU	Microcontroller Unit
MLU	Management Logging Unit
NRZ	Non-Return-To-Zero
OOB	Out Of Band
OSI	Open Systems Interconnection
PCB	Printed Circuit Board
PDU	Protocol Data Unit
SMP	Security Manager Protocol
SPS	Serial Port Service
STK	Short Term Key
TRU	Transport Refrigeration Unit
UART	Universal Asynchronous Receiver-Transmitter
URL	Uniform Resource Locator

Chapter 1

Introduction

1.1 Context

The never-ending increase in road usage by automotives has raised concern about air pollution and greenhouse gases, because road freight transport is regarded as being one of the major contributors to air pollution. As a result, in order to try to lower carbon dioxide (CO_2) emissions, international organizations, as well as national and local governments, have been trying to increase the proportion of freight transported by non-road freight modes, concentrating on inter-modal freight systems to provide for door-to-door demands[1]. Nonetheless, the freight transport industry focuses mainly on profitability, caring little of environmental concerns about truck-only freight transportation and consequently no substantial improvements have been made in this regard. As most governments around the world fall short to achieve meaningful developments in this issue, it lands on the responsibility of forward thinking companies to redesign the current landscape of the heavy truck transportation industry and therefore contribute to a cleaner and less polluted environment.[2]

AddVolt is a technology-based company that fits in this context. Traditionally, Heavy Goods Vehicles (HGVs) that deal with the transportation of temperature sensitive goods are equipped with a Transport Refrigeration Unit (TRU) which is powered by a dedicated diesel engine. AddVolt's innovative product, a plug-in electric system for refrigeration units of vans, trucks and trailers, accomplishes the same goal as the standard and massively used diesel engine but in a much more efficient, quieter and cleaner way. After two years of tests and trials on the road, made possible thanks to a partnership between Addvolt and the truck transportation company Luis Simões, it has been proved that this product allows a reduction in noise, fuel consumption, maintenance costs and less 87% of CO_2 emissions, with respect to the total carbon dioxide emissions produced by the refrigeration unit involved in the transportation of goods subjected to temperature control[3].

1.2 Motivation

The product from Addvolt is capable of producing, recovering and storing electrical energy. In particular, it recovers energy during the braking of the truck without the need of any structural modifications to the vehicle. The equipment is usually attached to the bottom of the truck and is connected to the Controller Area Network (CAN) bus of the vehicle.

Currently, modern trucks can have up to 100 Electronic Control Units (ECUs) for various subsystems such as engine, transmissions, body computers, doors or the instrument panel. The CAN bus is used in vehicles to allow communication between these ECUs and sensors/actuators (also known as nodes), with speeds up to 1 Mbit/s. Therefore, it somewhat acts as a central networking system of the vehicle and it is of extreme importance because a subsystem may need to control actuators or receive feedback from sensors. On top of that it enables this communication to happen in a efficient, robust and flexible way while also being low cost and allowing for central error diagnosis and configuration across all ECUs.

Transmissions on a bus, like the CAN bus, take place over wires laid out between nodes, leading to an increase in size and complexity of the cable harnessing in the vehicle which also adds supplementary weight. In some situations using a bus is highly undesirable because the driver's cabin and the CAN network of interest may be detachable (e.g. in a trailer). In this scenario in particular, where reliability and safety are less critical, having a wireless gateway to interconnect CAN networks would be extremely useful providing an easy installation and a flexible extension of the bus, even knowing that wireless communication technologies are less reliable than CAN-based communication. This type of technology needs more than just a cable replacement because the system design will convert from a single CAN network into a system with two CAN networks connected via a wireless gateway. In this situation, the wireless gateway acts as a point-to-point CAN bridge. A CAN message is transmitted through the bus and is received by the first wireless transceiver. This message is then sent through a wireless technology, interpreted by the second wireless transceiver and sent out to the second CAN bus. Other interesting case of use is when the vehicle is rigid and the driver intends to read diagnostic data from the truck's CAN bus, being that the most common solution nowadays still requires the device that reads such data to be connected via wire. Conversely, wireless technology provides freedom of movement, eliminates expensive and maintenance-heavy cabling and connectors and offers an easy integration of devices or subsystems in the network such as an operator or maintenance console.

1.3 Goals

The objective of this dissertation is to provide an extension to AddVolt's already implemented CAN bus, using a wireless technology, to connect to other systems such as the CAN bus of the vehicle and an operator console. Consequently, this work will simplify the intra-vehicle wiring to connect the entire AddVolt system but we need to select, adapt and put in place a wireless technology that can connect two or more physically separated CAN networks, while preserving

and meeting the requirements of the protocol. Although any wireless protocol can be used to send data, each one has its pros and cons. As a consequence, a study and research of various wireless protocols such as WiFi, Bluetooth and ZigBee is required. Performance metrics such as range, signal quality, data throughput and latency will be taken into consideration for the final product.

A certain level of data security is expected so the actual communication does not expose sensitive data nor allow malicious interference that could degrade throughput and latency. The work will start with detailing the system requirements leading then to the development of the firmware and perhaps some hardware that augments the existing AddVolt's local CAN network to be compatible with said wireless CAN bridge.

1.4 Organization of this document

In this dissertation we focus on the design of the referred wireless gateway to interconnect two or more CAN segments.

- In chapter 1 we introduced the project along with the context and its goals.
- In chapter 2 we present some properties of communication technologies for distributed embedded systems that are relevant for our problem, focusing on CAN, WiFi, Bluetooth and ZigBee.
- In chapter 3 we aim for a detailed study of the Bluetooth Low Energy technology with focus on the improvements regarding the last specification (Bluetooth 5).
- In chapter 4 we specify the problem and provide a possible solution. We also specify the architecture of the system, detailing the hardware and software structures that will help design a full functional prototype of the gateway.
- In chapter 5 we analyze and explain the main parts of the implementation.
- In chapter 6 we present the validation tests that were made and the results of those tests. An explanation for these same results is given within the context of the environment that the tests were made.
- In chapter 7 the final conclusions of the project are presented and aspects that could be further improved in the future.

Chapter 2

Literature Review and State of the Art

This chapter comprises a literature review on the subjects involved in this dissertation, as well as a general view of the current technologies involved. Fundamental concepts will be presented here, more specifically on the wired communication protocol CAN bus and three wireless technologies, Wi-Fi, Bluetooth and Zigbee. Existent solutions in the market that are used to augment a CAN bus wirelessly will also be exploited.

2.1 Communication Networks

The vast majority of communication networks are designed following, at least in some way, the Open Systems Interconnection Model (ISO 7498). This conceptual model was introduced in 1978 and characterizes and standardizes the communication functions of a computing system without taking into consideration its underlying internal structure and technology. The model partitions a communication system into seven different abstraction layers. The following diagram illustrates the OSI reference model and its vertical layer stack.[4]

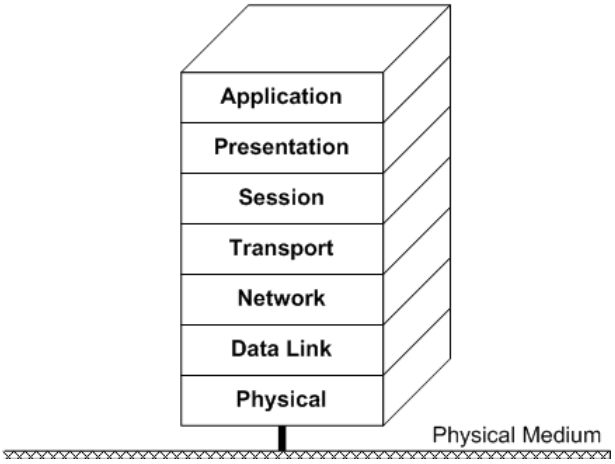

Figure 2.1: The OSI Reference Model.[4]

- The Application layer is the OSI's closest layer to the end user. It provides a set of services directly to the user, following some adequate interaction paradigm.
- The Presentation layer transforms data into the form that the application accepts and it formats data to be sent across a network.
- The Session layer controls the dialogues (connections) between nodes. It establishes, manages and terminates the connections between communication end points.
- The Transport layer deals with the coordination of the data transfer between end systems and hosts. Problems like how much data to send, at what rate or its destination are all dealt with within this particular layer.
- The Network layer is where nodes are connected (via logical paths, known as virtual circuits) and it provides a way of transferring data between them through packets (variable length data sequences). Forwarding and routing take place in this layer.
- The Data Link layer provides immediate node-to-node data transfer and also handles error correction from the physical layer. It comprises two sub-layers - the Medium Access Control (MAC) layer and the Logical Link Control (LLC) layer.
- The Physical layer represents the electrical and physical communication medium. This can include everything from the cable type to radio frequency links and aspects like voltages.

A layer serves the layer above it and is served by the layer below it. For example, a layer that provides error-free communication across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that comprise the contents of that path. Two instances at the same layer are visualized as linked by a horizontal connection in that tier.

In a machine-to-machine communication system it is necessary to understand its limitations and implement a number of precautions with the guarantee that the system will be able to handle with undesired behaviour of the network or the hardware itself. Nodes must be able to identify each other and also to be able to synchronize with each other in order to have a more reliable interaction. Other important property is flow control, meaning that it must prevent situations where the sender overflows the recipient with data. The communication network should also be able to deal with errors, using error check algorithms to bypass those situations. All these aspects make the OSI model (and the protocols in it) very useful when designing or implementing a communication system.[5]

2.2 Wired Communication

The recent history of the vehicle industry is characterized by intensive electronics. The driving force for this is the increased growth of customer wishes for modern vehicles. In addition, increas-

ingly stricter regulations are being legislated for exhaust emissions. Another factor is globalization, which increases competitive and cost pressures that result in continual innovative pressure.

At first, independently operating electronic units were sufficient to implement electronic functions. However, it was soon recognized that the coordination of Electronic Control Units (ECUs) could greatly enhance automobile functionality. Data exchange between ECUs was initially implemented conventionally, i.e. a physical communication channel was allocated to every signal to be transmitted.

However, in spite of the intensive wiring effort, just limited data exchange. One of the solutions that seemed to offer a way out of this dilemma came from serial bit exchange of data via a single communication channel (bus). This led to the concept of a serial communication system tailored to the requirements of the vehicle.[5]

2.2.1 Universal Asynchronous Receiver-Transmitter

A Universal Asynchronous Receiver-Transmitter (UART) is a computer hardware device that performs serial-to-parallel conversion of data, received from peripheral devices, and parallel-to-serial conversion of data, coming from the CPU, for transmission to peripheral devices. The electric signaling levels and methods are handled by a driver circuit external to the UART.

Communication may be simplex (in one direction only, with no provision for the receiving device to send information back to the transmitting device), half duplex (devices take turns transmitting and receiving) or full duplex (both devices send and receive at the same time). For full-duplex communication, an independent communication line is needed for each transfer direction.

Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires. Because of that one or more UART peripherals are commonly integrated in microcontroller chips.

2.2.1.1 Frame Format

A standard UART frame consists of:

- An idle time with the signal level 1.

- One start of frame bit (SOF) with the signal level 0.

- A data field containing a programmable number of data bits (1-63).

- A parity bit (P), programmable for either even or odd parity. It is optionally possible to handle frames without a parity bit.

- One or two stop bits with the signal level 1.

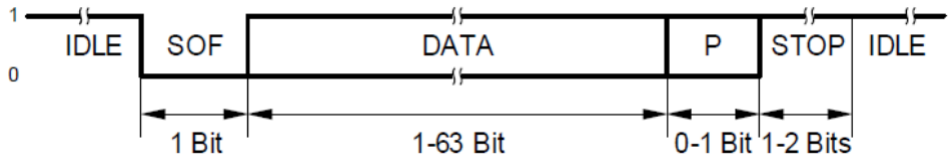

Figure 2.2: Standard UART Frame Format.[6]

8-N-1 is the most common configuration for serial communications today. It's a common shorthand notation for a serial port parameter setting or configuration in asynchronous mode, in which there is one start bit, eight (8) data bits, no (N) parity bit, and one (1) stop bit.

The abbreviation is usually given together with the line speed in bits per second, as in 9600/8-N-1. The speed includes bits for framing (stop bits, parity, etc.) and the effective data rate is lower than the bit transmission rate. For 8-N-1 encoding, only 80% of the bits are available for data (for every eight bits of data, ten bits are sent over the serial link — one start bit, the eight data bits, and the one stop bit).

2.2.2 Controller Area Network

In the 1980s a serial communication system appeared being developed by Bosch, given by the name of CAN (Controller Area Network). To this day, CAN is still in use in motor vehicles in networking ECUs in the power-train, chassis and convenience areas. Above all, CAN is characterized by very reliable data transmission that satisfies the real-time requirements of such target usage areas. It is essentially a broadcast network, where a message transmitted will be received by all nodes that are attached to the bus. There is no way to send a message to just a specific node.

With CAN, complex wire harnesses in the automobile have become a thing of the past. The specification calls for signaling rates up to 1 Mbit/s, high immunity to electrical interference, and an ability to self diagnose and repair data errors. These features have led to its popularity in a variety of industries. It can be found in a multiplicity of vehicles such as trucks, passenger cars, boats, spacecrafts and others. For example, the CAN network has been mandatory since 2001 for European vehicles and since 2008 in the United States.[7]

The main properties of the CAN bus are:

-Multimaster;

-Error detection;

-Configuration flexibility;

-Prioritization of messages;

-Guarantee of latency times;

-System wide data consistency;

-Retransmission of corrupted messages;

-Multicast transmission with synchronization;

-Distinction between temporary errors and permanent failures of nodes.

2.2.2.1 Standardization

CAN technology has been standardized since 1993. ISO has defined a standard which incorporates the original CAN specification named ISO 11898-1.

AddVolt's transceivers are compliant with the ISO 11898 standard, therefore it is important to further explore it. ISO 11898 describes the general architecture of CAN in terms of hierarchical layers according to the OSI model described before in this chapter, in order to achieve transparency and implementation flexibility.

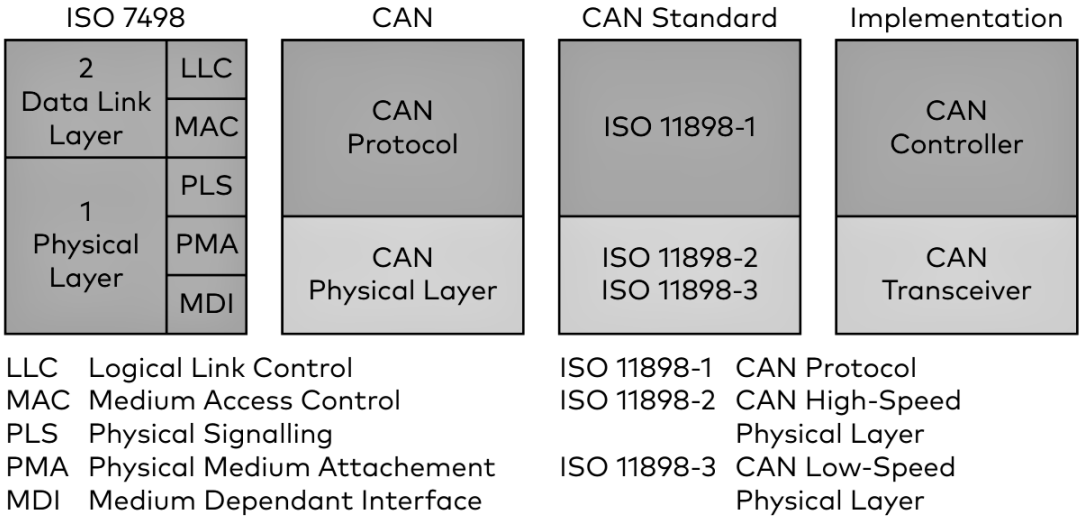

Figure 2.3: CAN Standard regarding the OSI model.[8]

The ISO 11898 standard series is a multi-part standard including the following parts:
Part 1: defines the data link layer including the logical link control (LLC) sub-layer and the medium access control (MAC) sub-layer, as well as the physical signalling (PS) sub-layer;
Part 2: defines the high-speed physical medium attachment (PMA);
Part 3: defines the low-speed fault-tolerant physical medium attachment (PMA);
Part 4: defines the time-triggered communication;
Part 5: defines the power modes of the high-speed physical medium attachment (PMA);
Part 6: defines the selective wake-up functionality of the high-speed physical medium attachment (PMA).

2.2.2.2 Architecture

Both ISO 11898-2 and ISO 11898-3 cover the PMA sub-layer and describe two different CAN physical layers: the high-speed CAN physical layer and the low-speed CAN physical layer. They differ primarily in their definition of voltages and data transmission rates. ISO 11898-3 allows data rates up to 125 kbit/s. It is primarily used in the convenience area of the automobile. ISO 11898-2 allows data rates up to 1 Mbit/s. Consequently, ISO 11898-2 is primarily used in the power-train and chassis areas of the automobile and it is the one relevant for the scope of this dissertation.

No standard exists for the MDI (Medium Dependent Interface) sub-layer of the physical layer although it is recommended the use of SUB-D9 connectors.

High-speed CAN configuration is represented by a serial bus that allows the interconnection of various ECUs present in a vehicle, being that they can be more or less sophisticated, from simple input/output devices to computers with more complex software. In this configuration all the nodes are connected to a twisted pair of wires. To suppress signal reflections, the bus must be terminated at both ends with impedance matching resistors of 120 Ohm.

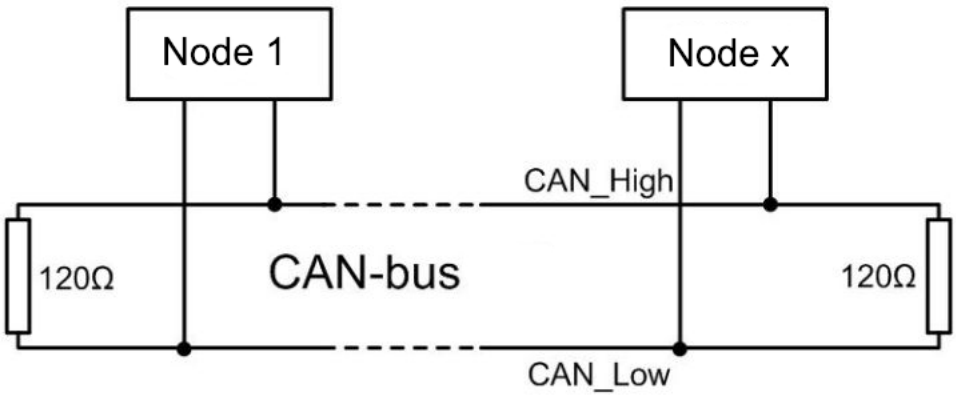

Figure 2.4: Representation of a high speed CAN bus as described by ISO-11898-2.[9]

Traditional CAN communication is event based: transmissions are triggered asynchronously by the nodes. With a higher bus load this may lead to delays and particularly to high network delay jitter. Note that CAN allows computing deterministic upper bounds to network delays, based on fixed priorities assigned statically to messages. To assure deterministic communication in a CAN network with low delay jitter, ISO 11898-4 is available. ISO 11898-4 is an extension of the Data Link Layer that adds a time triggered communication option for CAN-based networks that allows avoiding asynchronous interference.

An ECU that performs its tasks in a CAN network is referred to as a CAN node. Each node in the CAN bus requires the following:

- **CAN Transceiver** – It connects the CAN controller to the physical transmission medium (responsible for data conversion from the controller to bus levels and vice versa).

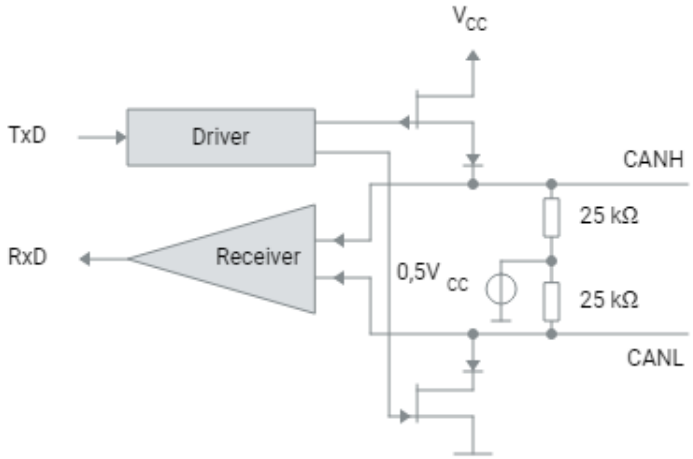

Figure 2.5: A typical CAN transceiver module.[10]

- **CAN Controller** – Receives the message to be transmitted and stores the message in a buffer before further message processing takes place.
- **Microcontroller** – Runs the node software and it may be connected to a sensor/actuator.

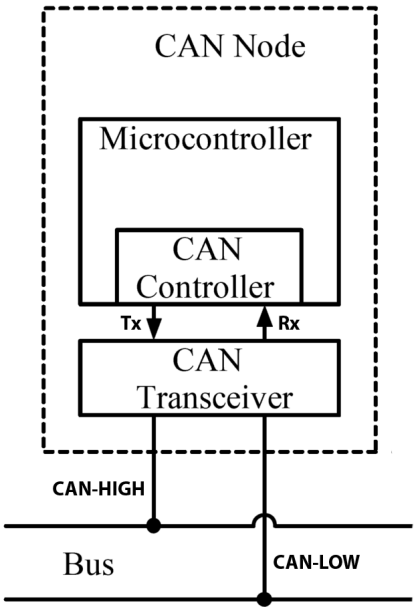

Figure 2.6: Representation of a common CAN node.[9]

2.2.2.3 Communication Principle

High usage applications, such as those in the automotive area, place severe demands on a communication system's availability. On the other hand, safety-critical applications in this domain stress the demand for high reliability. Thus, assigning the responsibility for data distribution to just a

single bus node seems inadequate since it would generate a single point of failure. The failure of a single node could cause all communications to fail. A much more efficient solution is to decentralize bus access, so that each bus node has the right to communicate independently of the remaining ones.

That is why a CAN network is referred as following a multi-master architecture: essentially each CAN node is authorized to place CAN messages on the bus in a CAN network. The transmission of CAN messages, except for 11898-4, does not follow any predetermined time sequence, it is event-driven, as said before.

As specified in ISO-11898-2 the bit stream in a message is coded according to the Non-Return-To-Zero (NRZ), meaning that the total bit time generates either 'dominant' (0) or 'recessive' (1) bit levels. The dominant level prevails over the recessive one if the two logical states collide. This property, knows as dominance, is what allows a deterministic computation of network delay by enforcing a fixed priority medium access protocol.[11]

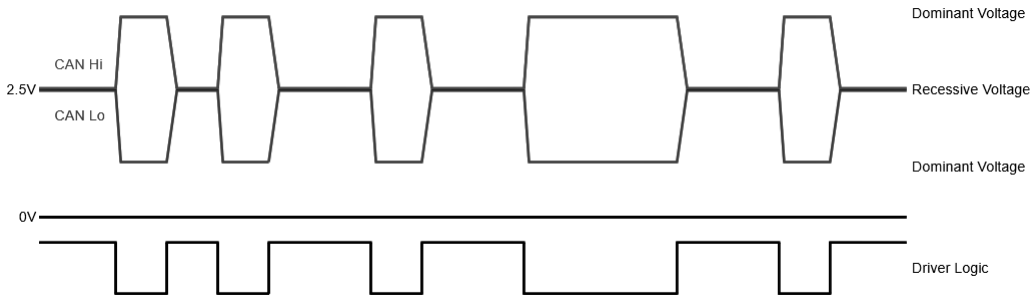

Figure 2.7: A CAN bus signal example.[12]

Therefore, every bit transmitted on the bus is defined as recessive or dominant, which maps to 1 or 0. All nodes can listen and transmit at the same time. If more than one node is transmitting, the result will carry a dominant bit if at least one node is transmitting a dominant bit. When a node transmits a dominant bit, it will see a dominant bit on the bus. In this case, the node will not know if anyone else was trying to transmit and thus, continues transmitting. If a node transmits a recessive bit, but a dominant bit is seen on the bus, the node knows that another node in the bus is transmitting a message with higher priority. Thus, it backs off and retries after the ongoing transmission. This is the CAN arbitration and the most interesting effect is the node decision to back off if some other node transmits a dominant bit the first time the first node sends a recessive bit. At data rates up to 1 Mbit/s, real-time data transmissions with network delays of the magnitude of a few milliseconds are possible in a CAN network.

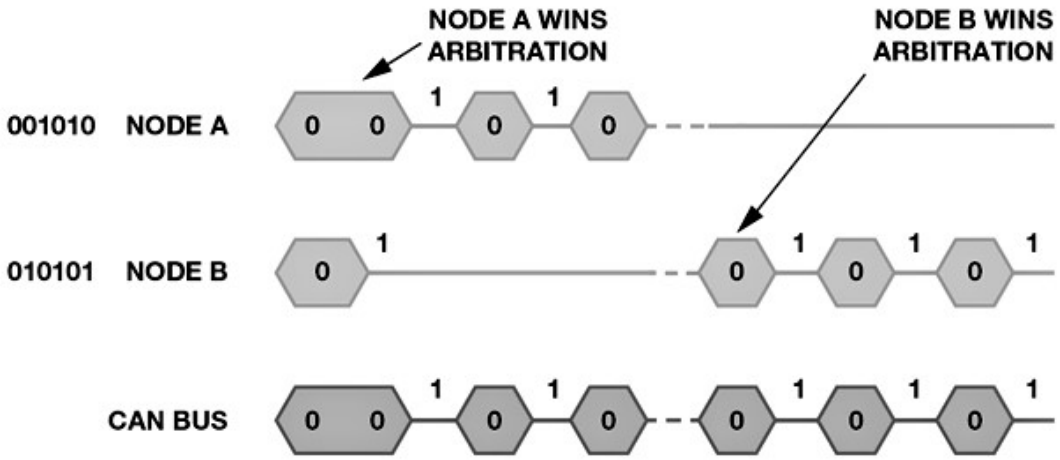

Figure 2.8: Arbitration logic between two CAN nodes.[13]

For all of the above to work it must be possible to mark each CAN message with a message identifier (ID). Although this increases overhead, it allows a deterministic arbitration that is responsible for the desirable real-time properties of CAN.

ISO 11898-1 defines a multi-master architecture to assure high availability and event-driven data transmission. Each node in the CAN network has the right to access the CAN bus without requiring permission and without prior coordination with other CAN nodes. Although bus access based on an event-driven approach enables very quick reactions to events, there is the inherent risk that several CAN nodes might want to access the CAN bus at the same time, which would lead to undesirable overlaps of data on the CAN bus.

To preserve the communication system's real-time capability, ISO 11898-1 provides for a bus access that guarantees nondestructive data transport. The so-called CSMA/BA (Carrier Sense Multiple Access with Bit-wise Arbitration) method is used. The CSMA/BA method ensures that CAN nodes wishing to send do not access the CAN bus until it is available. In case of simultaneous bus access, the CSMA/BA method based on the bit-wise bus arbitration principle explained above, ensures that the highest priority CAN message among the CAN nodes prevails. In principle, the higher the priority of a CAN message the sooner it can be transmitted on the CAN bus. In case of poor system design, low priority CAN messages even run the risk of never being transmitted.

2.2.2.4 Synchronization

Arbitration is only effective if both nodes are able to sample the bus during the same bit time so that all nodes "see" the same bit. Each bit on the CAN bus is, for timing purposes, divided into at least 4 quanta. The quanta are logically divided into four groups or segments.

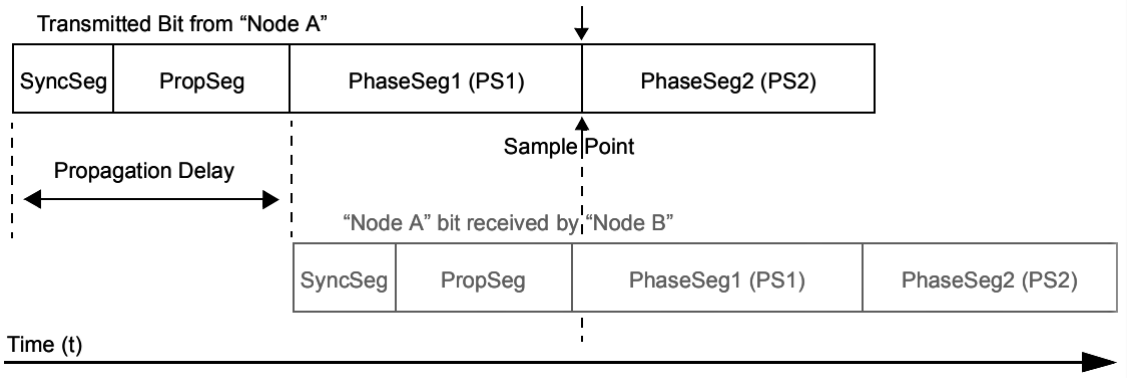

Figure 2.9: CAN bit timing.[12]

The Synchronization Segment, which always is one quantum long, is used for synchronization of the clocks. A bit edge is expected to take place here when the data changes on the bus.

The Propagation Segment is needed to compensate for the delay in the bus lines.

The Phase Segments may be shortened (Phase Segment 1) or lengthened (Phase Segment 2) if necessary to keep the clocks in sync.

The bus levels are sampled at the border between Phase Segment 1 and Phase Segment 2.

Hard synchronization occurs on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.

Re-synchronization occurs when a bit edge does not occur within the Synchronization Segment in a message. One of the Phase Segments is shortened or lengthened with an interval that depends on the phase error in the signal.

2.2.2.5 Frame Types

There are four different frame types that can be transmitted on a CAN bus: the data frame, the remote frame, the error frame, and the overload frame.

1) The Data Frame: This is the most common message type, and it is made of the arbitration field, the data field, the CRC field, and the acknowledgement field. Transmission of a data frame begins with the start bit SOF (Start of Frame). It is transmitted by the sender as a dominant level which produces a signal edge from the previous recessive (bus idle) level which is used to synchronize the entire network. The arbitration field determines the priority of a message when two or more nodes are contending for the bus. The arbitration field contains an 11-bit identifier for CAN 2.0A (standard data frame) and the RTR (Remote Transmission Request) bit, which is dominant for data frames. For CAN 2.0B (extended data frame) it contains the 29-bit identifier and the RTR bit. A dominant IDE (Identifier Extension) bit indicates that a CAN message is in standard format (CAN2.0A). A recessive IDE bit indicates that the CAN message is in extended format (CAN2.0B). Next is the data field which contains from zero to eight bytes of data, and the CRC field which contains the 15 bit checksum used for error detection. Lastly, there is the acknowledgement field. Any CAN controller that is able to correctly receive a message sends

a dominant ACK (Acknowledgement) bit that overwrites the transmitted recessive bit at the end of correct message transmission. The transmitter checks for the presence of the dominant ACK bit and re-transmits the message if no acknowledge is detected. A message is considered to be error free when the last bit of the ending EOF (End of Frame) field of a message is received in the error–free recessive state. A dominant bit in the EOF field causes the transmitter to repeat a transmission.[14]

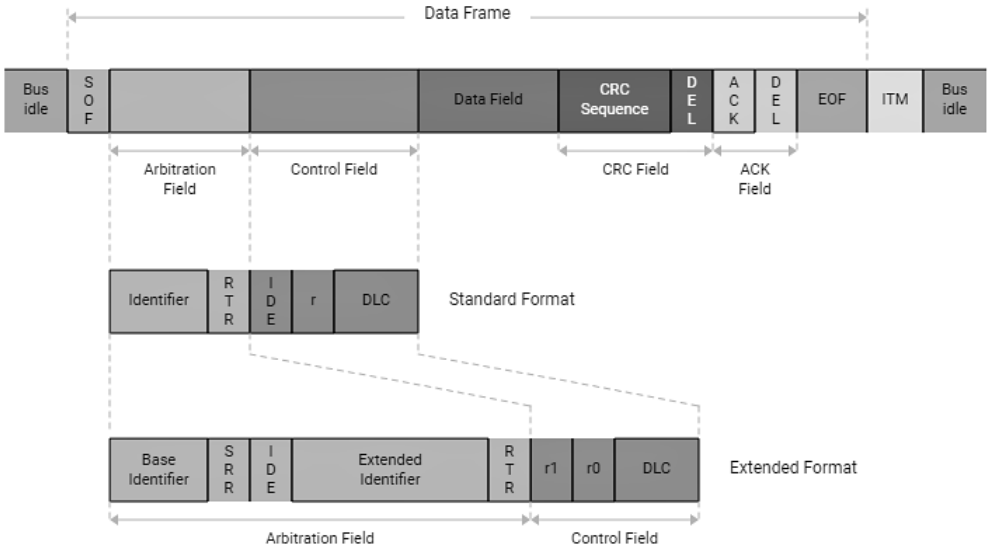

Figure 2.10: The CAN Data Frame.[8]

2) The Remote Frame: The intended purpose of the remote frame is to solicit the transmission of data from another node. The remote frame is similar to the data frame, with two important differences. First, this type of message is explicitly marked as a remote frame by a recessive RTR bit in the arbitration field, and secondly, there is no data.

3) The Error Frame: The error frame is a special message that violates the formatting rules of a CAN message. It is transmitted when a node detects an error in a message, and causes all other nodes in the network to send an error frame as well. The original transmitter then automatically re-transmits the message. It consists of just two parts: The error flag and the error delimiter.

4) The Overload Frame: The overload frame is similar to the error frame with regard to the format, and it is transmitted by a node that becomes too busy and requires more time to retrieve incoming messages from its receive buffer. It is primarily used to provide for an extra delay between messages.

2.2.2.6 Error Handling

Error detection and error handling are important for the performance of CAN. Because of complementary error detection mechanisms, the probability of having an undetected error is very small (4.7e-11 or less)[14]. Error detection is done in five different ways in CAN: bit monitoring and bit stuffing, as well as frame check, ACK check, and CRC.

Bit monitoring simply means that each transmitter monitors the bus level, and signals a bit error if the level does not agree with the transmitted signal.

After having transmitted five identical bits, a node will always transmit the opposite bit. This extra bit is neglected by the receiver. The procedure is called bit stuffing, and it can be used to detect errors and also provide as a synchronization mechanism.

The frame check consists of checking that the fixed bits of the frame have the values they are supposed to have, e.g., EOF consists of seven recessive bits.

During the ACK in the message frame, all receivers are supposed to send a dominant level. If the transmitter, which transmits a recessive level, does not detect the dominant level, then an error is signaled by the ACK check mechanism.

Finally, the CRC is that every receiver calculates a checksum based on the message and compares it with the CRC field of the message. Every receiver node obviously tries to detect errors within each message. If an error is detected, it leads to an immediate and automatic retransmission of the incorrect message. In comparison to other network protocols, this mechanism leads to a high data integrity and a short error recovery time. CAN thus provides elaborate procedures for error handling, including retransmission and re-initialization.

2.3 Wireless Communication

Wireless communication in tough, demanding applications is not new. Wireless communication has been used for more than 30 years, however, only recently its use in industrial domains became an option, given the strong improvements in reliability and bandwidth that were developed in the meanwhile. During the last years, standards like Wireless LAN (IEEE 802.11), Bluetooth technology (IEEE 802.15.1) and ZigBee (IEEE 802.15.4) have become the dominating wireless technologies for the embedded systems realm. In 2011, Bluetooth Low Energy technology also entered the scene.

The use of this type of technologies provides for greater mobility and also for an easier integration of devices into the network. It also allows faster and easier installations and freedom when and where cabling is an issue.

IEEE 802: IEEE 802 is a family of IEEE standards dealing with local area, personal area and metropolitan area networks.

The services and protocols specified in IEEE 802 map to the bottom two layers (Data Link and Physical) of the seven-layer OSI networking reference model. IEEE 802 splits the OSI Data Link Layer into two sub-layers named Logical Link Control (LLC) and Medium Access Control (MAC), so the layers can be listed like this:

- Data Link Layer
  - Logical Link Control sub-layer
  - Medium Access Control sub-layer
- Physical Layer

2.3.1 Wi-Fi Technology

Wi-Fi was the name given by Wi-fi Alliance to devices that are compatible with the 802.11 standard. It is one of the most popular wireless communication standards available, if not the most. This standard describes the characteristics of a wireless local area network (WLAN). It is well-suited for monitoring, configuring and data acquisition, but can also be used for time-critical control in the same applications. Furthermore, the built-in roaming functionality is useful in factory automation applications with moving devices.

In its initial phase, the Wi-Fi technology was almost exclusively used to connect portable computers to the Internet through Local Area Networks (LANs), but thanks to the flexibility and bandwidth it provides, Wi-Fi is now also found in a number of devices such as home theater receivers, game consoles, DVD players, digital cameras and even GPS devices. Typically the IEEE 802.11 standard reserves the lowest layers of the OSI model for a wireless connection that uses Radio-Frequency (RF) waves. The physical layer defines the modulation of radio waves and the signaling characteristics for data transmission, while the data link layer defines protocols that allow the transmission of bits of information using the services of the physical layer.

IEEE 802.11 is in fact a set of standards. There are multiple types of IEEE 802.11, and even though they have different letters as suffixes, it doesn't mean that one necessarily supersedes another.

IEEE 802.11a and IEEE 802.11b were released in 1999, and are still widely popular. IEEE 802.11a is centered around 5 GHz. It has a high data rate potential approximately 54 Mbit/s.

IEEE 802.11b is centered on the 2.4 GHz range, and has a maximum data rate of 11 Mbit/s but it is a low speed technology that was supplanted by faster alternatives. One potential problem is that the 2.4 GHz band is fairly crowded, with microwaves, Bluetooth, cordless telephones, and baby monitors all occupying this range. Most modern devices will "frequency hop" to avoid conflict with other devices.

There is also 802.11g that provides up to 54 Mbit/s in the 2.4 GHz band, and IEEE 802.11n, which is the most recent one and works both on 2.4Ghz and 5Ghz and with a data rate up to 150Mbit/s. These two became the de facto standards and are the most common flavors of WiFi found today, by far.[15]

One of the disadvantages of this technology is the fact that it requires a lot of energy, giving that the other two (Bluetooth and ZigBee) are a lot more friendly in the power consumption aspect. Despite this, its versatility, reach and data throughput make it a very interesting technology for this kind of project.

2.3.2 Bluetooth Technology

Bluetooth is a communication protocol that was adopted by IEEE standards as the standard IEEE 802.15.1. It is very common nowadays for the exchange of information between devices such as mobile phones, laptops, computers, printers and digital cameras through a short-range radio frequency link. It is therefore a specification for Wireless Personal Area Networks (WPANs).

The IEEE 802.15.1 standard specifies the architecture and operation of Bluetooth devices, but only as far as physical layer and medium access control (MAC) layer operation is concerned (the core system architecture). Higher protocol layers and applications defined in usage profiles are standardized by the Bluetooth SIG.

Bluetooth technology is well-suited for wireless integration of automation devices in serial, fieldbus and Ethernet networks. Bluetooth technology is specially suitable for devices with high demands on small footprint, low power consumption and cost-efficiency.

Communications made by Bluetooth operate in the 2.4Ghz band. Some of the key elements of this type of communication are the low sensitivity to reflections, the multiple paths that the signals can take, allowing for an existence of multiples channels operating at the same time for the same receptor. Other key element is the detection and error correction (FEC) and the adaptability to changes in frequency (AFH) that allows safer communications.

Bluetooth Low Energy: Bluetooth v4.0, with the hallmark feature of Low Energy technology, entered the market in 2011. Previously marketed as Bluetooth Smart, it has been a hot topic ever since. The technology has some important limitations as well as benefits and is quite different from Classic Bluetooth technology. In a Bluetooth application where streaming data is used, Classic Bluetooth technology is the preferred choice as it achieves substantially greater throughput than Bluetooth Low Energy technology, so in that way, Bluetooth Low Energy technology is ideal for episodic or periodic transfer of small amounts of data.

	Classic Bluetooth (v4.0)	**Bluetooth LE (v4.0)**
Maximum Data Rate	2 Mbit/s	100 kbit/s
Range	300 m	250 m
Scalability	+	++
Low latency	+++	+++
Connection set-up speed	+	+++
Power consumption	++	+++
Cost	++	+++

Table 2.1: Comparison between Classical Bluetooth and Low Energy Bluetooth regarding the 4.0 Core Specification. Adapted from [15].

However, The Bluetooth SIG presented Bluetooth 5 on 16 June 2016. This version has proved to be a real update that significantly increases the range, speed, and broadcast messaging capacity of Bluetooth applications, making use cases in smart home automation, enterprise, and industrial markets a reality. New innovations built on the Bluetooth 5 specification can take advantage of four times greater range or double the bandwidth. Four times the range helps to deliver an

extended, more robust connection, enabling outdoor use cases. By doubling the speed, without increasing energy consumption, we benefit from faster data transfers in critical applications, such as Bluetooth modules increasing responsiveness and lowering latency. With eight times the capacity for data transfer when broadcasting, Bluetooth 5 can transmit richer, more context-relevant data, propelling the next generation of connectionless services.

2.3.3 ZigBee Technology

ZigBee is a technology based on the IEEE 802.15.4 standard for WPAN (Wireless Personal Area Network). This standard allows for the creation of low cost and low power networks - aiming at applications that run for years rather than months, without requiring user intervention. These networks are created from sensors and actuators and can wirelessly control many electrical products. Its promise of providing a simpler, cheaper, more power-efficient WPANs alternative to WiFi and Bluetooth has opened up new data collection possibilities in application areas from industrial controls to medical devices to intruder alarms. Thus, a objective of this standard is also an attempt to standardize the development of applications and devices in this area, abandoning distinct systems to give way to a standard, that is compatible and provides clear advantages of inter-operability.

The allowed raw data rate is high enough to satisfy a set of simple needs such as interactive toys, but is also scalable down to the needs of sensor and automation needs using wireless communications. It also has the possibility to withstand high number of nodes per network (maximum of 65535 devices) when compared to Bluetooth (maximum of 8) and still have relatively lower intakes.

Similarly to the previously referred protocols, ZigBee also operates in the ISM bands (Industrial, Scientifical and Medical), which are exempt from licensing. Namely, it can use the band of 2.4Ghz and still the bands of 915Mhz (United States) and 868 Mhz (Europe). Depending on the band, the possible transmission rate varies: within 2.4Ghz 250Kbit/s can be obtained, with 16 channels available; to 915Mhz, a rate of 40Kbit/s and 10 communication channels; in the case of 868Mhz, enables 1 channel and 20Kbit/s transmission rate. In terms of modulation, O-QPSK (Offset quadrature phase-shift keying) is used for the 2.4 GHz band and BPSK (binary phase shift keying) for the 915Mhz or 868Mhz.[16]

The main objectives of a Low Rate WPAN (LR-WPAN) like ZigBee are ease of installation, reliable data transfer, short-range operation, extremely low cost, and a reasonable battery life, while maintaining a simple and flexible protocol.

2.4 Related Work

2.4.1 CANlink Bluetooth

CANlink Bluetooth is designed to transfer CAN data wirelessly using the Bluetooth standard. The device can be used in two different ways: either as a wireless CAN bridge or as CAN to Bluetooth interface. In case of the CAN bridge two CANlink Bluetooth can be used to transfer CAN data

between each other in order to replace a wire. If CANlink Bluetooth is used as single CAN to Bluetooth interface, its major application is to transfer CAN data to mobile Bluetooth devices such as PCs or PDAs for CAN data monitoring and analysis.

CANlink Bluetooth is able to establish a connection to a Bluetooth device automatically. If more control is necessary e.g. to change the Bluetooth destination while being in full operation, CANopen commands can be used for connection management. If this functionality is not needed, the CANopen stack of the device can be disabled in order to enhance the range of CAN IDs that can be transmitted by the device.

CANlink Bluetooth is equipped with a 16-bit microcontroller with an integrated full-CAN-interface and a flash memory that can be programmed via Bluetooth interface.

It supports Bluetooth Classic (version 4.0).

Figure 2.11: CANlink Bluetooth device.[17]

2.4.2 Ixxat CANblue II

The CANblue II allows multiple CAN networks to be connected wirelessly via Bluetooth technology. This device forwards from the CAN network received messages to the Bluetooth connection. The messages that are received via a Bluetooth connection are transmitted to the CAN network and other existing Bluetooth connections.

Using several CANblue II devices network segments can be networked among each other. This enables the reliable connection of rotating or moving parts, e.g. in wind turbines, as well as the bridging of larger distances between different sub-systems up to 200 meters. Failure-prone slip ring connections or expensive installations of large CAN lines can in fact be avoided.

The Bluetooth-based communication also offers significant advantages in terms of range and noise immunity, compared to traditional WLAN solutions.

It's main features and benefits are:

- Quick start-up and connection;

- High transmission range;

- Low latency transmission and high data throughput;

- Flexible configuration options (combination of several CANblue II).

Figure 2.12: CANblue II device.[18]

2.4.3 Anybus Wireless Serial Bridge

Anybus Wireless Bridge Serial - Bluetooth is a rugged dual-mode Bluetooth (Bluetooth Smart Ready) serial-port adapter with Bluetooth gateway functionality and u-blox Serial Port Service. It allows connection for up to seven serial slaves with an RS-232/422/485 interface over Bluetooth, with a maximum range of up to 300 meters.

The Bluetooth Access Point features connectBlue Low Emission Mode which solves any potential interference between classic Bluetooth and Wi-Fi that is not handled by Adaptive Frequency Hopping (AFH).

Figure 2.13: Anybus Wireless device.[19]

2.4.4 C2BT - CAN Bus to Bluetooth Adapter

The C2BT is capable of transmitting and receiving both standard and extended CAN frames via Bluetooth. Distances up to 15m may be achieved with the integrated Bluetooth Antenna. Exact timing analysis may be done through switchable timestamps according to incoming CAN Messages. CAN data rates, filter and acceptance mask settings are configurable via Bluetooth. C2BT can not form a bridge between two independent CAN networks.[20]

Application Examples:

- Transmission of SOC and Transaction data (State of charge – eBikes);
- Machine Configuration and Parameterisation;
- Wireless CAN Diagnostics and CAN Debugging;

- Monitoring.

2.5 Remarks

Over the years, CAN continues being used, resisting even the competition of other buses with higher transmission rates, such as FlexRay. Mainly higher costs for the hardware as well as an immense development effort to switch systems to new technologies were the main reason. However, CAN does exhibit some limitations with respect to ultra-high reliability applications. In addition, only a maximum of 1Mbit/s can be transmitted with CAN. This lack of bandwidth often is circumvented by applying a higher number of CAN buses. But this involves using gateways to transfer data between these buses.

As far as standardized wireless technologies, they all cater for different needs and specifications. One wireless technology cannot offer all the features and strengths that fit the various application requirements. As we have seen, there is no one-for-all technology for industrial wireless communication. On a general note, we can say that if high data throughput is the most important feature, than WiFi should be the choice. If connection robustness/stability or cost efficiency is more important, then Bluetooth should be considered. ZigBee won't probably be chosen because of its very limited throughput, but it's easily the most scalable.

Various existent solutions that provide interconnection between some of these wired and wireless communications have also been described in this chapter. A small summary can be found in the next table:

Product Name	**Manufacturer**	**Main Features**
CANlink Bluetooth	Proemion	Can be used as CAN bridge and as diagnosis interface; Uses Classic Bluetooth SPP (v4.0); Range is about 300m.
Ixxat CANblue II	HMS Industrial Networks	Can be used as CAN bridge and as diagnosis interface; Uses Classic Bluetooth SPP (v4.0); Bluetooth transfer delay up to 4ms average; Range is about 200m.
Anybus Serial Bridge	HMS Industrial Networks	Can be used as a serial wireless bridge; Supports serial rates up to 460.8 kbit/s; Uses Classic Bluetooth SPP (v4.0); Range is about 300m.
C2BT	Case	Can only be used as diagnosis interface; Supports CAN rates up to 1Mbit/s; Uses Classic Bluetooth SPP; Range is about 15m.

Table 2.2: Comparison of existent solutions for wireless gateways.

Although these kind of products already exist on the market AddVolt opted for the development and implementation of its own CAN wireless gateway for the following reasons:

- Third-party independence, for instance, by using a third-party solution AddVolt would be vulnerable to price volatility as well as for technical specifications, meaning that any modifications made to the product could have a negative effect on AddVolt's product which they would not have control over.
- Direct integration into the hardware and firmware that AddVolt develops which helps minimizing the cost and having total control of the design and features of the solution. It avoids having more devices segregated and installed in the vehicles.

Chapter 3

Bluetooth Low Energy

Bluetooth Low Energy (BLE) started as part of the Bluetooth 4.0 Core Specification. It is tempting to present BLE as a smaller, highly optimized version of its bigger brother, Classic Bluetooth, but in reality, BLE has an entirely different lineage and design goals. While Classic Bluetooth is mostly used for data streaming, BLE data rates restrain its applications to sporadic, short range, data transmissions, while accomplishing very low power consumption. BLE achieves it because it has a lower duty cycle and a lower power peak.[21] Modularity was also a main concern while designing BLE. The technology can effectively keep up with the evolution of the Internet of Things (IoT), requiring minimal future changes, and effectively accommodating those that may be required. BLE does not support backward compatibility, as devices that support only Classic Bluetooth communications cannot communicate with BLE devices.

Since its implementation in 2010, Bluetooth Low Energy has developed through the ratification of Bluetooth revisions 4.1 (December 2013), 4.2 (December 2014) and 5.0 (December 2016).

Bluetooth Low Energy has seen an uncommonly fast adoption rate for a comparatively young standard, and the amount of product designs that already include BLE puts it well ahead of other wireless technologies at the same stage in their release cycles. BLE has gone further faster because it is so closely linked to the tremendous emergence of smartphones, tablets and mobile computing. More specifically, the very early adoption of this technology by relevant businesses such as Apple and Samsung enabled broader implementation of Bluetooth Low Energy.[22]

3.1 Protocol Stack Overview and Basics

The Bluetooth Low Energy stack consists of two separate detached parts: a controller and a host. The controller comprises the Physical and Link Layer. The host comprises the upper layer functionality of the stack, such as the Link Controller and Adaptation Protocol (L2CAP), Attribute Protocol (ATT), Generic Attribute Protocol (GATT), Security Manager Protocol (SMP) and the Generic Access Profile (GAP). The host and the controller communicate through an interface called the Host Controller Interface (HCI) that acts as a two-way communication channel. The layers of the stack can be put together as follows:

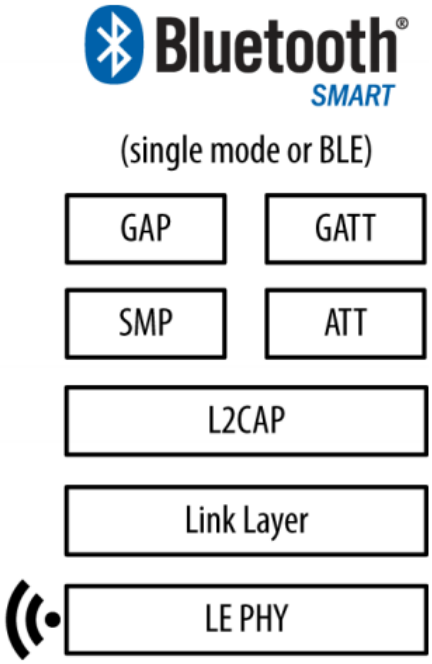

Figure 3.1: The Bluetooth Low Energy Protocol Stack.[21]

- **LE PHY:** The bottom layer specifies features from the BLE technology such as frequency, number of RF channels, and modulation. It contains the analogue circuitry to transmit and receive the radio signal, using the 2.4GHz ISM band, which is divided into 40 channels, spaced 2MHz apart from each other. The last three channels (37, 38 and 39) are reserved to advertising purposes only, such as setting up connections and sending broadcast data.
- **Link Layer:** This layer is directly interfaced with the Physical Layer. It is the only layer with hard real-time constraints, as it has to comply with all the timing requirements set by the upper layers. It defines how the physical channels are used, the packet format as well as the air interface protocol which consists of multiple access scheme, device discovery, connection establishment and maintenance, and data transfer.
- **L2CAP:** Transmits standard BLE packages to the controller. Also acts as the protocol multiplexer, responsible for routing incoming packages to either the ATT layer or SMP layer.
- **SMP:** The security manager is responsible for the low level security operation needed in order to establish a secure connection, such as managing integrity during the pairing procedure, authentication, and encryption between Bluetooth Low Energy devices.
- **ATT:** In the top layers of the Bluetooth LE stack, data is based on attributes. An attribute is a data container coupled with identifiers and security parameters for the data. How these attributes are defined and how they can be accessed and interacted with is defined in the ATT layer of the stack.

- **GATT:** The Generic Attribute Profile makes use of the Attribute Protocol and defines the types of attributes and how they may be used. It introduces concepts such as services and characteristics, the relationships between them and procedures to discover and access them. This is often considered the backbone of BLE communication as it defines how data is organized and sent between different applications.
- **GAP:** This layer specifies some operational modes and procedures, such as device discovery and connection establishment, the data format from advertising packets and also some security aspects.

As said in the beginning of this section, the HCI offers an uniform interface. This interface is based on commands and events. The specification describes several HCI commands that the host uses to instruct the controller to perform operations such as scan nearby devices, connect to devices, transfer data, terminate connection, etc. The controller sends HCI events to the host to inform the status of those commands (e.g. success, fail, busy, etc.).

Finally, on the top of this stack are the Applications, that use profiles and the services provided by them to implement a certain functionality or to cover a particular use case.

Bluetooth Low Energy defines something called a Bluetooth profile. Bluetooth profiles are often grouped with the Bluetooth protocol, but displays some key differences.

- **Protocol:** A core part of all Bluetooth devices. It's the layers that include packet formatting, routing, encoding, and other elements required to send data between two systems.
- **Profile:** Essentially, profiles describe how protocols should be used to accomplish a certain objective. Is either a element of the functionality that describes basic operations such as reading, writing or representing data (GATT, GAP) or a way to accomplish a particular case of use.

Bluetooth profiles are described so that a rigid hierarchical structure can be enforced and profiles can be constructed on top of each other. At the moment, all BLE profiles have to implement both GATT and GAP as these are the top most BLE communication data and control layers. Bluetooth SIG describes a few use-case-specific Bluetooth profiles aimed as a standardized route to accomplish certain tasks.[23]

3.1.1 Bluetooth Low Energy packet

The Bluetooth Low Energy packet structure is defined by the Link Layer. The individual parts of a BLE packet are shown in the next figure.

Figure 3.2: General parts of a BLE packet.[24]

- The Preamble is a fixed sequence of zeros and ones to support the receiver, e.g. time synchronization and Automatic Gain Control (AGC).
- The Access Address is a random value that identifies the access to a physical channel.
- The Protocol Data Unit (PDU) carries the actual message. This includes both the transmitter and receiver addresses as well as user data, for example.
- Cyclic Redundancy Check (CRC).

The packets are mainly differentiated by their PDU and the Bluetooth addresses are defined directly in the PDU payload and are 6 bytes (48 bits) long. Addresses can be of two types, *Public* or *Random*. Public device address is a fixed, factory-programmed device address, while the Random device address may be pre-programmed or generated at run time.

3.1.2 Bluetooth Low Energy states

The Link Layer defines a state machine with five states which is implemented by the BLE controller. The states are: Standby state (no packets are sent or received), Advertising state (send advertising packets and possibly respond to requests), Scanning state (listen for advertising packets and possibly send requests to advertisers), Initiating state (listen for advertising packets from a specific device and initiate a new connection), and Connection state (connection is established). A device can run multiple states simultaneously, e.g. to support advertising and scanning in parallel.

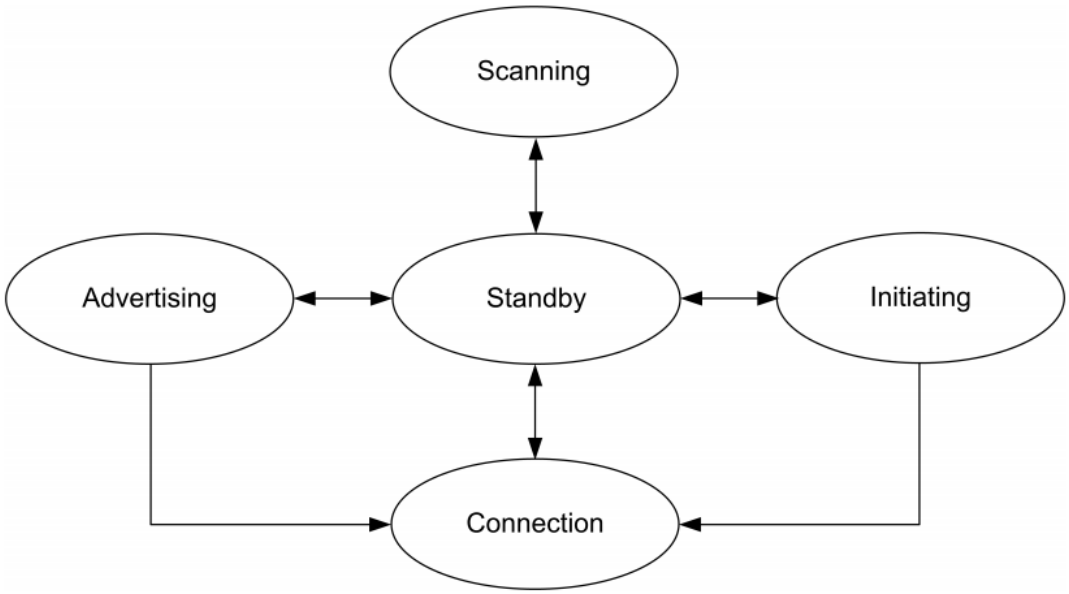

Figure 3.3: State machine for Bluetooth Low Energy.[24]

3.1.3 Bluetooth Low Energy roles

Bluetooth Low Energy describes a few distinct roles depending on the level of abstraction and the situation proposed.

The Link Layer of the BLE stack defines four roles:

- **Master** The device responsible for initiating a connection to a slave and maintaining it when connected.
- **Slave** The device that accepts connections from a master.
- **Advertiser** A device responsible for sending advertising packets, making it discoverable for scanners.
- **Scanner** A device scanning for advertisement packets.

The GAP layer of the BLE stack also defines four roles. These roles always correspond to one or more of the Link Layer roles:

- **Central** The Central is responsible for scanning for peripherals and initiating connections. The central is also responsible for maintaining several concurrent connected slaves. It corresponds to the Link Layer Scanner and Master.
- **Peripheral** When a peripheral is not connected to a central, it transmits advertisement packets to help a central find and connect to it. It corresponds to the Link Layer Advertiser and Slave.
- **Broadcaster** A device optimized for regular broadcasting of data. A broadcaster device transmits data by sending advertisement packets, rather than requiring a connection. It corresponds to the Link Layer Advertiser.
- **Observer** A receive-only device responsible for scanning for packets sent by a broadcaster. It corresponds to the Link Layer Scanner.

ATT, GATT layers and BLE profile also describe a server / client communication protocol at the top of the L2CAP layer. The client application can discover the server attributes and read or write attributes by applying the GATT defined procedures. The BLE profiles standardize the application specific attributes. For example, any application which implements the temperature profile should define the same set of attributes. The client and server roles are determined by the BLE profile and they are independent of the slave or master role.[25]

3.2 Bluetooth Low Energy security

Bluetooth Low Energy offers a diverse range of security methods to protect the exchange of information between two connected devices. Security in BLE is especially important when there is a need to deal with older and less secure Bluetooth devices. The fact that they might need to interact

with new BLE devices which are capable of better security makes it important to determine what level of security is actually being used.

As said before the SMP is responsible for pairing, encryption and signing so the application (that sits at layer 7 on the OSI model) does not have to worry about the security details and therefore it does not care about the lower layers various processes so long as it can send and receive data.

If we want to do something that needs safety, the systems must first pair. This process is triggered by a Central device that is attempting to access a characteristic on a Peripheral device that requires authenticated access. Pairing involves authenticating the identity of two devices, encrypting the link using a short-term key (STK) and then distributing long-term keys (LTK) used for encryption. The LTK is saved for faster re-connection in the future. That is called **Bonding**. If a Central wants to bond with a Peripheral device it believes is bondable, it uses the bondable procedure. It initiates pairing with the same bonding bit set in the authentication requirements of the Pairing Request message. If the Peripheral device is bondable, it will respond with the bonding bit set. If all this happens, the keys will be distributed after the link is encrypted and they will be stored. Once this happens, the pair of devices are said to be bonded.

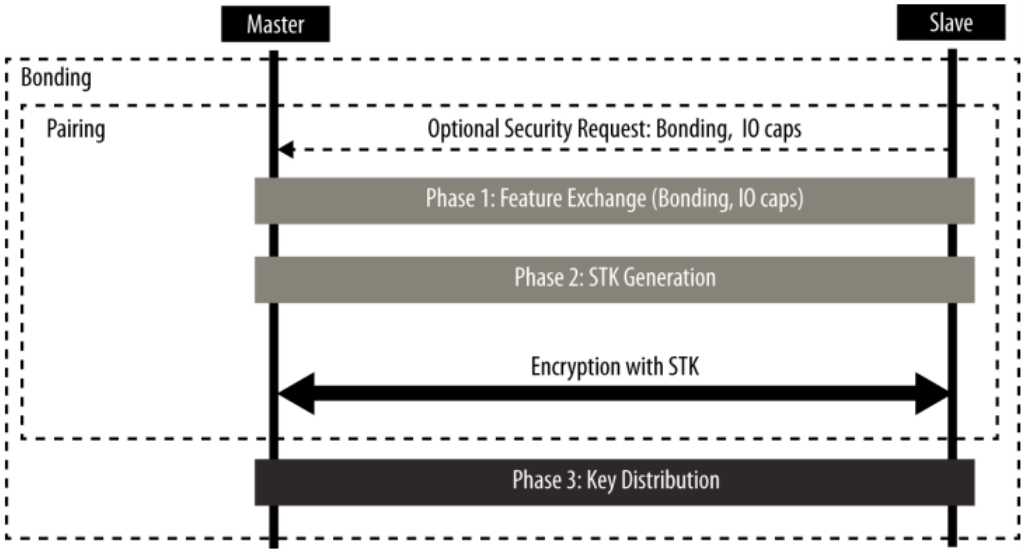

Figure 3.4: BLE Pairing and Bonding sequences.[21]

Encryption in BLE is based on 128-bit Advanced Encryption Standard—Counter with CBC-MAC (AES-CCM). LTK is used with this algorithm to create the 128-bit “shared secret” key.

Authentication is provided in BLE by digitally signing the data using the connection Signature Resolving Key (CSRK). The sending device places a signature after the Data PDU. The receiver verifies the signature using the CSRK.

3.2.1 Pairing Procedure

A pairing procedure involves an exchange of SMP packets to generate the temporary encryption key (STK) as depicted in figure 3.4. During the packet exchange, the two peers negotiate one of the following STK generation methods[26]:

- **Numeric Comparison** was intended for the scenario where both systems can display a six-digit number and allow a user to enter a "yes" or "no" answer. A user is displayed a six-digit amount on each device and gives a "yes" reply on each device if the numbers match. Otherwise, the user replies "no," and the pairing fails. The displayed number is not used as input for link key generation. Therefore, a sniffer capable of capturing the displayed value could not use it to determine the resulting link or encryption key.
- **Just Works** was designed for a scenario where at least one of the pairing systems does not have a display or a digit keyboard (e.g. headset). The first phase is equivalent to the Numeric Comparison model, except that there is no display accessible. The user is required to accept a link without checking the calculated value on both units, so Just Works does not provide MITM security.
- **Passkey Entry** was constructed for the situation where one Bluetooth device has input capability (e.g., keyboard), while the other device has a display but no input capability. In this model, the device with only a display shows a six-digit number that the user then enters on the device with input capability. As with the Numeric Comparison model, the six-digit number used in this transaction is not incorporated into link key generation and is of no use to an eavesdropper.
- **Out of Band** (OOB) uses an external means of communication to exchange some information used in the pairing process. The OOB media could be any other wireless communication standard that can deliver the respective data for pairing, such as NFC or QR code.

3.2.2 Security Modes and Levels of a connection

Realize that safety levels and security modes can be mixed. First, starting with the security levels:

- **Security Level 1** No security (no authentication and no encryption).
- **Security Level 2** Unauthenticated pairing with encryption.
- **Security Level 3** Authenticated pairing with AES-CCM encryption.
- **Security Level 4** Authenticated LE Secure Connections pairing with encryption. Level 4 uses Elliptic Curve Diffie-Hellman P-256 (ECDH) and AES-CCM encryption.

And the security modes:

- **Security Mode 1** Comprises all the levels without signing of data.

- **Security Mode 2** It is those same levels with signing of data, including both paired and unpaired communications.
- **Mixed Security Mode** It is when a device is required to support both Security Mode 1 and 2, i.e., it needs to support signed and unsigned data.
- **Secure Connection Only Mode** It is Secure Mode 1 with Security Level 4, meaning that all incoming and outgoing traffic in a Bluetooth device involve authenticated connections and encryption only.

3.2.3 Security Manager Protocol packet

For a better understanding of how a BLE pairing is actually configured it is best to take a look at the SMP packet. If we "are" the initiator we're referring to the Pairing Request packet. Conversely, if we are talking about the receiver device, we want the Pairing Reply packet.

<table>
<tr><td rowspan="2">Opcode
1 byte</td><td rowspan="2">I/O Capability
1 byte</td><td rowspan="2">OOB Data Flag
1 byte</td><td colspan="5">Auth Request
1 byte
(Bit order LSB to MSB)</td><td rowspan="2">Max Encryption Key Size
1 byte</td><td rowspan="2">Initiator Key Distribution
1 byte</td><td rowspan="2">Responder Key Distribution
1 byte</td></tr>
<tr><td>Bonding Flags
2 bits</td><td>MITM Flag
1 bit</td><td>Secure Connection
1 bit</td><td>Keypress
1 bit</td><td>Reserved
3 bits</td></tr>
</table>

Figure 3.5: Security Manager Protocol pairing request/reply packet.[27]

The opcode will be 0x01 for a pairing request and 0x02 for a pairing reply.

I/O capacity will be one of the following:
-0x00 Display Only;
-0x01 Display Yes/No (both a display and a way to designate yes or no);
-0x02 Keyboard Only;
-0x03 No Input/No Output (e.g. headphones);
-0x04 Keyboard Display (both a keyboard and a display screen);
-0x05-0xFF Reserved;

OOB Data Flag will be 0x00 for no OOB data or 0x01 for OOB data present.

Max Encryption Key Size tells the size of the encryption key in octets, and both the Initiator and Responder Key Distribution bytes convey with flags what keys will be distributed.

The Auth Request byte consists of five fields, and uses the various bits as flags. From least significant bit to most significant bit:

- **Bonding flags** two bits, either 0 or 1 for the very least bit, the other bit is reserved. Therefore, 0 is for no bonding, the 1 is for bonding. If bonding is used, an LTK will be exchanged, which means that two devices can be paired, and a reboot or sleep mode will not unpair the devices. If encryption is endorsed, this will occur independently after pairing.
- **MITM flag** one bit, 0 is that MITM protection is not requested, and 1 is that MITM protection is requested.
- **Secure connection** one bit. If this is set to 1, the device is requesting to do Secure Connection Only Mode, otherwise it is set to 0.
- **Keypress flag** one bit. If set to 1 it means Passkey Entry needs to be used, otherwise it is ignored.

3.3 Bluetooth Low Energy Connection

The Bluetooth specification defines some connection modes and procedures. The connection modes are implemented by the Peripheral device only and they are as follow:

- The **Non-Connectable** mode do not allow a connection to be established.
- The **Directed Connectable** mode accepts a connection request from a specific Central device
- The **Undirected Connectable** mode accepts a connection request from any Central device.

The connection procedures are implemented by the Central device only and they are the following:

- The **Auto Connection** establishment procedure allows the Host to configure the Controller to autonomously establish a connection with one or more Peripheral devices.
- The **General Connection** establishment procedure allows the Host to establish a connection with a set of known Peripheral devices.
- The **Selective Connection** establishment procedure is very similar to the general procedure but it takes advantage from the Controller's white list so only advertising events from the selected Peripherals are considered during the procedure.
- Finally, the **Directed Connection** establishment procedure allows the Host to establish a connection to a specific Peripheral device.

In general, a Peripheral device enters in connectable mode when it starts sending connectable advertising events and a Central device starts any connection establishment procedure by scanning advertising events.

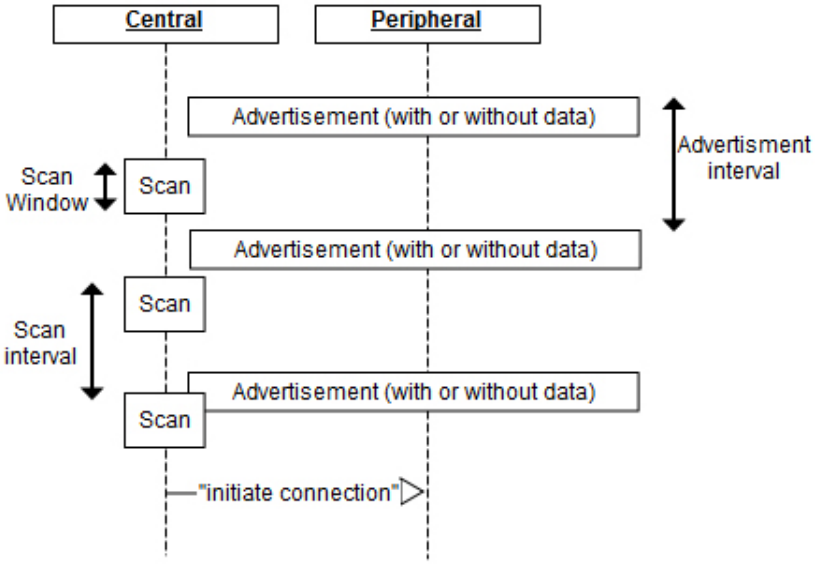

Figure 3.6: BLE connection sequence.[28]

It is not feasible to connect to a Peripheral that is not advertising, even though one knows its address from before. This is because after transmitting an advertisement, the Peripheral only turns on the receiver for a specified duration. This time is used to listen for connection requests and scan requests. Connection requests sent from the Central contain connection parameters for the new connection: the connection interval informs the peripheral how often they should communicate. The channel map describes which channels they shall communicate on, and in which order. The Peripheral does not respond to this request directly If it accepts the connection, it will tune in to the right frequency at the correct moment. If it does this, the connection is established. The Central and Peripheral will continue to meet at the set channels and times as long as the connection is up. Either part can actively disconnect at any given time by telling the other device. The other way to disconnect is when one device stops "meeting up" at designated channels and times. If a certain time has elapsed since the last communication, the remaining device will have a disconnect event.

3.3.1 Data Transfer

Once a link is formed, Central and Peripheral devices can swap information. Data transfer happens in very specific moments, called connection events. During a connection event, the Central and Peripheral alternate sending and receiving packets. The connection event is deemed open while both devices continue to deliver packets. In the following situations, the connection event is closed: 1) nor the Central nor the Peripheral have more data to send; 2) no acknowledgment packet received; 3) two consecutive packets received with an invalid Cyclic Redundancy Check (CRC); and 4) the next connection event is about to open. It's important to know that connection

events may differ in duration. The interval between connection events can be set according to the application requirements.

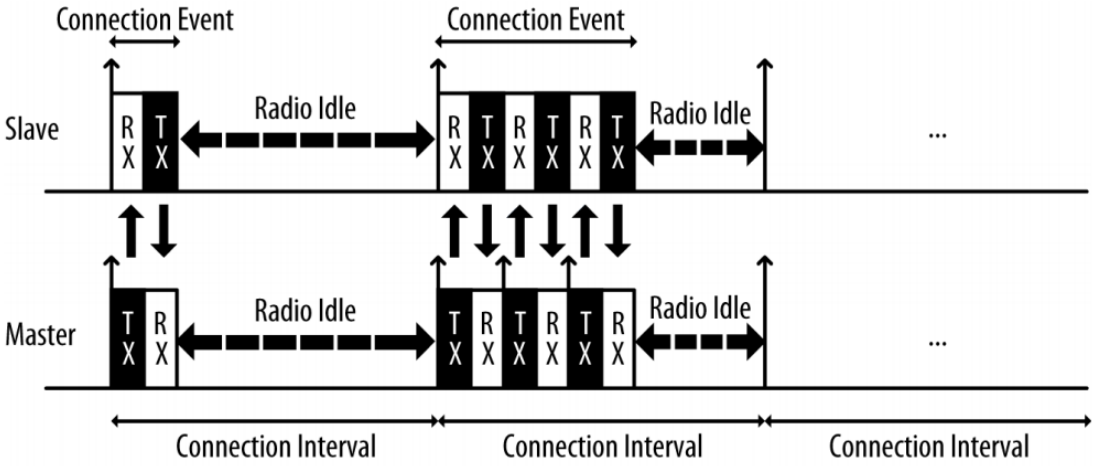

Figure 3.7: BLE connection interval and connection event.[21]

A Peripheral may intentionally ignore some connection events, but the connection is not declared terminated. This feature allows the peripheral to save energy when it does not have any data to send. There are two relevant parameters when talking about a BLE connection. One is called Slave Latency which is the maximum number of events a slave can skip before the central considers the connection as lost. The other one is Connection Supervision Timeout which is the maximum time between two received and validated packets before considering the connection as lost.

3.4 ATT, GATT, and GAP layers

Bluetooth Low Energy data is represented as attributes (ATT), each assigned a 16-bit handle, a Universally Unique Identifier (UUID), a value and a list of permissions. The handle acts as a unique identifier for each attribute and works as an internal address for each attribute. The UUID acts as a type identifier for the attribute and specifies what kind of data it contains.

	Handle	UUID	Value	Permissions
Attribute 1	0x0000	UUID (16 or 128 bit)	0x03	read only
Attribute 2	0xFFFF	UUID (16 or 128 bit)	0x15	write only
Attribute 3	0x3964	UUID (16 or 128 bit)	0x25	read/write

Table 3.1: Example of BLE attributes.

These attributes are grouped according to the GATT layer into services, characteristics, and descriptors. A Server can contain many services, a service can contain one or more characteristics

and a characteristic can contain zero or more descriptors. GATT attributes are stored in a table called a GATT table, which is just a sequential table of the attributes contained in a GATT device. Attributes can not exist outside of this structure if the device wants to be compatible with other GATT devices, as this is the core of how data is represented in Bluetooth Low Energy.

- A **Service** is a group of related attributes, intended to represent the behavior of a part of a system. A Service is represented in GATT as a Service Declaration Attribute, followed by a list of characteristics and/or references to other services.
- A **Characteristic** acts as a data container for user data. A Characteristic Declaration (Containing metadata) and a Value Declaration (The actual user data) is used to represent a characteristic.
- A **Descriptor** is intended to provide additional metadata about a characteristic or the value associated with it. A descriptor is always placed within a characteristic and is comprised of a single attribute. For example, a descriptor can define the unit of the characteristic value.

A client can discover and read GATT attributes from the server. When a new connection is established, the client has no information about what data the server holds. In order to get data from the server, the client has to initiate a so called discovery procedure to find what data the server holds. This allows the server to save energy by only transmitting data requested by the client. Communication between BLE devices is usually based on the client sending read/write requests to the server, allowing the server to stay in sleep mode as long as possible.

The two following figures show how the data flows between the GATT client and GATT server for a write operation both without and with response (unacknowledged and acknowledged).

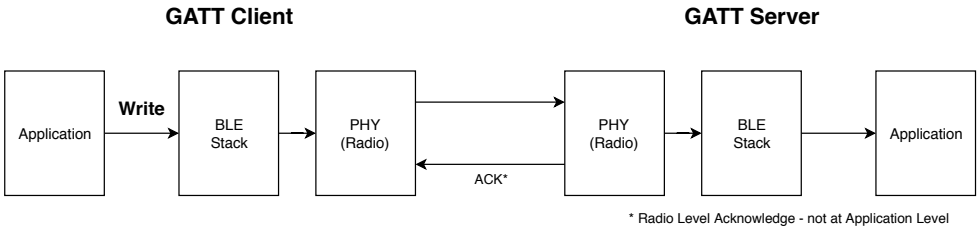

Figure 3.8: Write without response (unacknowledged write).

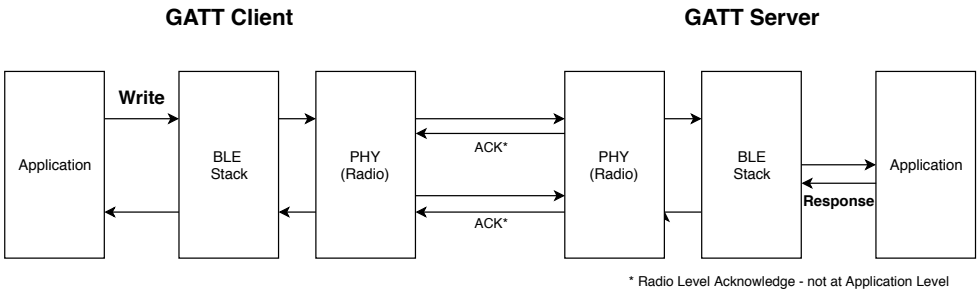

Figure 3.9: Write with response (acknowledged write).

The terms acknowledged and unacknowledged apply to GATT operations. This is not just about the data being transmitted across the radio link but also about the data actually reaching the Application layer. On both acknowledged and unacknowledged GATT operations the data is reliably transported across the radio link. The difference between the two transfer types is in knowing if and when the data sent has actually been received by the application on the other side.

Sometimes though, an asynchronous update from the server can be required. To get these updates without the client having to poll the server periodically (which cost both energy and bandwidth) there are two ways for the server to push packages to the client. This happen through Notifications and Indications. Notifications are unacknowledged and Indications are acknowledged. The data flow is again depicted below.

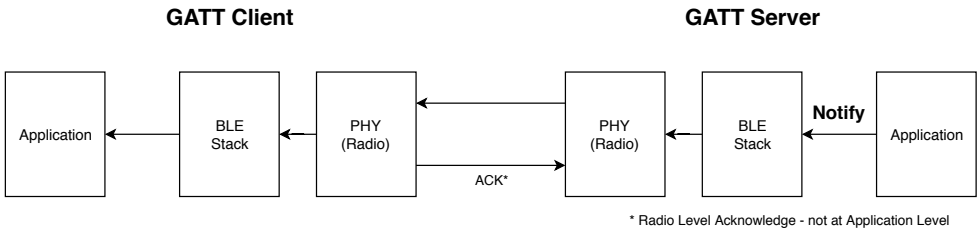

Figure 3.10: A server Notification.

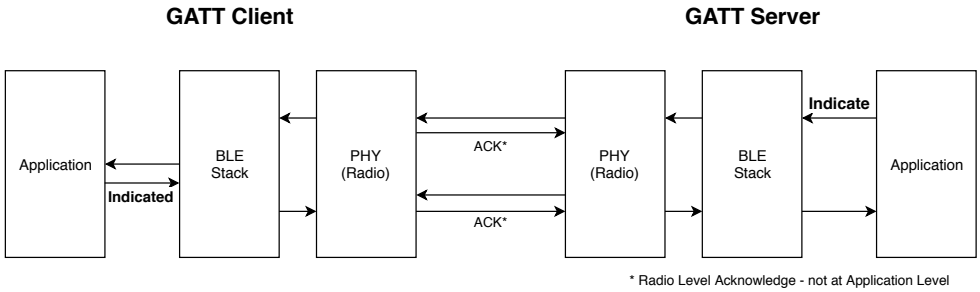

Figure 3.11: A server Indication.

3.5 Key advances in Bluetooth 5

Although Bluetooth Low Energy's first variant was very flexible, variants 4.1 and 4.2 offered minor extensions / improvements. Still, they weren't enough to make Bluetooth an appealing technology for more powerful, challenging manufacturing applications. Many BLE use cases tend to contain small amounts of information. But use cases are gaining prominence, in need of a low-power wireless communication technology that enables increased data rates.

Version 5 further improves several low energy characteristics for IoT applications:

- Two times the data rate.
- Four times the range.

- Eight times the broadcast capacity.

The new LE 2M PHY allows the physical layer to operate at 2 Mbit/s and thus enables higher data rates than LE 1M and Bluetooth 4.

The Host Controller Interface (HCI) promotes a fresh command with which the host is able to invoke Change PHY Procedure. This enables the host to select the PHY at any specified moment. For instance, apps may want to switch to 2Mbit/s mode for use cases where maximum information rates are needed or switch to long-range mode when needed (note that the maximum range multiplier for the 2Mbit/s mode is 0.8x, not 4x).

The LE Coded PHY enables quadrupling the range compared to Bluetooth 4, and this was achieved without raising the necessary transmission energy.

Bluetooth 4 advertising packets are 37 bytes long with a 6-octet header and a maximum payload of 31 bytes. Eight new advertising, scanning and connecting PDUs were added to the Generic Access Profile. These changes make it possible to broadcast much larger amounts of data in connectionless scenarios, to advertise in a deterministic fashion, and to broadcast multiple distinct sets of advertising information.

Contention and duty cycle are also significantly improved. Bluetooth 5 enables packets up to 255 bytes long. This is achieved by offloading the payload to one of the other channels in the 0-36 channel number range, initially only used for connection events.

One of the many interesting things about Bluetooth 5 advertising changes is how radio channels are now being used, with main advertising channels 37, 38, and 39 carrying less information and complementary channels 0-36 doing most of the heavy job.

Bluetooth 5 is another big shift in Bluetooth technology. LE 2M's greater symbol rate increases spectral efficiency and promotes emerging use cases such as sports and fitness and medical equipment. New industrial applications and smart city applications will become viable. Bluetooth 5 will have a major impact in many sectors, making it the preferred low-power wireless technology for the Internet of Things.[29]

3.6 Remarks

While the information provided by section 2.3 helps to achieve a general comprehension about the most common wireless technologies chosen for similar use cases, in this chapter we focus entirely on Bluetooth Low Energy and the improvements implemented in the last Core Specification and how those new features have a positive impact regarding similar use cases like the one in this project. It is important to note that the throughput not only depends on inherent Bluetooth Low Energy characteristics, but also on the how the implementation is designed. If possible, and being that this project demands high data rates, one should use Writes/Notifications instead of WriteRequests/Indications whenever possible.

Chapter 4

System Design and Architecture

In this project, we aim to develop a firmware to interconnect two independent CAN networks that communicate through a specific wireless technology. The aim of this chapter is to define the system architecture and to better specify the project, in terms of overall features as well as the devices that will be integrated in the prototype.

It is requested the implementation said firmware on top of AddVolt's existent framework, therefore there are some requirements that we need to pay attention to before drafting a solution.

4.1 Requirements Analysis

Parallel to the tasks to be carried out in approaching this problem, some requirements have been defined, after consultation with AddVolt to check how its application is designed, to which the gateway has to obey for the good functioning of the system as a whole. The following performance requirements were defined:

1. The communication protocol to be developed should be full-duplex. Both gateways should be able to receive a CAN message and forward it to the wireless link and at the same time they should be capable of receiving a message from the wireless link (sent by the other gateway) and forward it to the CAN network that each one is connected to.
2. The data transmission should be agnostic to the CAN frame in question. The gateway should be capable of transmitting both standard (11 bit ID) and extended (29 bit ID) CAN frames.
3. The gateway should be able to perform with a period of 1ms between CAN messages. It should perform with no errors under the worst case scenario, which is the longest message followed by shortest messages. This has to do with how fast a message in the CAN controller receive buffer can get overwritten by the next one.
4. The specified maximum transmission rate of the CAN protocol is 1Mbit/s and the gateway should be able to work under such circumstance. However, the actual AddVolt implementation still uses 500kbit/s.

5. Taking points 3 and 4 into consideration a careful approach has to be taken with consideration of the average data rates at multiple points of the system (e.g. chose a wireless module in which the UART baudrate will not be the bottleneck of the system).

4.2 AddVolt's Management Logging Unit

AddVolt has developed a Printed Circuit Board (PCB) that goes by the name of Management Logging Unit (MLU). In short, the MLU is where AddVolt handles and processes all the data from the entire system. It uses a 32-bit XMC4500 Industrial Microcontroller based on ARM Cortex-M4. The MLU does not have a built in RF antenna that enables communication for Wi-Fi or Bluetooth. Without a functioning antenna the development of this gateway would just not be feasible. However the MLU integrates mikroBUS in its design.

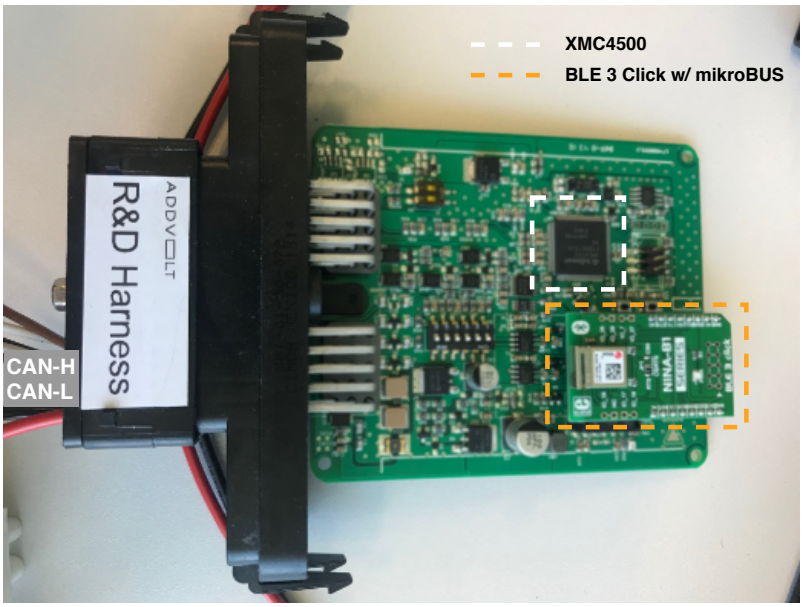

Figure 4.1: AddVolt's MLU with the Bluetooth Module connected via mikroBUS.

The mikroBUS is an add-on board standard, in the embedded electronics market, that offers maximum expandability with the smallest number of pins. The existence of a mikroBUS socket will allow us to chose from a large number of click boards that have a built-in antenna. It is the simplest way to add connectivity to the MLU with its plug and play feature. The mikroBUS socket comprises a pair of 1x8 female headers with a proprietary pin configuration and silkscreen markings. The pinout (always laid out in the same order) consists of three groups of communications pins (SPI, UART and I2C), six additional pins (PWM, Interrupt, Analog input, Reset and Chip select), and two power groups (+3.3V and 5V).

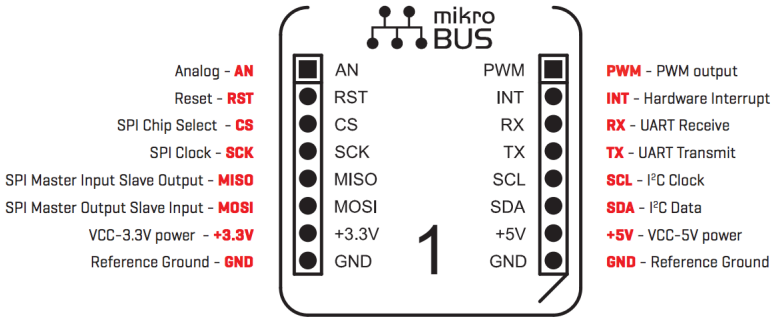

Figure 4.2: mikroBUS.[30]

As a consequence, we're limited to some of the click boards developed by mikroEletronika as a way to add a form of wireless connectivity to our project.[31]

4.3 Development Tools

In this section we present the various tools used to develop our proposed solution.

4.3.1 DAVE v4

Digital Application Virtual Engineer (DAVE) is a C/C++ language software development and code generation tool for microcontroller applications. DAVE is a standalone system with automatic code generation modules and is suitable for developing software drivers for Infineon microcontrollers and helps developers with automatically generated C-level templates and user-desired functionalities.

With DAVE, developers can generate the SW library to efficiently use the innovative application-optimized peripherals. Code generation is based on pre-defined and tested application-oriented SW components, called DAVE APPs.

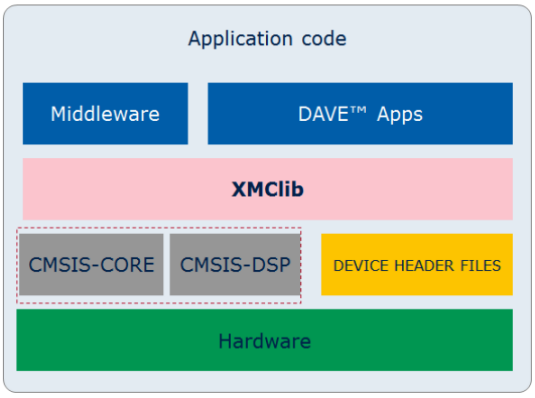

Figure 4.3: DAVE APPs are built on top of XMCLib.[32]

Version 4 provides a faster and more robust design. The time for code generation, responses to user interactions in the graphical user interfaces, and the build time, are all improved significantly. DAVE APPs are based on low level peripheral drivers (XMC Lib), which makes generated code more efficient and more readable. It has routines and data structures for all peripheral functions such as Initialization/Configuration, Event handling, I/O handling and runtime peripheral operation.

The APPs allow the user to configure parts of the hardware or complete functions graphically and according to parameters. The associated code is then automatically generated, and a check is made as to whether more hardware modules are being used than are actually available. The main advantage of this programming is that it is very fast and efficient because the apps are essentially well documented and the configuration menu is clear and intuitive. If somehow an APP is not working or the user wants a more low-level configuration it can resort to the XMClib and go from there.

4.3.2 PCAN-USB and PCAN-view

The PCAN-USB adapter enables simple connection to CAN networks.

Figure 4.4: PeakCAN-USB. [33]

PCAN-View is an easy to use monitoring program to supervise the data flow on a Controller Area Network (CAN). It is based on the PCAN system by PEAK-System Technik GmbH. This system's kernel is responsible for the connection of Windows applications to a CAN Bus connected to the PC in real time manner.

PCAN-View has the following features:

- Received CAN messages are displayed with ID, Data Length, and Data Bytes in a list. Additionally, the number of messages that have the same ID, and the time delay between the last two of those messages are shown.
- Any number of data messages can be added to a Transmit list. These messages can either be transmitted manually or automatically in fixed time intervals.
- Transmit Lists can be saved and loaded to and from files.

- Up to 1,000,000 CAN messages can be recorded by a tracer, either in Ring Buffer or Linear Buffer mode.

This tool will help us test our system sending periodic CAN messages to the first MLU and vary the load on the bus to check how the system behaves. It will also help see the messages that are sent by the second MLU. If the message that is sent to the first one is equal to the one that is sent by the second one the wireless gateway works for discrete messages. We can then slowly increase the frequency of this message to see which is the limit of our application.

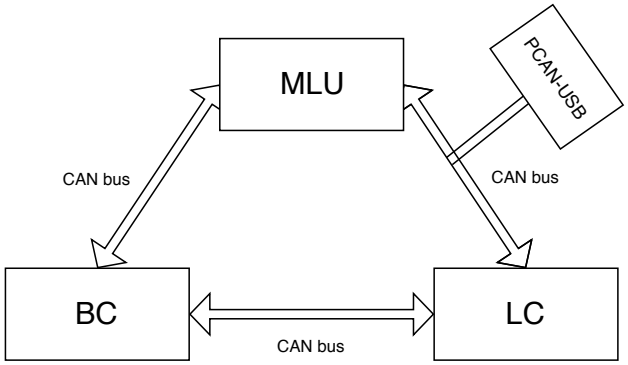

Figure 4.5: AddVolt's network with PCAN-USB sniffing the CAN bus.

The Load Controller (LC) and the Break Controller (BC) are integral part of AddVolt's network but are not relevant for the implementation of the project, therefore they won't be connected to the network in any implementation or test made.

4.3.3 AddVolt Analyze & Monitor

AddVolt Analyze & Monitor (ADDAM) is a program created by the company that allows the user to observe all received CAN messages and to send several types of messages making use of the PCAN-USB.

The messages that concern us are the ones that go in and out of AddVolt's MLU. Therefore we connect the PCAN-USB and the MLU to the same CAN bus as can be seen in figure 4.5.

There is a feature that allows us to keep track of MLU messages that can be converted to ASCII. For example, if we send a message though the CAN bus in which the fourth byte of the payload is an 'A', this window will display the 'A' only. Being that the Wireless Module will communicate with the Host (MLU's microcontroller) via UART, every byte that we receive from the Module will be forward to ADDAM through a CAN message. It is important to note that only messages with the ID 0x5F5 will show up so it is imperative that the MLU sends the message with this specific ID. This way, every time we send an AT command to the host we will be able to check what the response was. This is very useful for configuring the module and for debug purposes.

There is also a feature to do the opposite. The ASCII message feature allow us to configure and send a CAN message in ADDAM. If our MLU is setup to receive any ID, therefore receiving the message from ADDAM, we can then capture the fourth byte of the payload and forward it to

the Wireless Module via UART. This way we can send AT commands to the host very easily as the only thing we have to do is type it on the keyboard and ADDAM will send the CAN message with the fourth byte representing that character in hexadecimal.

4.4 Choosing the Wireless Technology and Click Board

After a research done on Wi-Fi, Classic Bluetooth, Bluetooth Low Energy and ZigBee and knowing that we would be limited to certain click boards from mikroEletronika a few options were quickly eliminated. ZigBee limitations on data rate make it impossible for this use case, therefore it won't be considered. Although Wi-Fi is the technology that allows us to achieve the best throughput via a wireless link, its security limitations are a concerning problem. It also consumes much more energy. Bluetooth is overall a much more robust and stable protocol when it comes to data security. It is also the better option if a connection with more than one device is to be considered as is a very good protocol for scalable networks. Classic Bluetooth is commonly regarded as the best solution for cable replacement and the number of use cases that use this solution proves it. But, as we discussed in section 3.5, the new features of Bluetooth Low Energy after the Core Specification 5 allow for a much higher throughput than before, aligned with sufficient range and sufficient data security for our use case.

As a result we opted for the BLE 3 click board as the Wireless Module of this project. It supports mikroBUS so it is compatible with AddVolt's Management Logging Unit. It also comes equipped with an integrated antenna mounted on the PCB. BLE 3 click is a quick and simple solution to add Bluetooth Low Energy to this project. It features the NINA-B1 Bluetooth module, from u-blox. More specifically the NINA-B112 which comes preflashed with u-blox's u-connectXpress software. The BLE chip on this module will be further described on section 4.5.2.

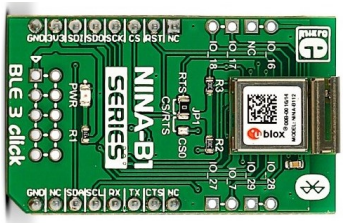

Figure 4.6: mikroE BLE 3 Click Board.[34]

4.4.1 U-blox throughput results

U-blox put out an application note with a series of tests for NINA-B1 in different configurations of low energy data packet length extension (DLE), Long ATT MTU size (MTU) and Physical Layer (PHY).

The S132 SoftDevice may transfer as many packets as can fit within the connection event as specified by the event length for the connection. The event length is equal to the connection interval.

The maximum data throughput numbers given in table 4.1 represent the maximum amount of data that can be transferred between two applications in a given time.

The maximum throughput depends on the mechanism used to transfer data. As said in section 3.4, when the application utilizes Unacknowledged Writes, the transactions are one direction only. When the application utilizes Acknowledged Writes the throughput will be limited to one packet every second connection interval, because it is assumed that the peer responds with a Response in the next connection interval. The amount of data in each packet is the MTU size subtracted by the ATT header size. Therefore, the throughput can be expressed as follows:

$$Throughput(bits/s) = number\_of\_packets * (ATT\_MTU - 3) * 8/seconds \quad (4.1)$$

Device A (Role)	Device B (Role)	PHY	Connection Interval(ms) [min, max]	MTU(bytes)	Dataflow	Throughput
NINA-B1 (central)	NINA-B1(peripheral)	2 Mbps	[7,5;7,5]	247	Simplex	780 kbit/s
NINA-B1(peripheral)	NINA-B1 (central)	2 Mbps	[7,5;7,5]	247	Simplex	780 kbit/s
NINA-B1 (central)	NINA-B1(peripheral)	2 Mbps	[30;50]	247	Duplex	764 kbit/s
NINA-B1(peripheral)	NINA-B1 (central)	2 Mbps	[30;50]	247	Duplex	764 kbit/s

Table 4.1: Data throughput for a single connection. Adapted from [35].

Note that this is the achievable values of the module and is not a guarantee for this implementation. Tests in our environment still need to be done to get proper results for this use case.

4.5 Hardware and Peripherals

4.5.1 XMC4500

Infineon's XMC4500 microcontroller is based on an ARM Cortex M4 core resulting in a power pack for energy-efficient industrial applications. The MCU runs either at an internal system clock or at a frequency generated by a PLL, typically 120MHz. It features a configurable peripheral set that allows to tailor the device to a specific application need.

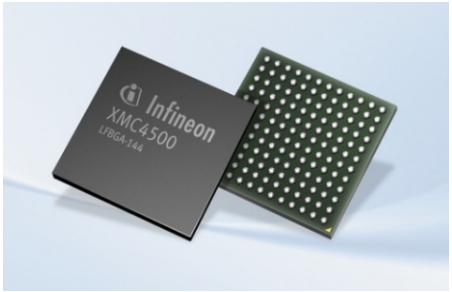

Figure 4.7: Infineon's XMC4500 microcontroller.[36]

Key Features:

- A comprehensive set of highly flexible timers/PWMs, fast and accurate ADCs and position interfaces in combination with a programmable hardware interconnect matrix enable deterministic behavior and full application control.
- 125 degrees ambient temperature for the ultimate robustness in harsh environments.
- 150 ps high-resolution PWM and smart analog comparator for achieving the highest energy-efficiency class for digital power conversion.
- Rich connectivity: 3x CAN2.0B, 64 Message Objects and 4x USIC.
- 1MB Flash and 160kB SRAM.
- Long-term availability with >15 years.

4.5.1.1 XMC4500 Relax Kit

The XMC4500 Relax Kit is a budget-priced evaluation board for the XMC4000 microcontroller family from Infineon built on ARM Cortex-M4. This item is equipped with a detachable on-board debugger to download and monitor code from the DAVE toolchain. This tool allows us to flash our program to the MLU's MCU very easily and very rapidly.

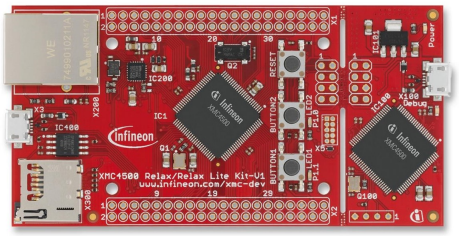

Figure 4.8: Infineon's XMC4500 Relax kit.[37]

4.5.2 NINA-B1

The NINA-B1 module is a small stand-alone Bluetooth Low Energy module featuring Bluetooth 5, a powerful ARM Cortex-M4 with FPU, and an excellent power performance. The embedded low power crystal in NINA-B1 minimizes power consumption, thus extending the battery life. The internal PIFA antenna is specifically designed for the small NINA-B112 form factor and provides an extensive range of more than 300m, independent of ground plane and component placement. Each NINA-B1 module is preprogrammed with a unique 48-bit Bluetooth device address and its UART interface supports baud-rates up to 1 Mbps. In general, it offers a good capacity for customer applications running on top of the BLE stack using SDK from Nordic Semiconductor.

Figure 4.9: u-blox's NINA-B112 module.[38]

In the BLE 3 Click Board, the NINA-B1 module is delivered with the preflashed u-connectXpress software. The u-connectXpress software enables the use of the Bluetooth Low Energy functions, controlled by AT-commands over the UART interface. Examples of supported features are u-blox Low Energy Serial Port Service (SPS), GATT server and client and Central and Peripheral roles. The software on the NINA-B1 module contains the SoftDevice S132 BLE protocol stack solution.

In this implementation the stack will run on the BLE module and our application will run on the microcontroller.

Wireless serial cable replacement feature is one of the module's main characteristics. The fundamental feature is to transfer information between the serial port and a wireless connection. The module can be configured to set up a link automatically and/or accept an incoming link using AT commands. This implies that an existing serial cable can be substituted by a wireless alternative for a host.

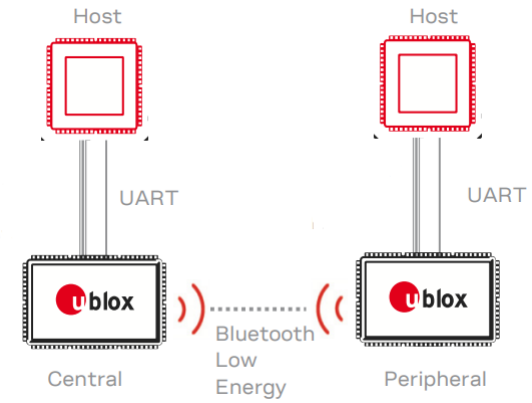

Figure 4.10: Bluetooth Low Energy SPS connection.[39]

The module can operate in the following modes:

- Command mode (default).
- Data mode.
- Extended data mode (EDM).

In the final implementation the module will be operating in Data Mode from the startup in order to have an "always connected" serial cable replacement. But for configuration purposes we will also use Command mode very often.

In the command mode the module can be operated using ATcommands. The command mode implements command set to send data commands and events.

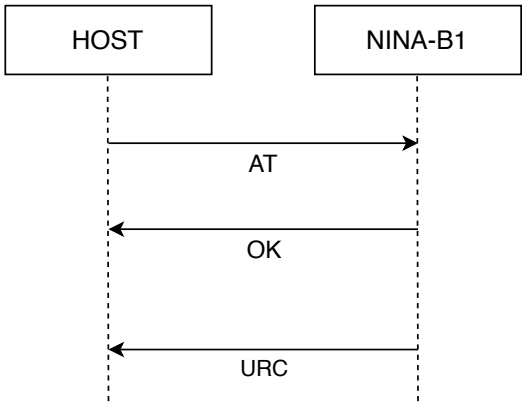

Figure 4.11: Example of an AT command and an URC.

Unsolicited result code (URC) is a string message provided by the module that is not in response to a previous AT command, at any time to inform the module of a specific event or status change. For example, when a connection is established, while disconnecting.

While for the data mode, anything transmitted on the UART, is transmitted over air to the connected remote device. The data received over air from the remote device is transmitted on the serial line without information about the remote device that transmitted the data. Only "raw" data is transmitted. This mode can be represented by figure 4.10.

The module can be configured to start in any mode. Once up and running, it is possible to switch between the modes by sending a command or escape sequence to the module.

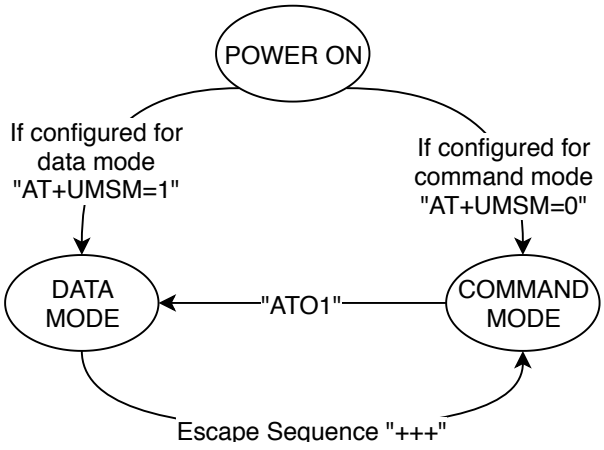

Figure 4.12: State diagram showing operational mode transitions.

Figure 4.12 shows how an AT command can be used to switch from command mode to either data mode. The figure also shows how to change from the data mode to command mode by sending an escape sequence over UART.

As described before, a BLE device either supports the central, peripheral, or both roles. The central is the client and makes connection to the peripheral, which is a server. A server provides a function or service to the client that initiate a request for such service. For the module, this service is an access to a data channel. The client “wants the data” and the server “has the data”.

A connection consists of a sender and one or several receivers of data. Every sender and receiver in a setup is referred to as a peer. A peer can either receive or send data. There are two kinds of peers: a Local peer and Remote peer. The Local peer is synonymous with the UART. The Remote peer is another BLE device. NINA-B1 addresses peers using a Uniform Resource Locator (URL).

The SPS peer is the BLE Serial Port Service. A client can be used to send and receive data to and from a specified URL. For example, to connect SPS to a remote device we use the following URL, "sps://0123456789ABC", in which the hexadecimal numbers represent a Bluetooth Address.

4.5.3 Final Hardware Setup

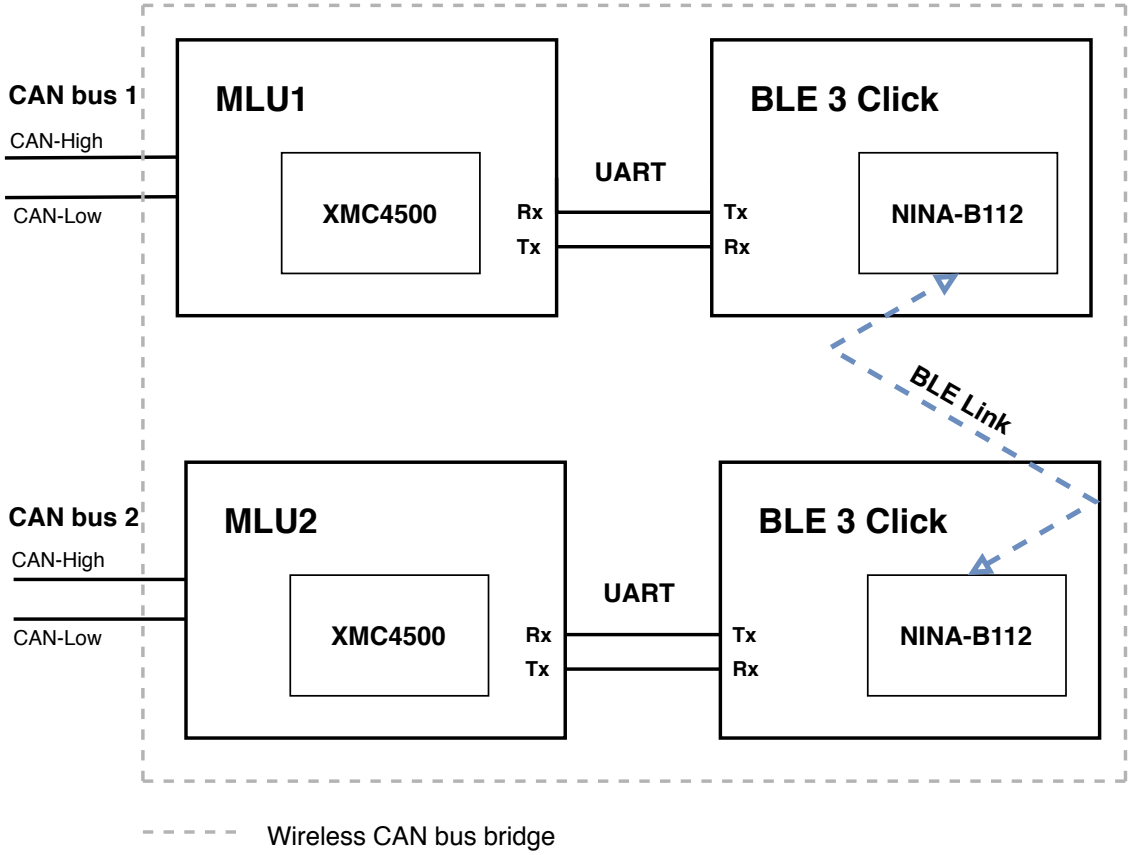

Figure 4.13: Hardware setup diagram of the final prototype.

4.6 Bluetooth Low Energy Connection

Serial Port Profile (SPP) emulates a serial port over air in Classic Bluetooth. There is no such profile for Bluetooth Low Energy and therefore no standardized manner of transmitting generic data over the air.

Serial Port Service (SPS) is a non-standard profile developed by u-blox. The specification is open and can be implemented in any BLE device to enable generic data transmission. SPS is implemented on top of the Generic Attribute Profile (GATT). It contains the following characteristics:

1. FIFO for reading and writing data.
2. Credits to simulate the Classic Bluetooth credit-based flow control.

In this implementation we will only be using the FIFO characteristic because connections without credits are faster to establish.

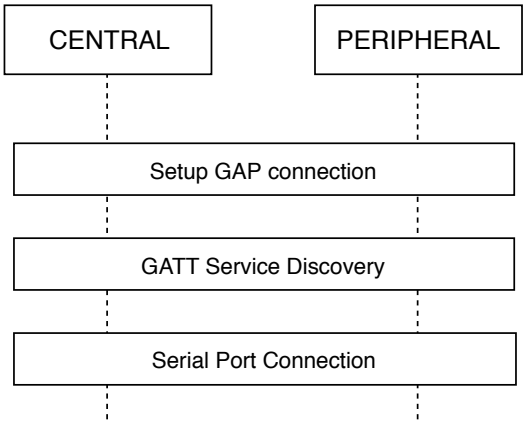

Figure 4.14: Overview on a SPS connection setup.

To connect, the Central/Client establishes an Asynchronous Connection-Less (ACL) connection and provides Indications for the FIFO characteristic. Once data transmission may start, the Client writes to the FIFO and the Server sends Indications with the FIFO data. We need to take into account that the receiver has no mean to stop the data flow meaning that data may get lost, if it cannot handle all the received data.

4.6.1 GAP Connection Setup

In this implementation both devices are configured by us so in this case the peripheral knows to what central it wants to establish a connection with. As a result, the peripheral makes a directed advertisement and when the central detects the advertisement, directed to itself, it must set up the connection.

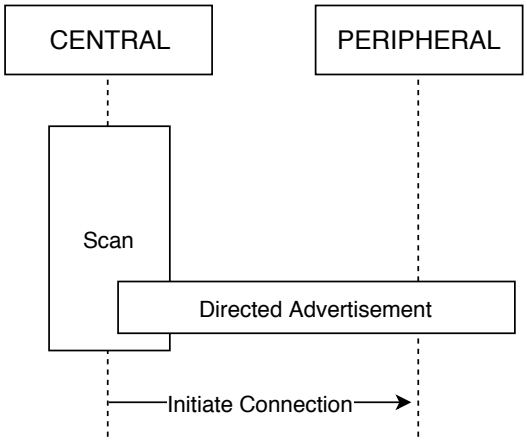

Figure 4.15: GAP connection setup.

It is important to note that the shorter the connection interval is, the shorter the connection time between the two devices will be.

4.6.2 GATT Service Discovery

Before a central makes a connection setup, it must find all relevant attribute handles. If they are not already cached, it is necessary for the GATT client to be able to discover the available services.

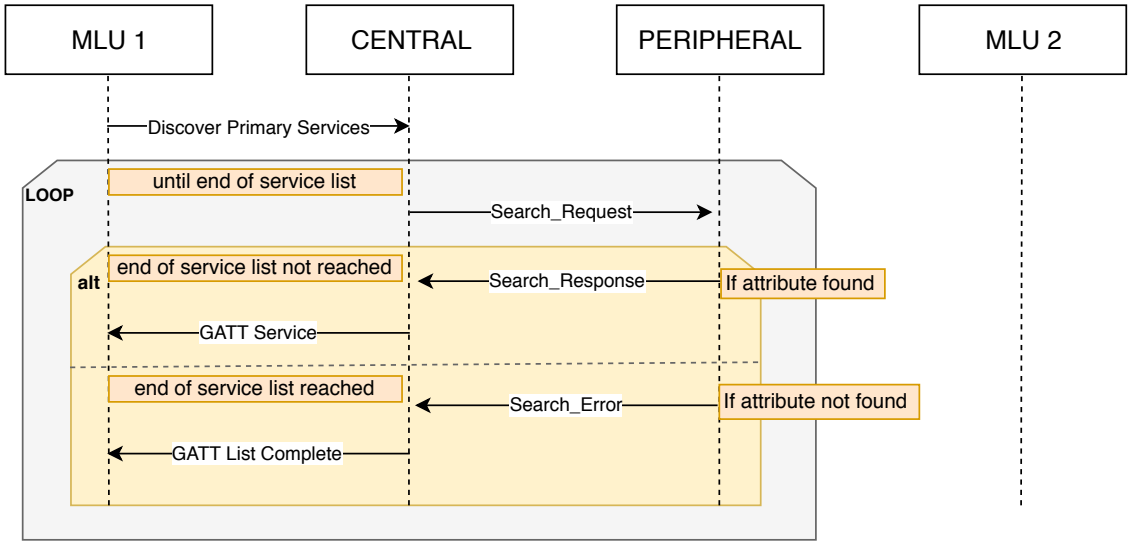

Figure 4.16: Process of discovering GATT services.

4.6.3 Serial Port Connection

The peripheral contains a Serial Port Service with the FIFO characteristic. The FIFO characteristic is used for writing data from the central to the peripheral and to Indicate data from the peripheral to the central.

In our implementation, cached attribute handles and client configuration will be used, therefore, application data can be transferred immediately after the ACL link has been established. As a result, there is no need of doing any service search or writing client configuration characteristics. With this behaviour the devices are said to be bonded.

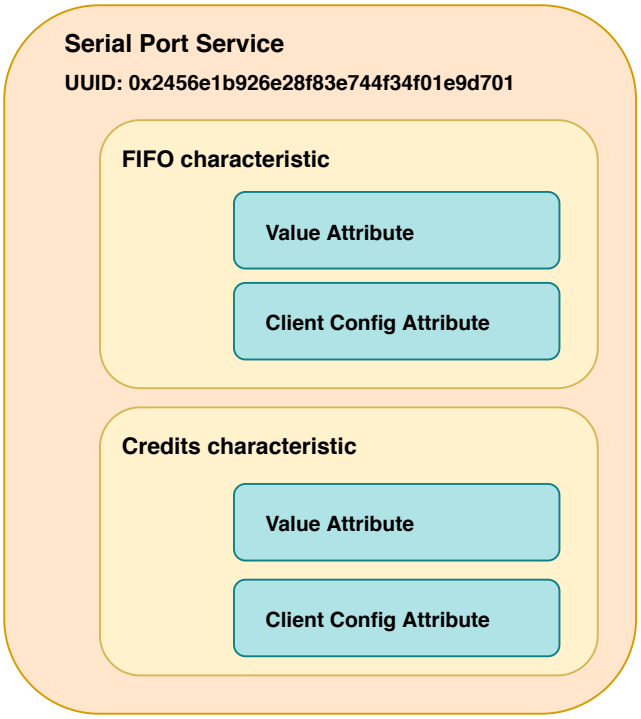

Figure 4.17: Characteristics of Serial Port Service.

Note that we won't be using the Credits characteristic, as mentioned before.

4.6.3.1 Serial Port FIFO characteristic

Value Attribute:

-UUID: 0x2456e1b926e28f83e744f34f01e9d703;

-Type: uint8 array (244 bytes = min MTU - opcode_size - handle_size);

-Properties: Indication/WriteRequest;

The FIFO characteristic represents the data to be sent to the UART. All data that is received on this characteristic should be placed in an internal FIFO or directly put on the external interface. When data is received on the external interface, the data should be put in the internal FIFO.

Client Configuration Attribute:

-Properties: WriteRequest;

The client configuration must be enabled for indications before receiving the data. The client configuration should be stored for bonded devices.

4.6.3.2 Connection Sequence Diagram

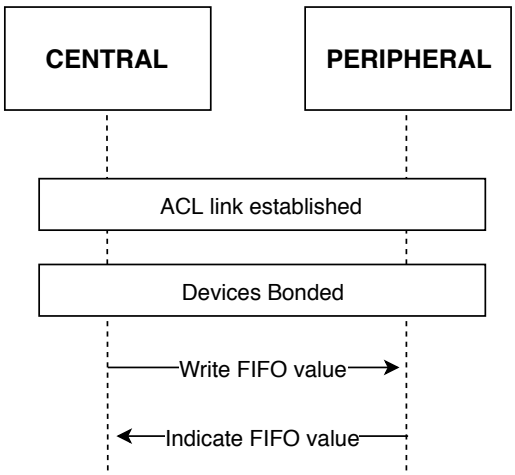

Figure 4.18: Connection without credits (bonded).

The entire ACL link must be teared down to disconnect the SPS link.

4.7 Gateway Firmware

In this section we describe the general structure of the software implemented.

4.7.1 Overview

As both Bluetooth Modules will be configured to start in Data Mode (figure 4.10) anything transmitted on the UART, will be transmitted over air to the connected remote device. The data received over air from the remote device is transmitted on the UART.

The figure 4.19 is an overview of the program implemented on one MLU. It is a representation of the two data flows of the implemented protocol:

1. The one that receives a CAN message from a CAN network and forwards that same message through the serial port to the Bluetooth Module.
2. The one that receives a CAN message (one byte at a time) from the serial port and forwards that same message to the CAN network to which that MLU is connected to.

Note that after a CAN message is received in one end, it has to go through a process so that it can be properly sent to the other end.

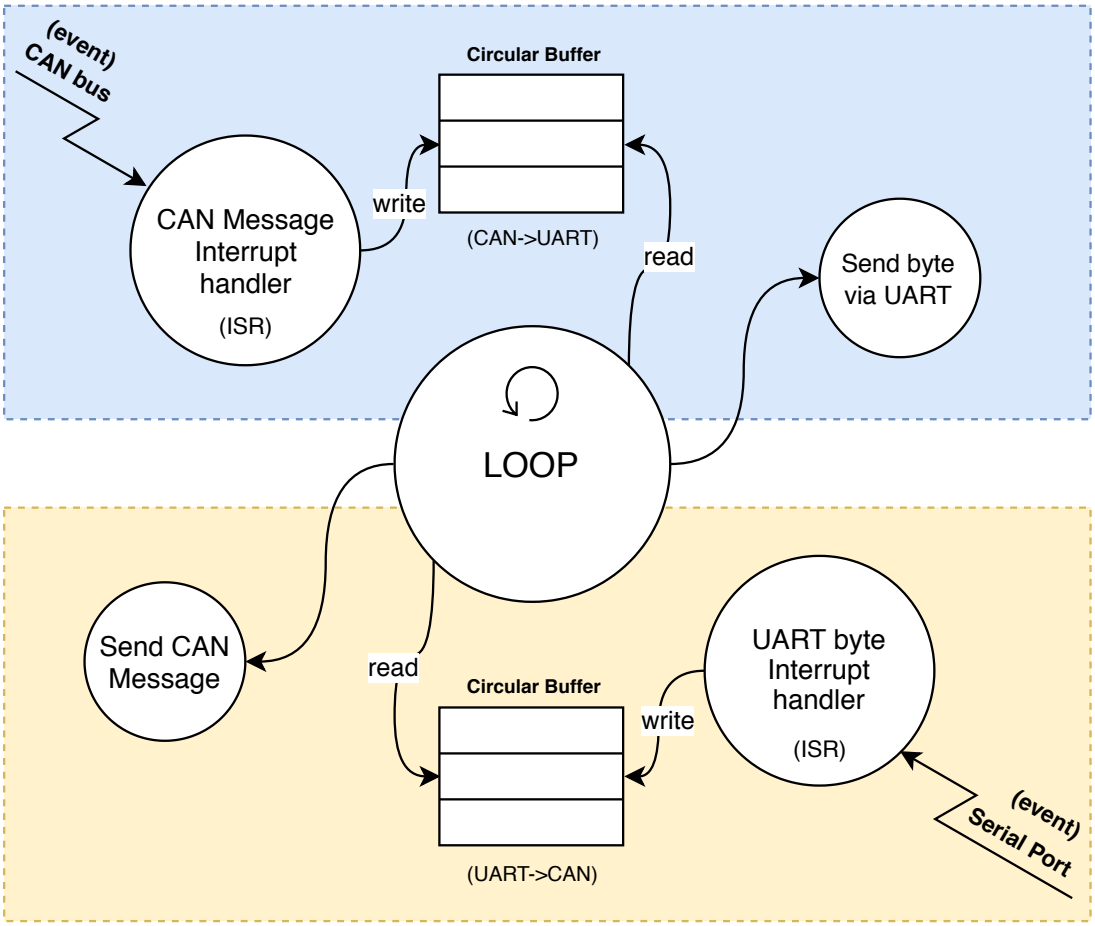

Figure 4.19: General diagram of the software implemented on one MLU.

Both MLU's of the final prototype will behave as described in figure 4.19 once the BLE connection is ultimately set up and running.

4.7.2 Interrupt Driven Serial Communication

As we receive bytes, to avoid constantly polling for data in the main loop we will handle the UART with an Interrupt Service Routine (ISR). Whenever a character is received, the UART will fire an interrupt. This will cause our main program to temporarily jump to an interrupt handler which reads the received byte and stores it in a circular buffer. In the main loop, we constantly check the circular buffer to see if it contains any characters and process them.

This a perfect application for a circular buffer. We can make a circular buffer interrupt-safe as long as we insert from the interrupt handler and remove from the main loop.

Circular buffers are very good for serial data streams in embedded systems. Bytes that just arrived need to be stored in order and dealt with later (bytes may come in at a faster rate than they can be handled). The benefit of a circular buffer is that we don't need to allocate infinite amounts of memory, since older entries get overwritten.

The buffer effectively splits the timing-critical response required (when the bytes come in, in microseconds) to the non timing-critical response to the whole message:

1. Upon the receipt of a byte the UART generates an interrupt to which the software responds by quickly taking the received byte and write it onto the circular buffer.
2. Our main routine will regularly check if the buffer has anything in it yet and empty it as required.

The application will need to be able to accept all the RX data coming to it at whatever data rate that is being used. Over the long term if data comes faster than it can be processed no amount of circular queue buffering will help to solve the problem. The size of the circular queue needs to be large enough to hold all the data stuffed into it by the RX interrupt whilst the processor is doing other time consuming events.

As the circular buffer size is going to be defined pre-compilation the size is then limited. This helps improve space efficiency and should eliminate memory corruption at a trade off to how many bytes can be received before older data starts to become lost.

4.7.3 Can Message in Binary Format

CAN Messages will be coded in binary format, instead of the usual approach that is the ASCII format because throughput is the main concern in this implementation.

Basic features of binary format:

- Allows faster transmission of CAN messages.
- Data of the CAN message is transmitted uncoded in a binary value.
- Fields are not separated by blanks.
- Fields are without Carriage Return (CR) or Line Feed (LF) characters.

This message format will be used to transmit CAN messages over the Bluetooth Low Energy connection to the other device.

Standard CAN Message

SB	TDL	STD_ID_H	STD_ID_L	D0	D1	D2	D3	D4	D5	D6	D7

Extended CAN Message

SB	TDL	EXT_ID_HH	EXT_ID_HL	EXT_ID_LH	EXT_ID_LL	D0	D1	D2	D3	D4	D5	D6	D7

Here follows a brief descriptions of what means each individual byte and the order in which the message will be written on both circular buffers.

Parameter	Description
SB	CAN Message Start Byte
TDL	Type and Data Length bit 4: 0 is Standard, 1 is Extended bit 0 to 3: Data length 0–8
STD_ID_H	High byte of Standard CAN ID (0–7)
STD_ID_L	Low byte of Standard CAN ID (0–FF)
EXT_ID_HH	High word, high byte of Extended CAN ID (0–1F)
EXT_ID_HL	High word, low byte of Extended CAN ID (0–FF)
EXT_ID_LH	Low word, high byte of Extended CAN ID (0–FF)
EXT_ID_LL	Low word, low byte of Extended CAN ID (0–FF)
D0 to D7	Up to 8 data bytes (0–FF)

Table 4.2: The parameters that define a CAN message in binary format.

Example of Shortest Message

```
0x94  0x00  0x07  0xFF
```

0x94	Start Byte
0x00	bit 4 = 0 (Standard Frame) and Data Length = 0
0x7FF	CAN Identifier

Example of Longest Message

```
0x94  0x18  0x1F  0xFF  0xFF  0xFF  0x01  0x02  0x03  0x04  0x05  0x06  0x07  0x08
```

0x94	Start Byte
0x18	bit 4 = 1 (Extended Frame) and Data Length = 8
0x1FFFFFFF	CAN Identifier
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08	8 data bytes

Having a specific pre-determined byte before the actual message will allow us to know when a new message has arrived and from there we know which kind of message we will receive and its total length from that point. 0x94 was chosen because it does not represent an ASCII character.

As we can, in this format, the maximum size of a CAN message can be represented in 14 bytes and the shortest in only 4 bytes.

4.8 Remarks

This chapter presented a detailed description of this project, as well as the steps to follow for its design, development and validation. A number of minimum requirements were defined that the implemented final prototype must comply to. We also described the available hardware and the firmware to be developed using the listed developments tools in section 4.3. One relevant aspect presented in this chapter is the selection of the BLE 3 Click Board that contains the NINA-B112 Bluetooth Low Energy module from u-blox. The wireless bridge will use the Serial Port Service, provided by u-blox, as a method for cable replacement.

Chapter 5

Implementation

In this chapter the entire implementation process of the structures explained in last chapter will be explained. We will describe which DAVE APPs will be utilized and why. C code segments and a brief explanation of those segments will also be presented.

5.1 Circular Buffer

The type of FIFO we will be implementing is called a circular buffer, also known as a ring buffer. It is called a circular buffer because data can wrap around back to the beginning. Really it is just implemented as an array but the beginning of the queue does not have to start at the first element of the array, and the end does not necessarily end at the last element in the array. The start of the queue could begin somewhere in the middle of the array, wrap around the last element back to the beginning and end there. The start of the queue is where new data will be written to. The end of the queue contains the oldest data and is where the caller will read from. These are commonly referred to the head and tail respectively. Note these are just naming conventions for the sake of theory, their exact meaning is implementation dependent.

We need four things: an array, a write pointer (head), a read pointer (tail) and the maximum capacity. These will all be accessed by both the main loop and the interrupt handler, so they should all be declared as volatile. Also, updates to the head and updates to the tail each need to be an atomic operation, so they should be the native size of our architecture.

```
#define BUFFER_MAX 256

typedef struct circular_buffer
{
   volatile uint8_t data[BUFFER_MAX];
   volatile uint32_t write;
   volatile uint32_t read;
   volatile uint32_t N_MAX;
}circular_buffer_t;
```

Listing 5.1: Data structure for our circular buffer.

This implementation uses two indices that grow unbounded, and eventually wrap around to zero once the unsigned integer overflows. This reclaims the wasted slot issue that the more common implementations of this type of structure face. The code modifying the indices also becomes simpler.

```
void buffer_init(volatile circular_buffer_t* cb){

   cb->N_MAX = BUFFER_MAX;
   cb->read=0;
   cb->write=0;

   for(uint16_t i=0; i < cb->N_MAX; i++)
      cb->data[i] = 0;
}

uint32_t buffer_mask(volatile circular_buffer_t* cb, uint32_t val){

   return val & (cb->N_MAX - 1);
}

bool buffer_full(volatile circular_buffer_t* cb){

   return buffer_size(cb) == cb->N_MAX;
}

bool buffer_empty(volatile circular_buffer_t* cb){

   return cb->read == cb->write;
}

void buffer_write(volatile circular_buffer_t* cb, uint8_t newByte){

   cb->data[buffer_mask(cb, cb->write++)] = newByte;

   if( buffer_full(cb) )
      buffer_read(cb);
}

uint8_t buffer_read(volatile circular_buffer_t* cb){

   if( !buffer_empty(cb) )
      return cb->data[buffer_mask(cb, cb->read++)];
}
```

Listing 5.2: Functions for our circular buffer.

We have implemented functions to check the status of the buffer to know if it is empty or full. It is empty if the write pointer is equal to the read pointer and it is full if the size equals the buffer maximum capacity. We also implemented a function to initialize the buffer (array and pointers).

Both indices will always be in the range 0 to (capacity - 1). This is done by masking the value after an index gets incremented. Note that if the buffer is full we read one value and then there will be space to write the new value. As there will be only one producer and only one consumer, there is no need to worry about race conditions as no other threads can write.

This implementation works without errors, assuming the following restrictions:

- The capacity must always be a power of two. It is required for the code to be correct on the unsigned integer overflow. The capacity of our buffer will be 256, which is a power of two.
- The implementation language supports wraparound on unsigned integer overflow. C does so there is no problem here.
- The maximum capacity can only be half the range of the index data types. Again, this is no problem here because the range of our indices is way larger than the maximum capacity. We use 2^8 for the capacity and 2^{32} for the indices.

In general, this will be the behaviour of the circular buffer:

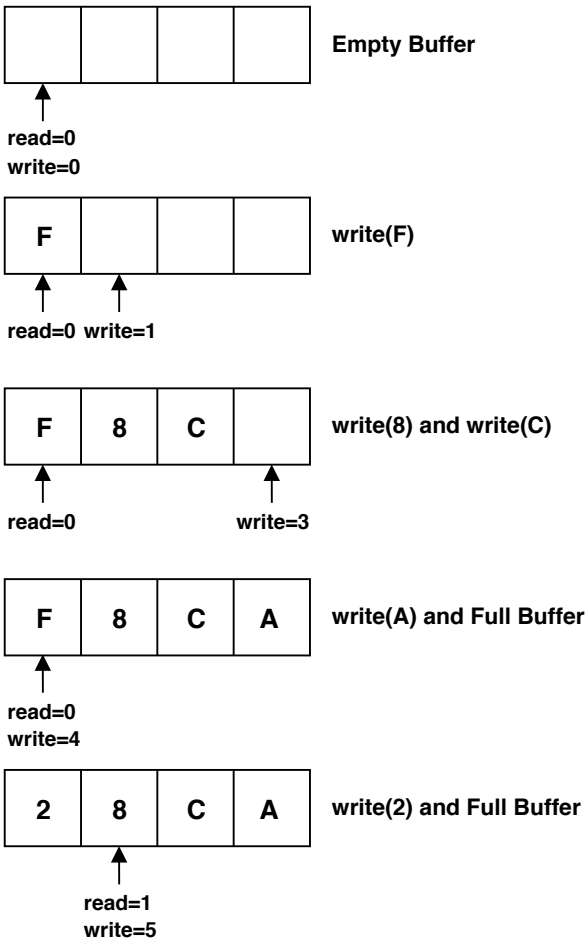

Figure 5.1: Diagram of a circular buffer of 4 elements.

5.2 Setup of a CAN Node

Our XMC device provides a MultiCAN module with up to 3 CAN nodes and a Buffer of 64 Message Objects (MO). To be able to receive and transmit CAN messages we first have to configure a CAN Node with DAVE's CAN_NODE APP.

The CAN_NODE APP provides the below features:

- Configures one node instance of the MultiCAN module.
- Each CAN node can be configured to connect to the external CAN bus. CAN node allows to configure pins for both transmission and reception.
- Supports data transfer rate from 100kbit/s to 1Mbit/s.
- Configures up to a maximum of 32 logical Message Objects.

The APP also provides signals for Message Object events (successful reception and transmission of CAN frames). To enter our interrupt handler functions on each event, the corresponding event has to be enabled in CAN_NODE APP and that event signal has to be connected to an instance of the INTERRUPT APP.

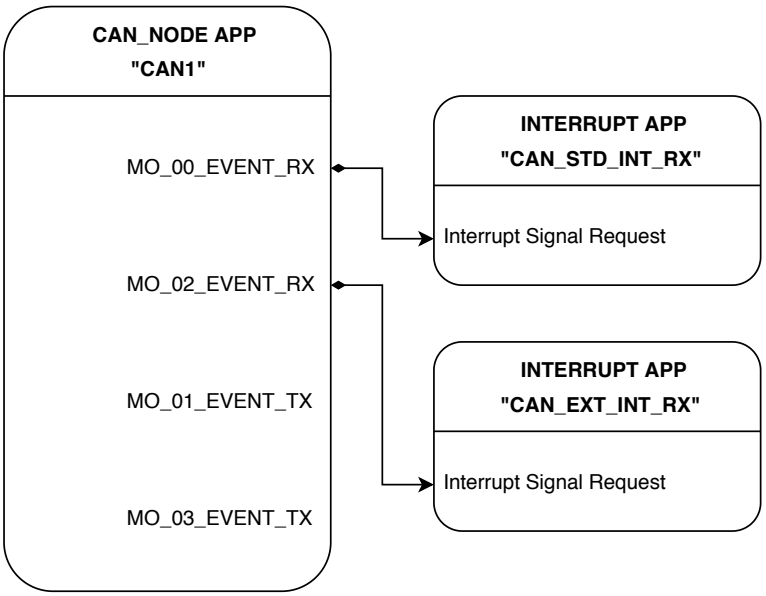

Figure 5.2: CAN NODE APP signal connectivity with INTERRUPT APP.

A Message Object serves as a storage container (frame buffer) for incoming and outgoing frames. It can be of the Transmit type or of the Receive type. Both message objects can be configured for receiving and transmitting standard and extended frames and acceptance mask can be configured for each frames.

We will configure and initialize an instance of a CAN NODE APP and in it we will configure four Message Objects for reception and transmission of CAN frames. More specifically, one to

receive a standard frame, one to receive an extended frame, one to transmit a standard frame and one to transmit an extended frame. The receiving ones are the ones that follow:

```
const CAN_NODE_t *CAN1Handle = &CAN1;
XMC_CAN_MO_t *MO_Ptr_std;
XMC_CAN_MO_t *MO_Ptr_ext;

buffer_init(&B_U_C);          //Circular Buffer UART_TO_CAN
buffer_init(&B_C_U);          //Circular Buffer CAN_TO_UART
CAN_config_LMO(CAN1Handle, 0x00000000, 8, 0, 0, 0x00000000, 11);
CAN_config_LMO(CAN1Handle, 0x00000000, 8, 0, 2, 0x00000000, 29);
```

The CAN1Handle is a pointer to a CAN_NODE instance, named CAN1. The baud rate was set to 1Mbit/s on the APP UI. The MO_Ptr allows access to CAN Message Object runtime elements, such as the ID and Data of a received message. The CAN_config_LMO function will basically configure two Receive Message Objects to accept all possible Identifiers regarding the CAN protocol. Message Object number 0 will accept all 11 bit IDs and Message Object number 2 will accept all 29 bit IDs. We also initialize both circular buffers so we can write on them.

5.2.1 Receiving Standard CAN Frames

As seen in figure 5.2, we named our interruption handler function, for the reception of standard CAN frames, "CAN_STD_INT_RX".

```
void CAN_STD_INT_RX(void){

   status_std = 0;
   MO_Ptr_std = CAN1Handle->lmobj_ptr[0]->mo_ptr;

   status_std = CAN_NODE_MO_GetStatus(CAN1Handle->lmobj_ptr[0]);
   if ( status_std & XMC_CAN_MO_STATUS_RX_PENDING ){
      CAN_NODE_MO_ClearStatus(CAN1Handle->lmobj_ptr[0],
    XMC_CAN_MO_RESET_STATUS_RX_PENDING);

      CAN_NODE_MO_Receive(CAN1Handle->lmobj_ptr[0]);
   }

//BEGIN_PROTOCOL_STD
   buffer_write(&B_C_U, 0x94);
   buffer_write(&B_C_U, MO_Ptr_std->can_data_length);
   buffer_write(&B_C_U, MO_Ptr_std->can_identifier >> 8);
   buffer_write(&B_C_U, MO_Ptr_std->can_identifier);
   for(int i=0 ; i<MO_Ptr_std->can_data_length ; i++)
      buffer_write(&B_C_U, MO_Ptr_std->can_data_byte[i]);
//END_PROTOCOL_STD
}
```

Listing 5.3: Interrupt handler for reception of CAN Standard Frames.

We start by checking the receive pending status of Message Object 0 (that was configured for the reception of all 11 bit IDs). If the reception of a CAN message is pending, we clear the flag status and we read the received Message Object and store the received data in the MO structure. After that, we are able to get the information we want and therefore write the ID and the Data to our CAN->UART circular buffer.

As explained in 4.7.3 we start by writing the startbyte. Next we just need to write the message length as is because bit 4 of this byte is already 0. Note that buffer_write() casts all values to an uint8_t. After that we perform a right bit shift so we can get the high byte of the two byte ID and also write the low byte to the buffer. The only thing missing now is the respective data bytes of the CAN frame. Inside the for statement we will write the number of data bytes that are available in that specific message. We now have a complete CAN message in our circular buffer, ready to be read by the main loop and sent to the Bluetooth link.

5.2.2 Receiving Extended CAN Frames

The same process applies to the reception of Extended CAN frames, only with slightly differences. The Message Object number is now 2 and the process of writing to the circular buffer differs a little bit. As seen in figure 5.2, we named our interruption handler function, for the reception of extended CAN frames, "CAN_EXT_INT_RX".

```
void CAN_EXT_INT_RX(void){

   status_ext = 0;
   MO_Ptr_ext = CAN1Handle->lmobj_ptr[2]->mo_ptr;

   status_ext = CAN_NODE_MO_GetStatus(CAN1Handle->lmobj_ptr[2]);
   if ( status_ext & XMC_CAN_MO_STATUS_RX_PENDING){
      CAN_NODE_MO_ClearStatus(CAN1Handle->lmobj_ptr[2],
    XMC_CAN_MO_RESET_STATUS_RX_PENDING);

      CAN_NODE_MO_Receive(CAN1Handle->lmobj_ptr[2]);
   }

//BEGIN_PROTOCOL_EXT
   buffer_write(&B_C_U, 0x94);
   buffer_write(&B_C_U, MO_Ptr_ext->can_data_length+16);
   buffer_write(&B_C_U, MO_Ptr_ext->can_identifier >> 24);
   buffer_write(&B_C_U, MO_Ptr_ext->can_identifier >> 16);
   buffer_write(&B_C_U, MO_Ptr_ext->can_identifier >> 8);
   buffer_write(&B_C_U, MO_Ptr_ext->can_identifier);
   for(int i=0 ; i<MO_Ptr_ext->can_data_length ; i++)
      buffer_write(&B_C_U, MO_Ptr_ext->can_data_byte[i]);
//END_PROTOCOL_EXT
}
```

Listing 5.4: Interrupt handler for reception of CAN Extended Frames.

We now have an Identifier of 29 bits. That means that in the message length byte (TDL) bit 4 needs to be 1 instead of 0. To achieve that we can simply add 16 in decimal or 0x10 in hexadecimal. We will store the extended ID in four different bytes, so we follow the same process as the standard method described above. We right shift 24 times to get the most significant byte, then 16 times to get the next after that, then 8 times to get the next after that and then we just need to write the least significant byte. To the data bytes the process is exactly the same as for the standard one. We now have a complete extended CAN message in our CAN->UART circular buffer.

5.3 Setup the UART

Universal Asynchronous Receiver/Transmitter is used for asynchronous serial communication. UART APP configures the Universal Serial Interface Channel (USIC) peripheral of the XMC microcontroller, to work in asynchronous serial communication (ASC) mode. The APP UI allows to configure baudrate, data length, number of stop bits and parity setting. The 8-N-1 configuration will be used without flow control and the baudrate will be set to 1Mbit/s. We can choose to get notified on completion of data reception using a callback function. The callback function name can be provided in the APP UI. To enter our interrupt handler functions on a reception event, that event signal has to be connected to an instance of the INTERRUPT APP.

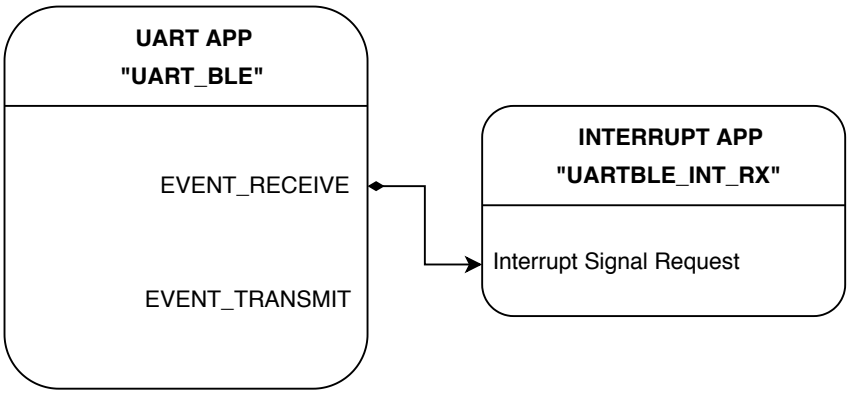

Figure 5.3: UART APP signal connectivity with INTERRUPT APP.

5.3.1 Receiving a byte from the serial port

Although we could use the UART APP APIs for the reception of a byte we opted to go for the more low level approach. This simplifies the code and the signal connectivities needed for this particular function of the code.

```
void UARTBLE_INT_RX (void){

   buffer_write(&B_U_C, USIC2_CH1->RBUF);
}
```

Listing 5.5: Interrupt handler for reception of an UART byte.

To write the received byte to our buffer we only need to access the Receive Buffer Register (RBUF) of our UART channel configured before. Here we are writing to the UART->CAN buffer.

5.4 Main Loop

After initialization of all the peripherals the program executes the main cycle.

```
int main(void){

   while(1U){

      //read byte from CAN_TO_UART_buffer and transmit to BT_module
      send_byte_BT();

      //read from UART_TO_CAN_buffer, build CAN message and transmit to CAN bus
      send_CAN_msg();
   }
}
```

Listing 5.6: Main loop of our program.

Here we simply call the two functions that will be responsible for reading from both buffers and send the respective messages to the corresponding network.

5.4.1 Send byte via UART

As our BLE modules will be configured to start in Data Mode everything that is sent on the serial port is going to be forward to the remote device serial port via Bluetooth Low Energy.

As this function is called in the main loop, it will always be checking if the buffer is not empty, meaning that there is something to read, and will also be checking if the TDV bit of the Transmit Status Register (TCSR) is cleared (it clears once a serial byte has finished being transmitted).

This way we know that the UART is not busy meaning we are cleared to send the next byte. To send the byte we just need to read the byte from the CAN->UART buffer and write it to the Transmit Buffer (TBUF).

```
void send_byte_BT(void){

    if( !buffer_empty(&B_C_U) && !(USIC2_CH1->TCSR & USIC_CH_TCSR_TDV_Msk) )
        USIC2_CH1->TBUF = buffer_read(&B_C_U);
}
```

Listing 5.7: Transmission of a byte to the serial port.

5.4.2 Send CAN Message to the CAN bus

On this section relies the most sensitive part of the program. As our UART->CAN buffer is filled with new CAN messages we need to make sure we carefully read from the buffer and build have a complete CAN message before sending it to the CAN bus. Due to the fact that we have a specific startbyte and know the length of the complete message, this becomes easier. Because the code is too long it is going to be shown in Appendix A. Therefore we opted for a flowchart to help better understand this whole process.

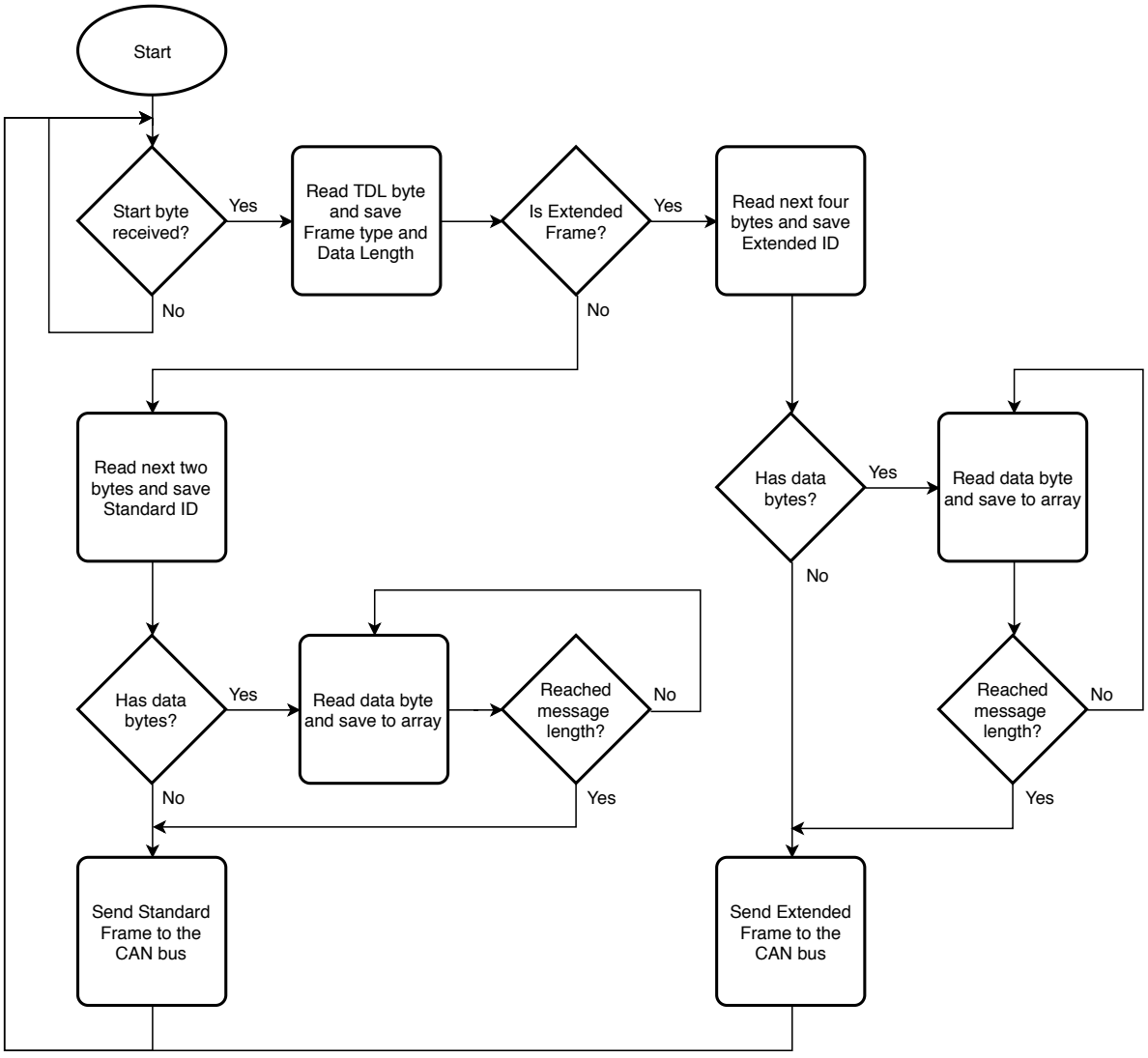

Figure 5.4: Flowchart of the process of sending a CAN frame to the CAN bus.

5.5 Bluetooth Low Energy Module Configuration

For the correct functioning of the entire system we need to configure our two NINA-B1's so the connection is properly established before the reception of any CAN messages. As said before, we

will use ADDAM (the process is explained in section 4.3.3) to send AT commands to our module and also to receive the responses.

First we need to apply the desired UART configuration for both modules. So we need to send "AT+UMRS=1000000,2,8,1,1", meaning 1Mbit/s for the baudrate, no flow control, 8 data bits, no parity and 1 stop bit.

The Central is the one responsible for discovering the Peripheral, so we have to configure one as the Central "AT+UBTLE=1" and other as the Peripheral "AT+UBTLE=2". Later on we will need the Peripheral's Bluetooth Address. In order to get it we use the command "AT+UMLA=1" in the Peripheral and it responds with the local Bluetooth Address.

Security Mode is going to be configured to the Just Works mode, guaranteeing that the data is going to be encrypted and for that we need to send "AT+UBTSM=2" for both modules.

After that there are a few parameters that need to be set regarding the BLE configuration. Note that all of these help us achieve the maximum throughput possible with Bluetooth Low Energy using Serial Port Service:

- **Connection Interval:** The connection interval must be lowered to the minimum value possible, which is 6x1,25ms=7,5ms, on both modules. In order to achieve this we must send "AT+UBTLECFG=4,6" to set the minimum interval to 7,5ms and "AT+UBTLECFG=5,6" to set the maximum interval to that same value.
- **ATT MTU size negotiation:** For our modules to request and accept incoming requests to negotiate the maximum size ATT MTU size (247 bytes) we need to send "AT+UBTLECFG=26,1".
- **Preferred Receiver/Transmitter PHY:** To set the physical layer to 2Mbit/s instead of 1Mbit/s we need to send "AT+UBTLECFG=27,2" for the Transmitter and "AT+UBTLECFG=28,2" for the Receiver.

As said before, the Central needs to know the Bluetooth Address of the Peripheral to whom he wants to establish a direct connection to. For that we need to send "AT+UDDRP=0,sps://*peripheral_address*,2" to the module that will be the Central. This will set the Peripheral as the only default peer using Serial Port Service and with always connected mode, that is, on errors and remote disconnect, the peer will automatically try to reconnect.

Finally, so that both modules start in Data Mode we need to send "AT+UMSM=1" and then to store all these parameters in non-volatile memory we need to send "AT&W0" and reboot the module by sending "AT+CPWROFF".

By following all the procedures described in this chapter we are able to successfully set up a Bluetooth Low Energy connection that will allow both MLUs to communicate and transmit CAN messages between them with some level of security.

5.6 Remarks

This chapter describes the whole process involved in the implementation of the firmware that will allow our MLUs to be able to receive CAN messages from the CAN bus and forward them to the other MLU via a Bluetooth Low Energy connection. We analyze the implementation of our circular buffers that allow possible bursts of data to be handled in a way that prevent possible data loss. Figure 5.4 explains how we are able read from a buffer (each byte at a time) and build a complete CAN frame (standard or extended) that can be sent to the CAN bus which the respective MLU is connected to. Finally we detail the AT commands used in order to achieve a fast and stable Bluetooth Low Energy connection and also the AT commands that will allow for the maximum throughput of the module.

Chapter 6

Validation

To test our Wireless CAN bus bridge, the communication protocols involved and the Bluetooth Low Energy connection, some tests were carried out. Selected performance metrics, namely Round-Trip Time (RTT), Gateway Delay (GWD) and Throughput are evaluated through experimentation. These metrics are essential to identify and better categorize the Wireless Gateway that we have just designed.

Note that the RTT and GWD tests are to be interpreted and understood from the MLU1(Central) perspective, meaning that all parameters were measured in that MLU only. MLU2(Peripheral) was configured to work as if it was performing in the final prototype, meaning that its program was not submitted to changes for these tests. Both MLUs are capable of forwarding CAN messages from both data flows (duplex) as they come and as fast as they possibly can. For RTT and GWD, both MLUs were connected to the same CAN bus and also via a BLE link. All tests were made in a safe environment and inside AddVolt's office. In order to present results close to the real-life use case, this work evaluates all performance metrics through measurement using real hardware that was provided by AddVolt and somewhat close to what is expected to be the final prototype.

Our system is capable of bi-directional communication (see figure 4.19) and so we need to test our gateway for both data flows. We will make these tests both for when only one data flow is active and when both data flows are active and see how it affects the behaviour of our system, especially the Bluetooth Low Energy latency and also the processing time. All the tests were made with the longest possible CAN message regarding our protocol, an Extended frame with 8 data bytes (14 bytes).

For the duplex communication tests we will "force" the transmission of two different CAN messages at the same point in time, one message via Bluetooth and one message via CAN. After that the system will continuously forward those two messages in a closed loop until 1000 messages are received in either data flow. For each test we will do this 5 times and calculate the average. This means that for each graph in this chapter, the total sample size is 5000 messages for one data flow, which a statistically significant sample size for testing the capabilities of the system.

For the simplex communication tests only one data flow will be active, therefore we will only "force" the transmission of a message in one direction.

Measures for CAN->BLE data flow

In the next figure we can see the timestamps measured on the CAN to Bluetooth direction regarding MLU1.

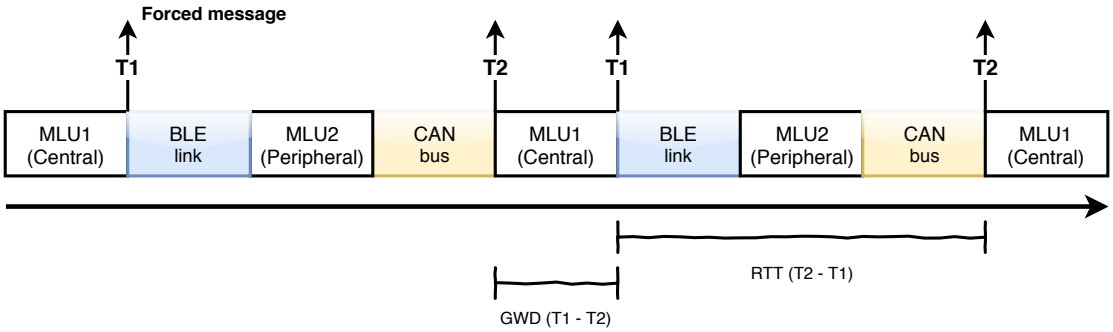

Figure 6.1: Round Trip Time (RTT) and Gateway Delay (GWD) measurements for CAN->UART buffer direction.

- **T1:** Timestamp 1 is taken before sending a CAN message in binary format via Bluetooth (right before the transmission of the last byte of the message to the serial port).
- **T2:** Timestamp 2 is taken after the reception of the message via CAN bus (right in the beginning of the CAN interrupt handler).

Measures for BLE->CAN data flow

In the next figure we can see the timestamps measured on the Bluetooth to CAN direction regarding MLU1.

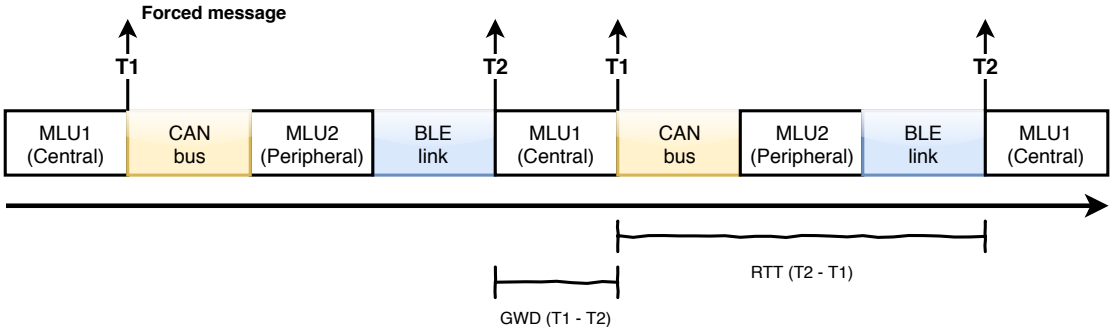

Figure 6.2: Round Trip Time (RTT) and Gateway Delay (GWD) measurements for UART->CAN buffer direction.

- **T1:** Timestamp 1 is taken right before sending a CAN message to the CAN bus (right before calling the CAN_transmit() function).
- **T2:** Timestamp 2 is taken right after the reception of a CAN message (beginning of the interruption handler).

6.1 Performance Metrics

6.1.1 Round Trip Time

Round-Trip Time (RTT) is the amount of time between the instant that the MLU1(Central) sends a CAN message and the instant that the MLU1(Central) receives the corresponding CAN message that was forwarded by the MLU2(Peripheral). Basically, the latency in each direction, plus the processing time (MLU1 to MLU2 back to MLU1).

These next experiments aim to evaluate the RTT from our gateway in different conditions and configurations. The factors from this experiment are connection interval parameters, the data flow type and the data flow direction. There are factors that could influence these tests but they are always the same for this experiment, such as PHY (2Mbit/s), ATT MTU (247 bytes) and Operation Type (see Figure 4.18) and the distance from the MLU1 to the MLU2 which for all the experiments in this chapter will be approximately 20 centimetres. The levels considered in this experiment are listed in the next table.

Connection Interval	Flow Type	Direction	Graph Number
min: 7,5ms max: 7,5ms	Simplex	CAN->BLE	Figure 6.3
		BLE->CAN	Figure 6.5
	Duplex	CAN->BLE	Figure 6.4
		BLE->CAN	Figure 6.6
min: 15ms max: 15ms	Simplex	CAN->BLE	Figure 6.7
		BLE->CAN	Figure 6.9
	Duplex	CAN->BLE	Figure 6.8
		BLE->CAN	Figure 6.10
BLE link replaced with an UART	Duplex	CAN->UART	Figure 6.11
		UART->CAN	Figure 6.12

Table 6.1: Collection of tests done for measuring RTT in different configurations.

The experiment described in this Section provides for 12 different type of sample (one for each combination of the factors). As said before each experiment is repeated 5 times, each one with 1000 messages for data flow.

The outcome from all RTT experiments may be affected by uncontrolled factors. The main factor is the environment signal noise. A high noise in the environment may cause BLE packet losses, increase packet delay as well as degrading the link capacity.

6.1.1.1 Results

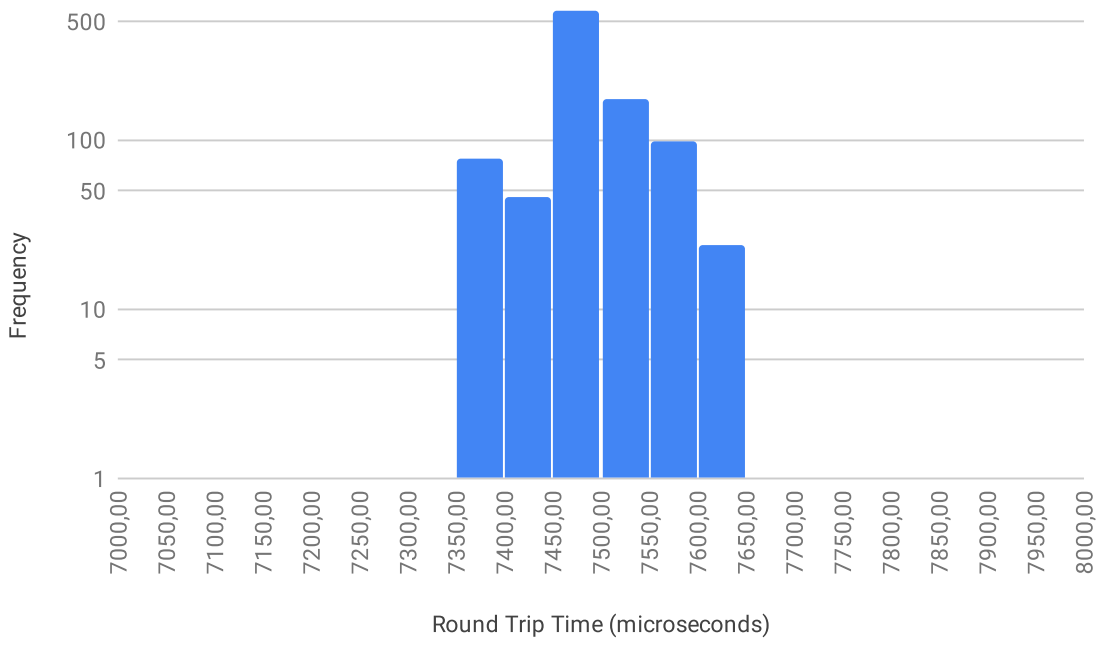

Figure 6.3: RTT for a conn. interval of 7.5ms, simplex communication in the CAN->BLE flow.

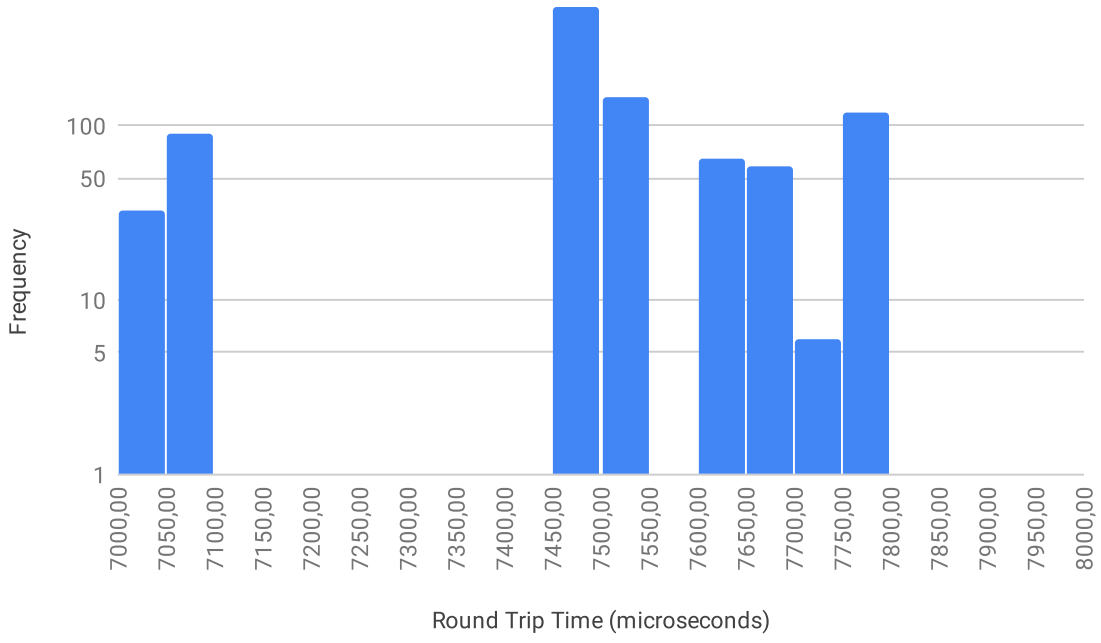

Figure 6.4: RTT for a conn. interval of 7.5ms, duplex communication in the CAN->BLE flow.

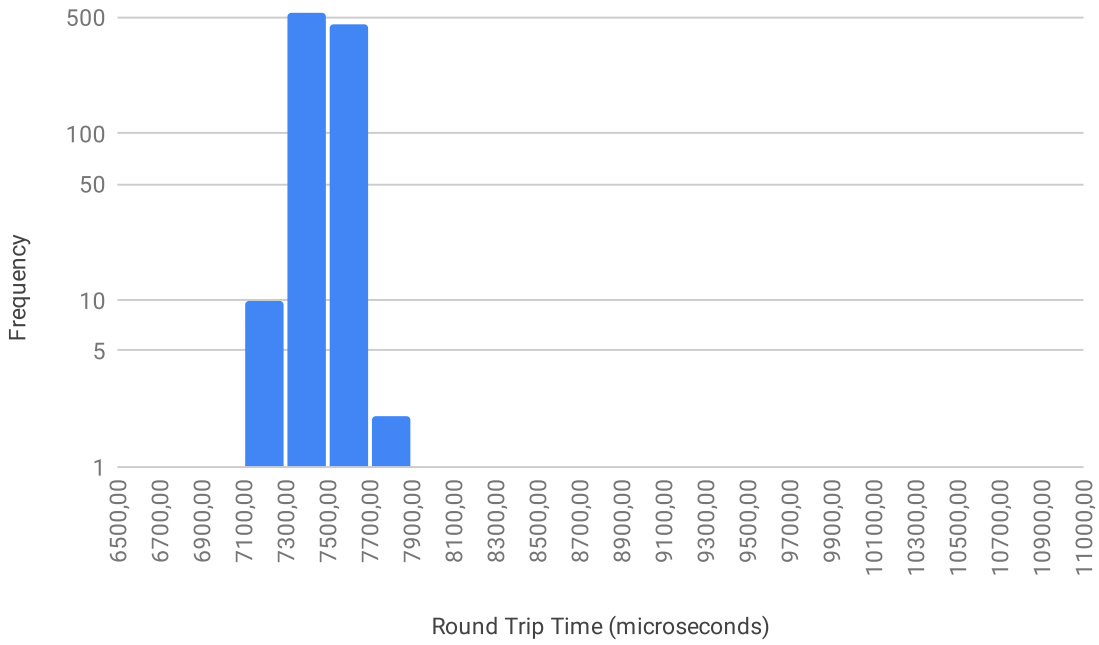

Figure 6.5: RTT for a conn. interval of 7.5ms, simplex communication in the BLE->CAN flow.

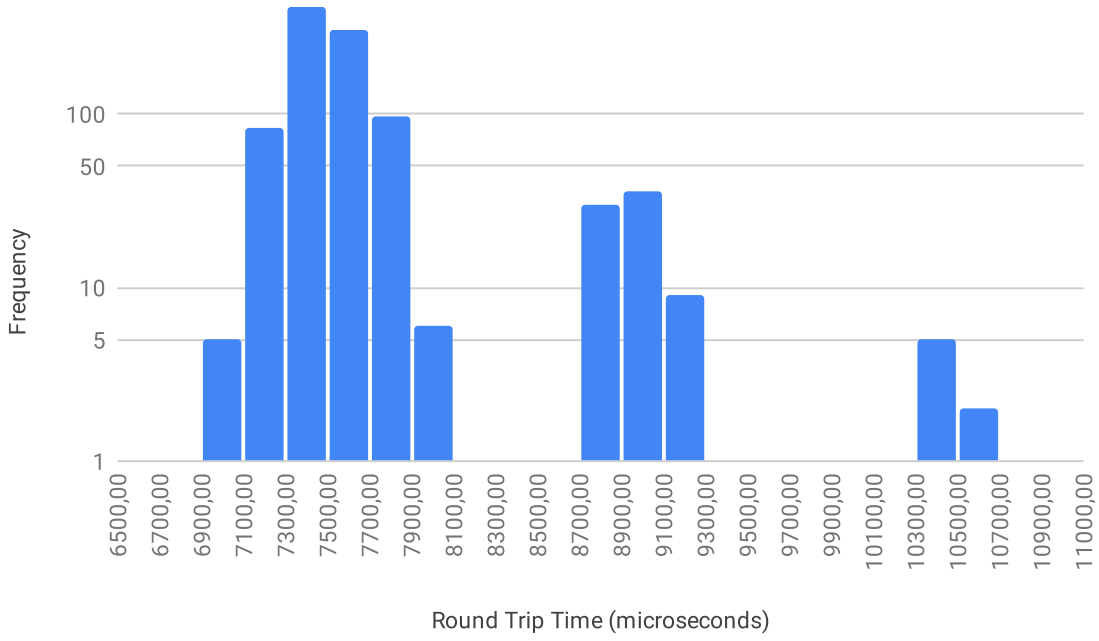

Figure 6.6: RTT for a conn. interval of 7.5ms, duplex communication in the BLE->CAN flow.

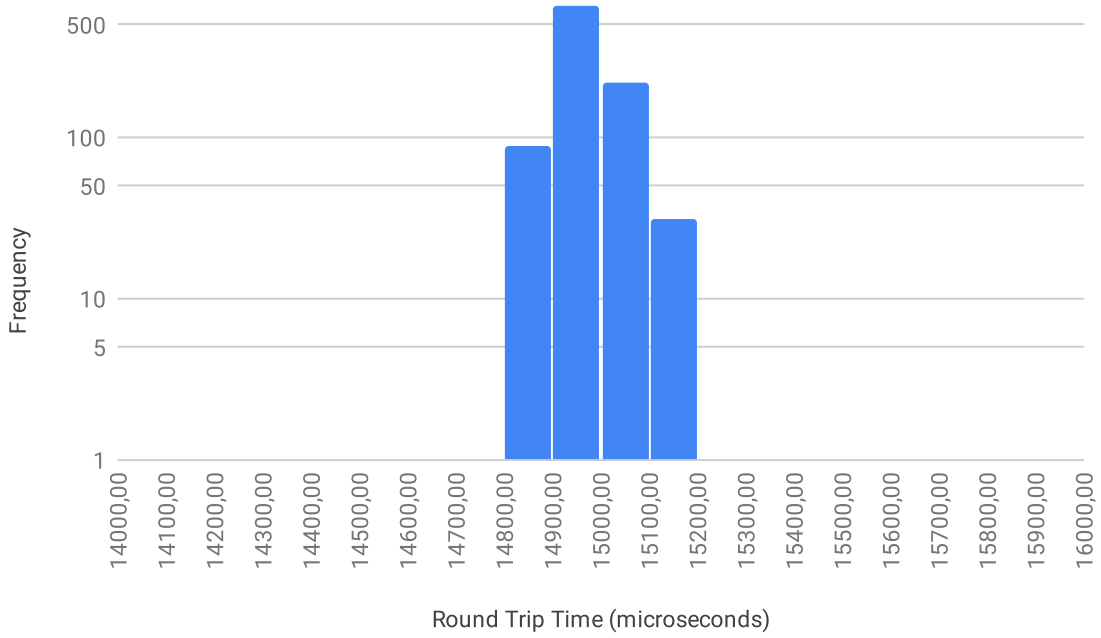

Figure 6.7: RTT for a conn. interval of 15ms, simplex communication in the CAN->BLE flow.

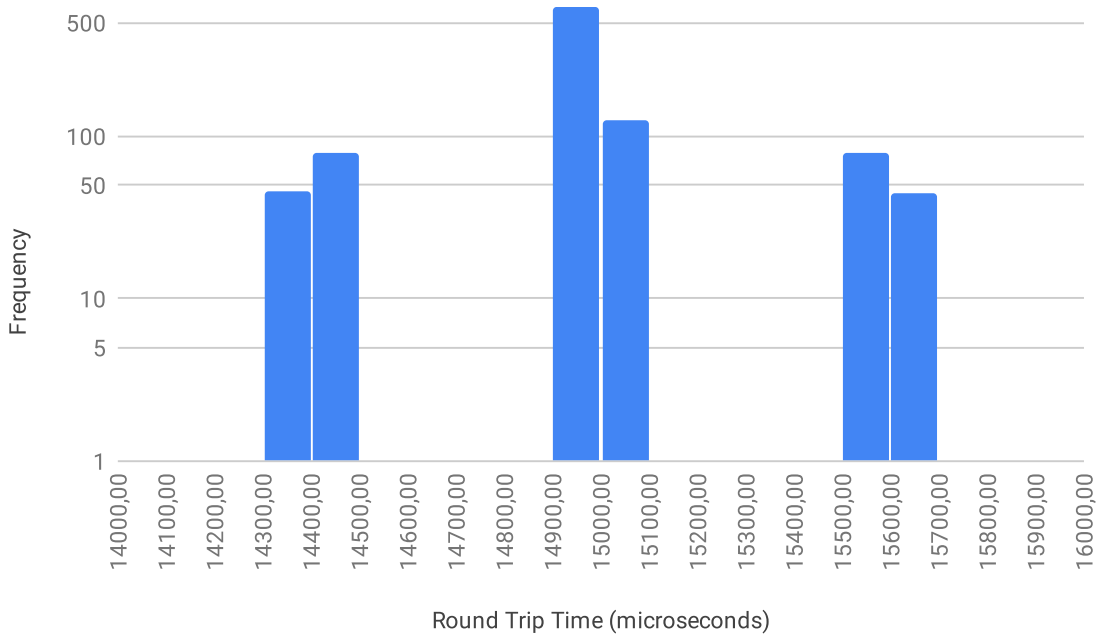

Figure 6.8: RTT for a conn. interval of 15ms, duplex communication in the CAN->BLE flow.

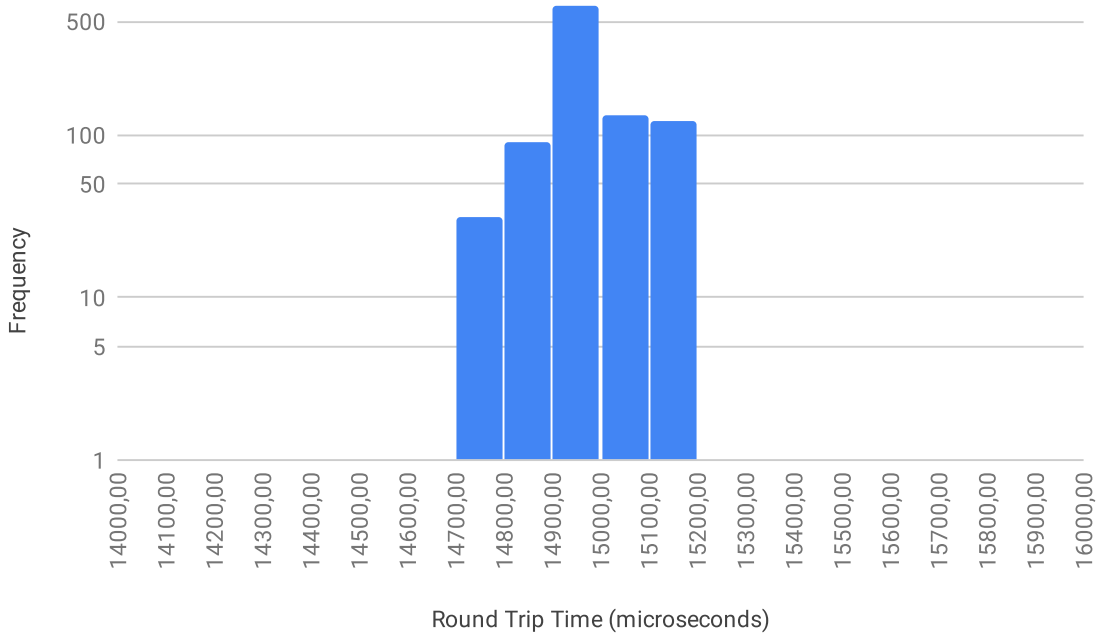

Figure 6.9: RTT for a conn. interval of 15ms, simplex communication in the BLE->CAN flow.

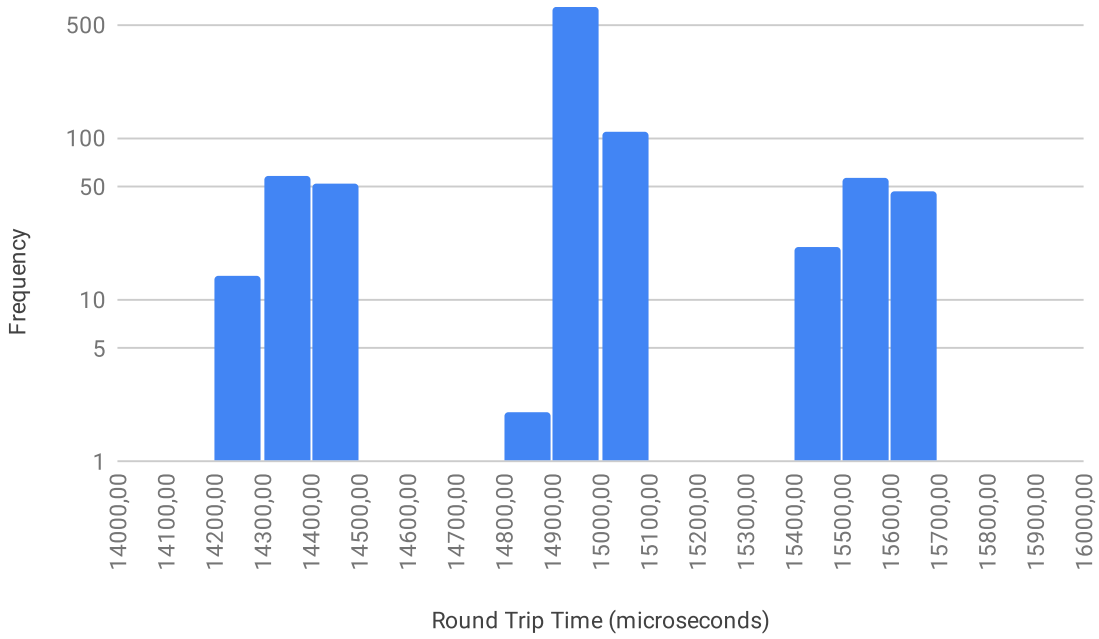

Figure 6.10: RTT for a conn. interval of 15ms, duplex communication in the BLE->CAN flow.

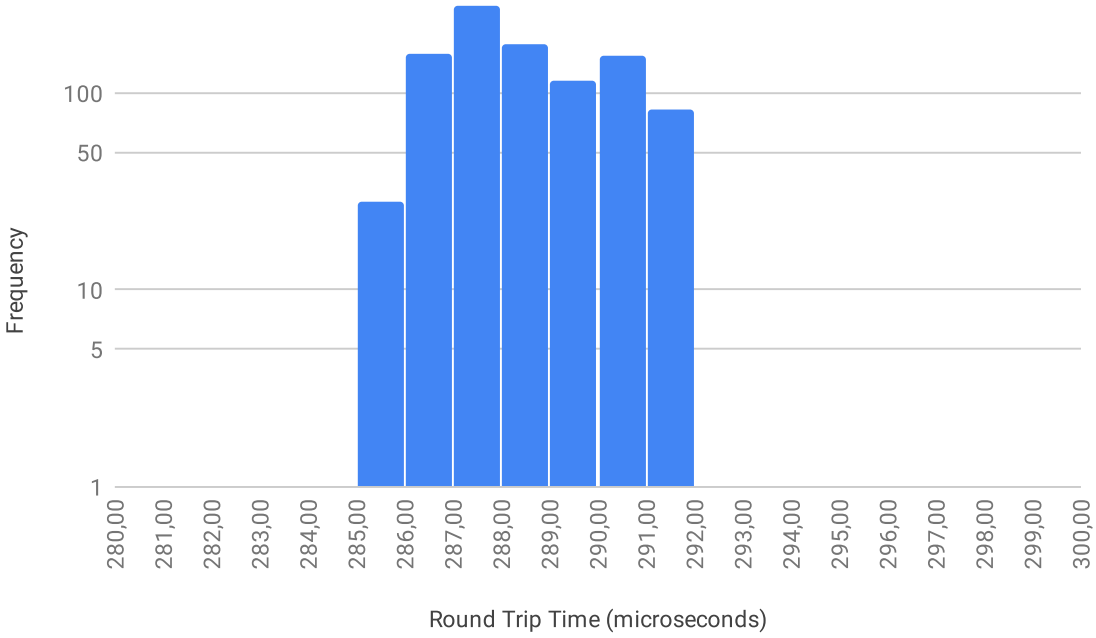

Figure 6.11: RTT with an wired connection, duplex communication in the CAN->UART flow.

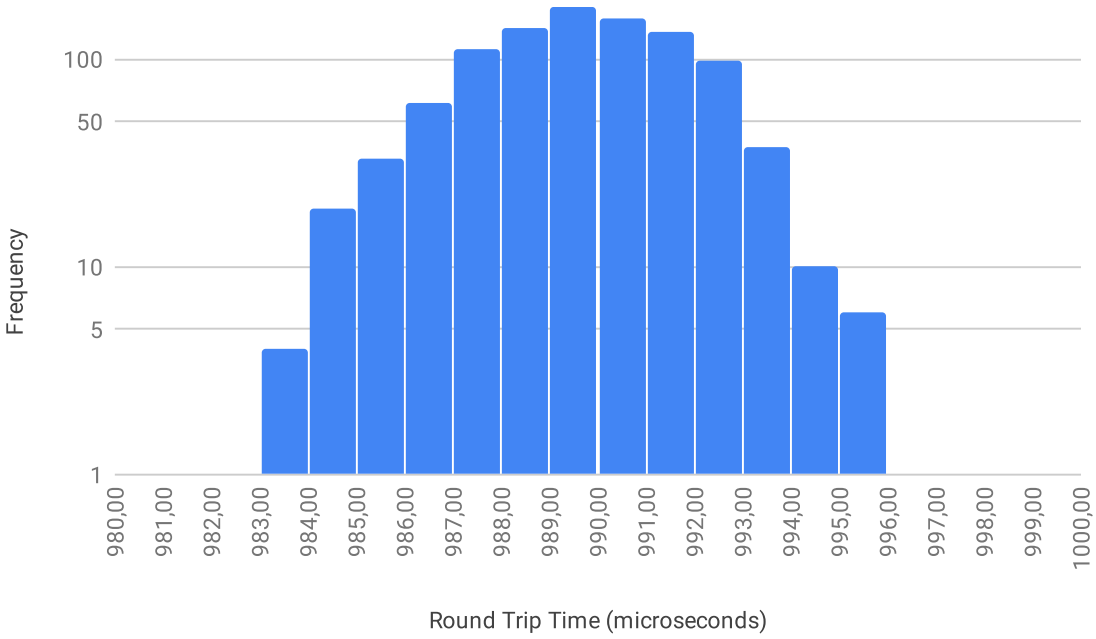

Figure 6.12: RTT with an wired connection, duplex communication in the UART->CAN flow.

6.1.2 Gateway Delay

The gateway delay is the processing time of the MLU between the moment it receives a message and the moment it sends that same message back to the network. This can happen in two ways:

1. Receives a message from the CAN bus, processes it, and forwards it to the Bluetooth link. The time it takes for a CAN message to be processed in this direction is directly influenced by the code displayed in listings 5.3 and 5.4 for the reception of a CAN frame and by the code displayed in listing 5.7 for the transmission of that same message to the serial port.
2. Receives a message from the Bluetooth link, processes it, and forwards it to the CAN bus. The time it takes for a CAN message to be processed in this direction is directly influenced by the code displayed in listings 5.5 for the reception of a message and by the code displayed in listing A.1 for the transmission of that same message to the CAN bus.

These next experiments aim to evaluate the GWD from our gateway in different conditions and configurations. The factors from this experiment are connection interval parameters, the data flow type and the data flow direction. The factors that don't show up on the table are the same as for the RTT experiment. The levels considered in this experiment are listed in the next table.

Connection Interval	Flow Type	Direction	Graph Number
min: 7,5ms max: 7,5ms	Simplex	CAN->BLE	Figure 6.13
		BLE->CAN	Figure 6.15
	Duplex	CAN->BLE	Figure 6.14
		BLE->CAN	Figure 6.16
min: 15ms max: 15ms	Simplex	CAN->BLE	Figure 6.17
		BLE->CAN	Figure 6.19
	Duplex	CAN->BLE	Figure 6.18
		BLE->CAN	Figure 6.20

Table 6.2: Collection of tests done for measuring GWD in different conditions.

The experiment described in this Section provides for 10 different type of samples (one for each combination of the factors). As said before each experiment is repeated 5 times, each one with 1000 messages for data flow.

The outcome from all GWD experiments may also be affected by uncontrolled factors, although much less than the RTT experiment.

6.1.2.1 Results

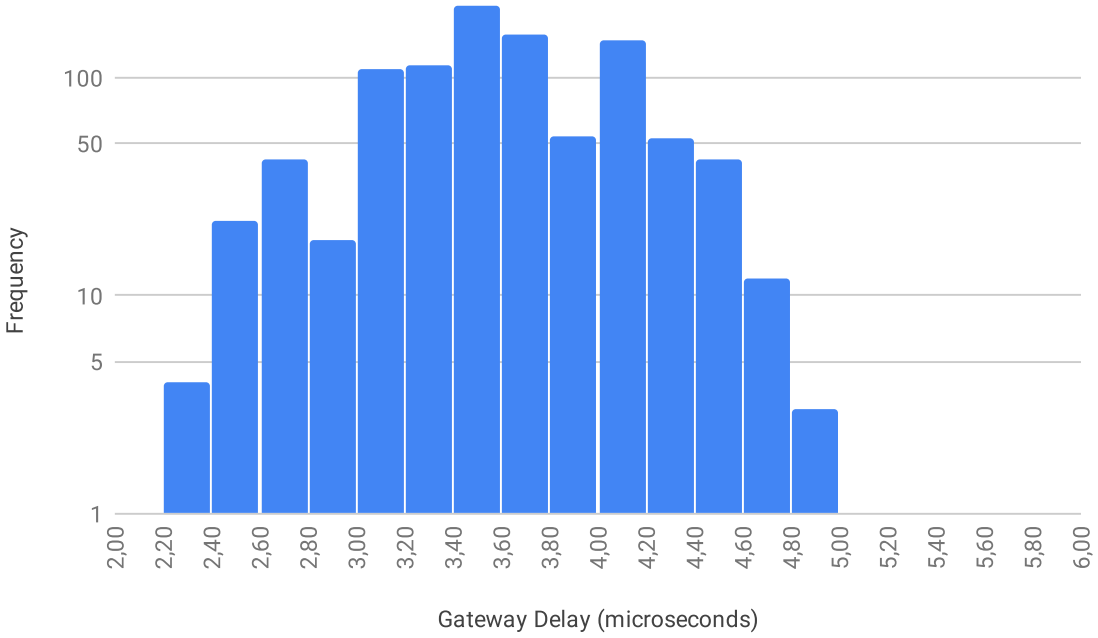

Figure 6.13: GWD for a conn. interval of 7.5ms, simplex communication in CAN->BLE flow.

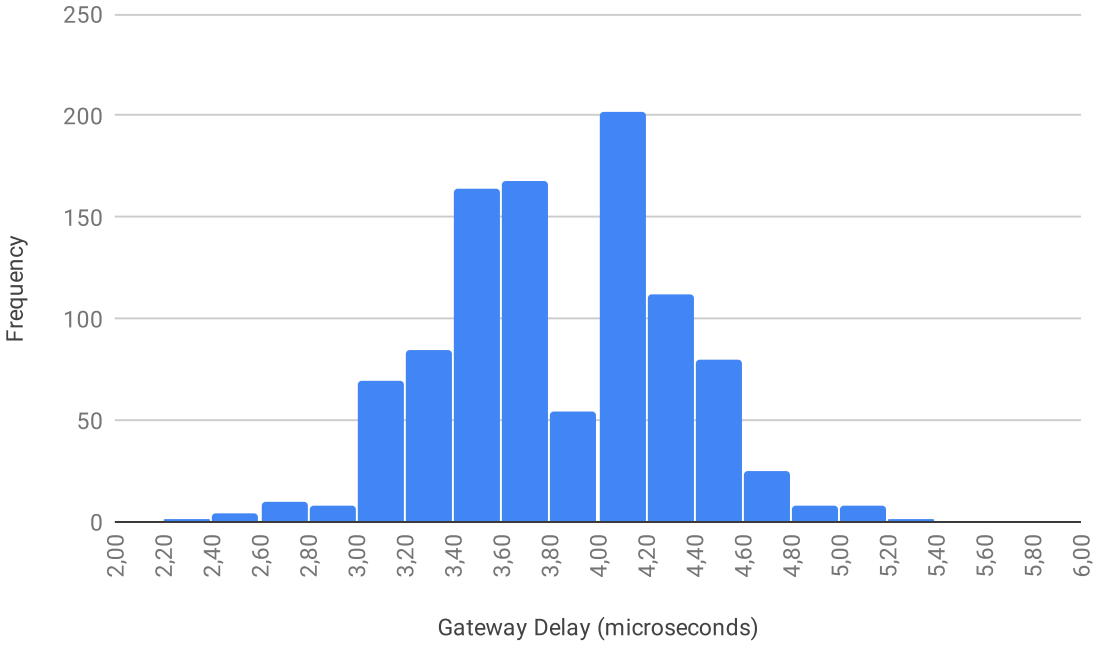

Figure 6.14: GWD for a conn. interval of 7.5ms, duplex communication in the CAN->BLE flow.

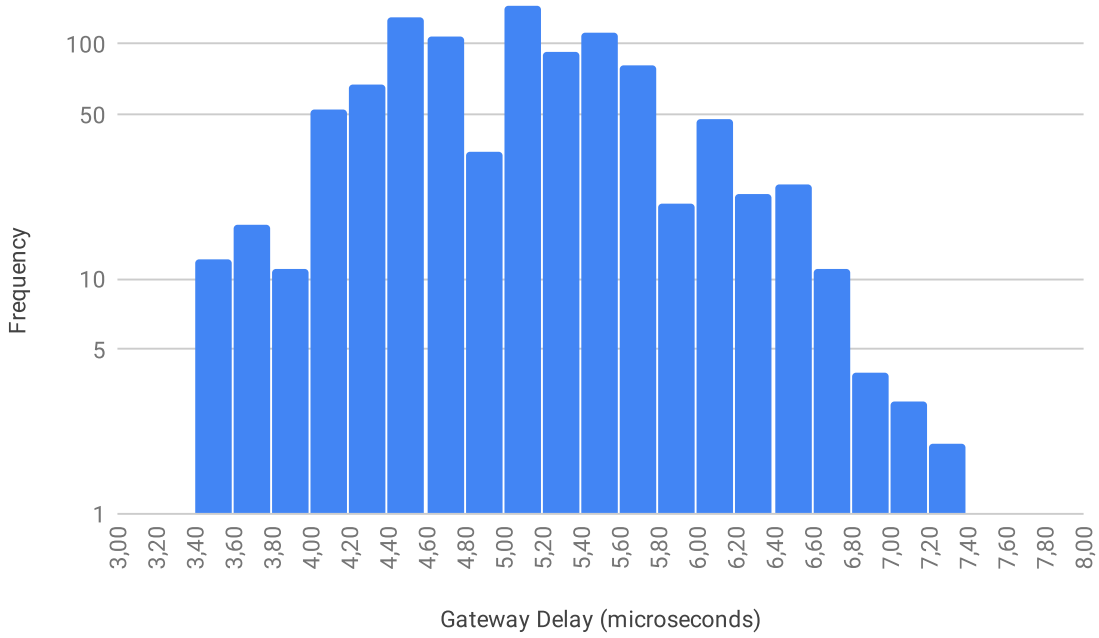

Figure 6.15: GWD for a conn. interval of 7.5ms, simplex communication in BLE->CAN flow.

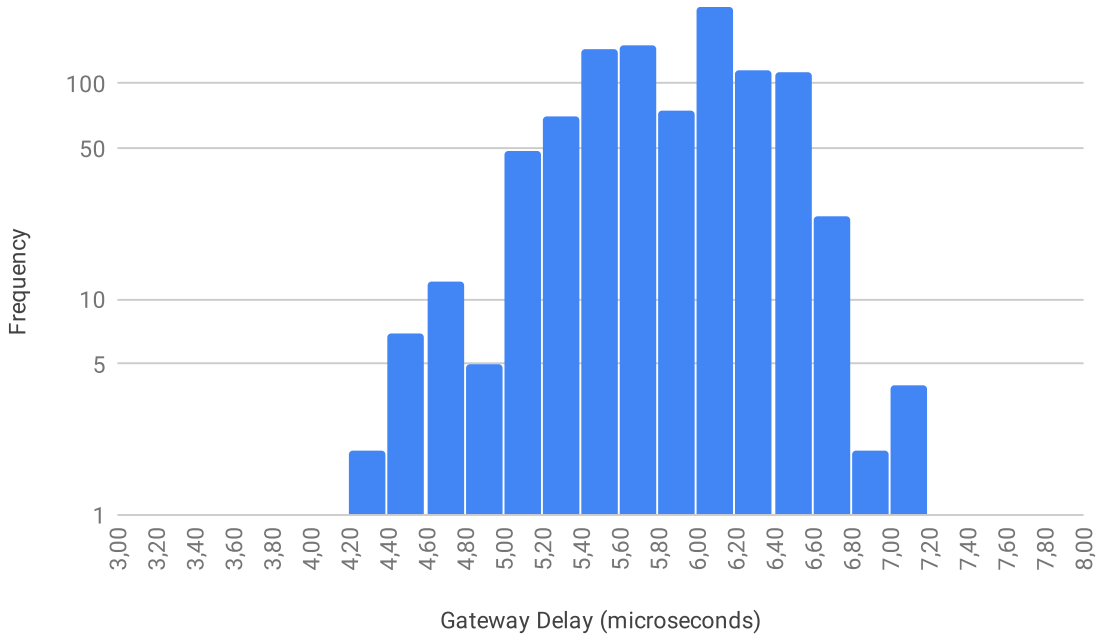

Figure 6.16: GWD for a conn. interval of 7.5ms, duplex communication in the BLE->CAN flow.

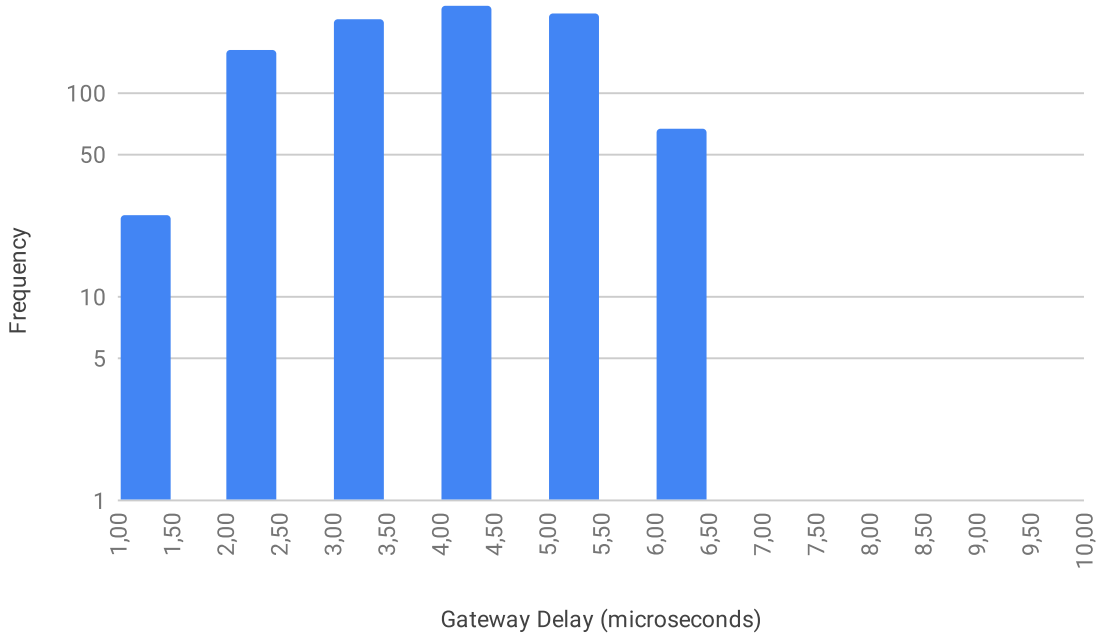

Figure 6.17: GWD for a conn. interval of 15ms, simplex communication in the CAN->BLE flow.

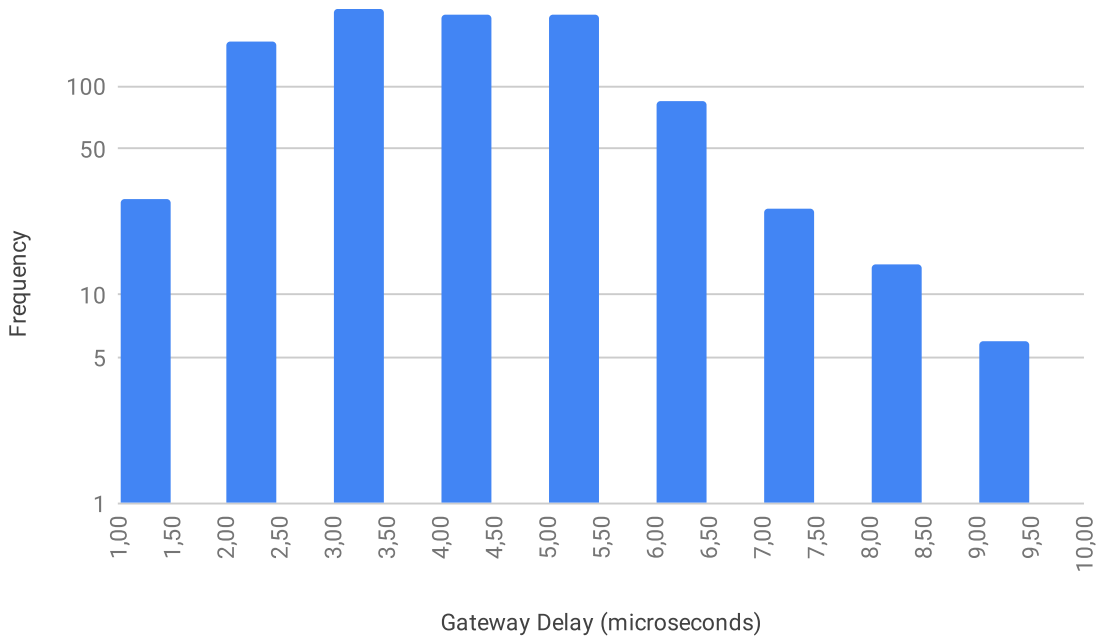

Figure 6.18: GWD for a conn. interval of 15ms, duplex communication in the CAN->BLE flow.

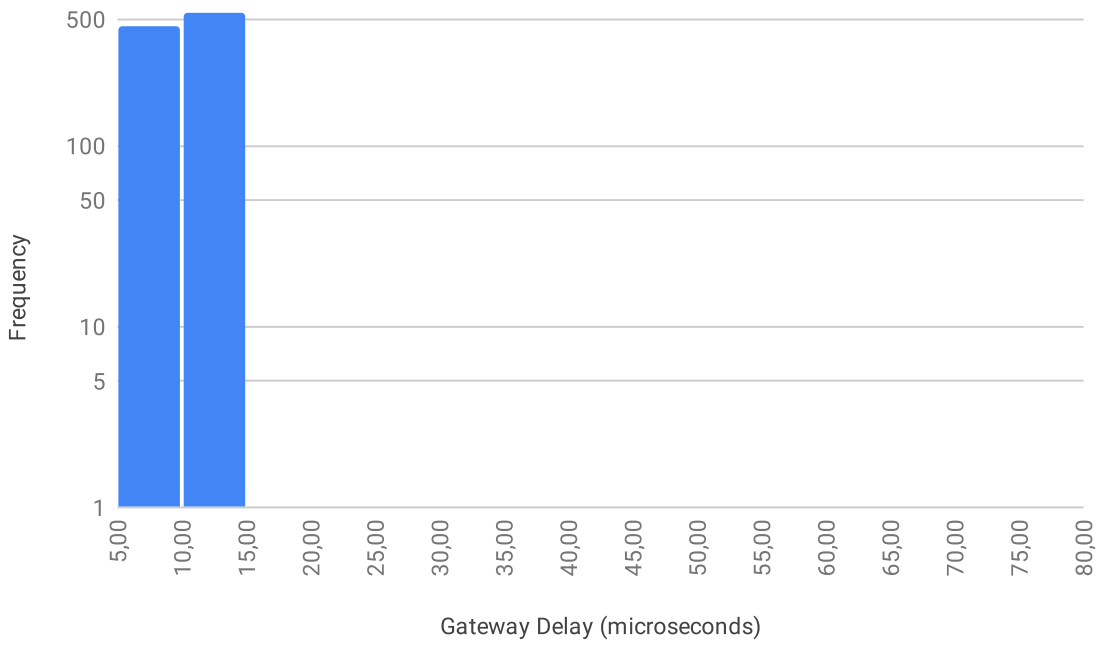

Figure 6.19: GWD for a conn. interval of 15ms, simplex communication in the BLE->CAN flow.

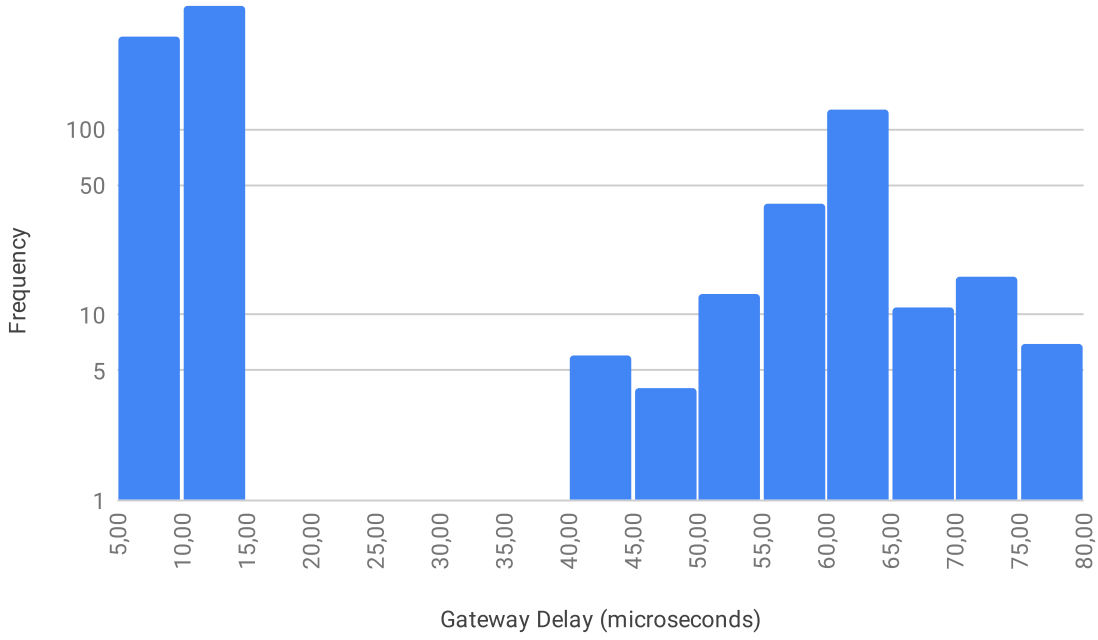

Figure 6.20: GWD for a conn. interval of 15ms, duplex communication in the BLE->CAN flow.

6.1.3 Throughput

Throughput is the rate of data successfully delivered over a communication channel and it is usually measured in bits per second (bit/s). This metric is calculated through the following formula:

$$R = D/T \tag{6.1}$$

D is the total amount of data captured in the MLU2. This value can be found by multiplying the total number of messages received by the size of those messages. T is the time that took to receive the total number of messages.

This experiment aims at evaluating the throughput from the Bluetooth Low Energy link. The test terminates when all messages are received by the MLU2. MLU2 dumps the messages after it receives them. In our case, the total amount of data transmitted is found by multiplying the number of CAN messages captured (1000) and the message size (14 bytes or 112 bits) while the elapsed time is calculated by subtracting the timestamp from the last message and the time stamp from the first message.

To measure throughput we will send a burst of messages at the speed of 1ms per message and take a timestamp before sending each message. In the receiver we take a timestamp on each message received. The total number of bits sent and received in each message divided by the time between messages gives the sending and receiving rate, respectively. If they are equal, the throughput of the system is higher than the maximum rate at which we generated messages and therefore we can use this rate as the lower bound of the throughput. If the reception rate is different (either because messages have been lost or because times between messages are higher) then throughput is the reception rate.

Conn. Interval	Flow Type	MLU Number	Graph Number
min: 7,5ms	Simplex	MLU1 (Sender)	Figure 6.21
max: 7,5ms		MLU2 (Receiver)	Figure 6.22

Table 6.3: Collection of tests done for measuring a lower bound for the throughput of our system.

Contrary to the other two experiments the factors and levels for this experiment are very simple. We will transmit from MLU1(Central) to MLU2(Peripheral) only using the BLE link in simplex communication mode with a Connection Interval of 7,5ms for both the minimum and maximum. There are other factors, if changed, that could influence this test, such as PHY (2Mbit/s), ATT MTU (247 bytes) and Operation Type (see Figure 4.18) and the distance from the MLU1 to the MLU2 which in this test will be approximately 20 centimetres.

The outcome from this experiment may be affected by uncontrolled factors. The main factor is the environment signal noise. A high noise in the environment may cause BLE packet losses, increase packet delay as well as degrading the link capacity.

6.1.3.1 Results

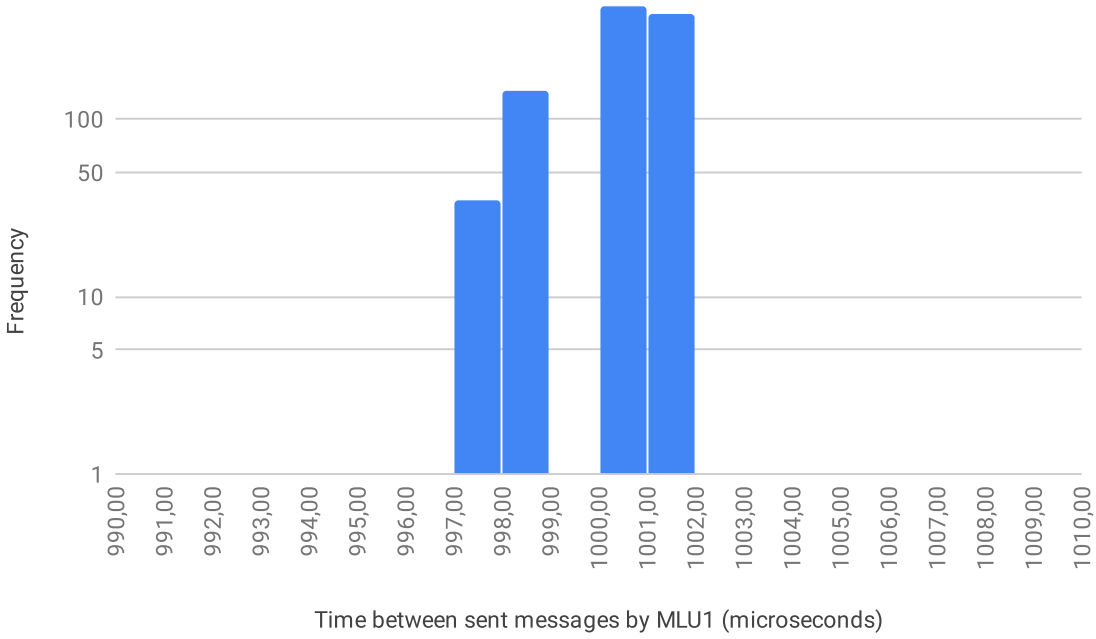

Figure 6.21: Time between transmitted messages for a conn. interval of 7.5ms in simplex communication.

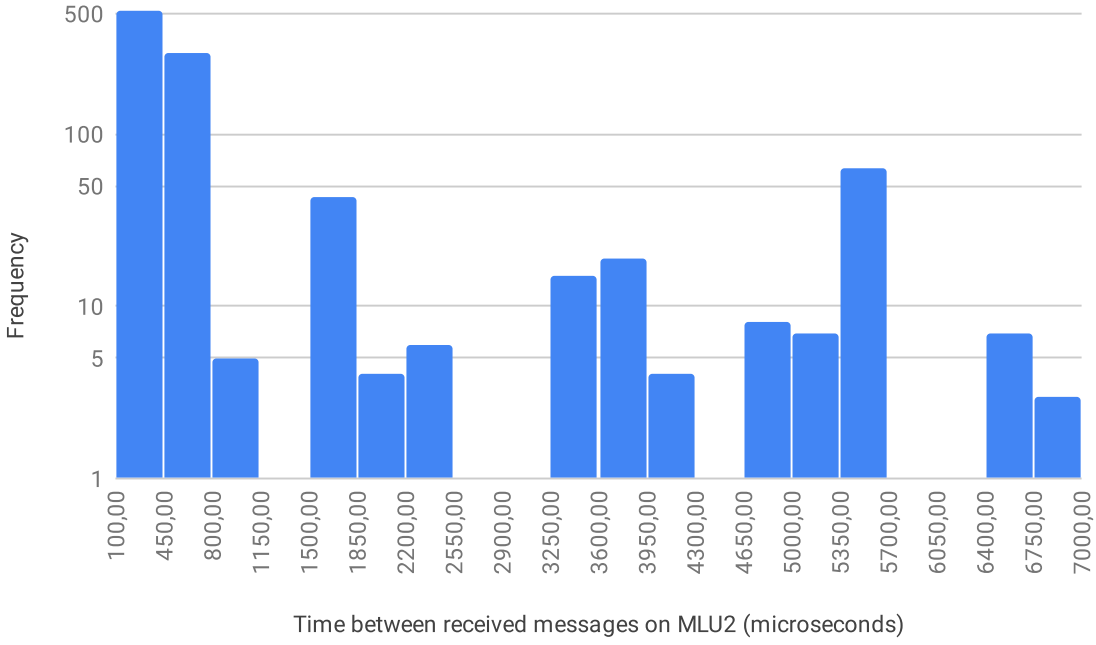

Figure 6.22: Time between received messages for a conn. interval of 7.5ms in simplex communication.

6.2 Remarks

Round Trip Time considerations

We will make a brief discussion on the results presented in the last section and some conclusions of those same results.

Connection Interval	Flow Type	Direction	Average RTT value
min: 7,5ms max: 7,5ms	Simplex	CAN->BLE	7,49ms
		BLE->CAN	7,49ms
	Duplex	CAN->BLE	7,49ms
		BLE->CAN	7,63ms
min: 15ms max: 15ms	Simplex	CAN->BLE	15,00ms
		BLE->CAN	15,02ms
	Duplex	CAN->BLE	14,98ms
		BLE->CAN	14,96ms
BLE link replaced with an UART	Duplex	CAN->UART	0,99ms
		UART->CAN	0,29ms

Table 6.4: The values for RTT for the various factors and levels.

For the RTT we can see that it is directly affected by the connection interval and that the flow type (simplex/duplex) and the direction of the data flow have little influence on this experiment. More importantly, we can clearly see the delay that a wireless connection causes compared to a wired solution.

Gateway Delay considerations

Connection Interval	Flow Type	Direction	Average GWD value
min: 7,5ms max: 7,5ms	Simplex	CAN->BLE	3,6μs
		BLE->CAN	5,1μs
	Duplex	CAN->BLE	3,8μs
		BLE->CAN	5,9μs
min: 15ms max: 15ms	Simplex	CAN->BLE	3,7μs
		BLE->CAN	10μs
	Duplex	CAN->BLE	3,9μs
		BLE->CAN	21,6μs

Table 6.5: The values for GWD for the various factors and levels.

As for the GWD we can see that for simplex communication the gateway delay stays consistent for each of the four combinations. It's when we use duplex communication that our delay increases, even more so if we chose for a higher connection interval. This is because there is more stress on the circular buffers which leads to more processing time.

Throughput considerations

The average time between CAN messages sent by the MLU1 was 1000μs and the average time between messages received from MLU2 was 1010μs. This means that we if take into account equation 6.1 and the fact that our message size is 112 bits, the average data rates are going to be 112kbit/s and 110,9kbit/s for the transmission and reception, respectively.

As said in section 6.1.3, the reception rate is different from the transmission rate and so, as a result, throughput is equal to reception rate (110,9kbit/s).

Note that being that the transmission rate is very similar to the reception rate, it probably means that the system is not yet in saturation and can still process messages faster and thus have a higher lower bound on the maximum tested throughput.

Chapter 7

Conclusions

This document addressed the design and the development of a wireless CAN bridge with the main purpose of cable replacement. For the realization of this project different types of wireless communications that could be relevant for this use case were studied and a discussion was made around these same possibilities. After this study, the somewhat recent technology Bluetooth Low Energy was the chosen wireless communication protocol, mainly due to the improvements featured on the last core specification released by the Bluetooth Special Interest Group in 2016 (version 5).

The hardware study, although done, was not exhaustive, much due to the fact that nowadays it is possible to find compatible wireless modules for almost any board, being that this was also the case with AddVolt's MLU.

It was defined the use of the NINA-B1 module, developed by u-blox, because of its easy integration and due to the fact that it has Bluetooth 5 and also because it includes the non standard Serial Port Service, developed by u-blox, for a fast and secure connection between two of these same modules with the objective of replacing an existent cable connection.

The firmware development was done in C, with the help of DAVE and integrated within AddVolt's already existent framework.

Overall the development was successful, having been possible to transmit Extended and Standard CAN messages in both directions and at the same time between the two gateways via a Bluetooth Low Energy link and in compliance with all the requirements presented in section 4.1.

A significant number of experiments were carried out in the end, that allow us to have a better perspective on the performance of the implemented Bluetooth - CAN bridge. Only the throughput test, while on duplex communication, was not done, being easily performed in the future. The reason we failed to perform this test was due to lack of time in the final phase of the project and also because the hardware setup for all experiments are different and require changes to be made for each one.

Despite these difficulties, we have been successful in achieving several performance metrics for the gateway, such as Round Trip Time, Gateway Delay and Maximum Throughput. The implementation was fortunate and the final product proved to be functional in the replacement of a cable between two different CAN buses.

7.1 Future Work

In this section we present some suggestions that can lead to future improvements of the prototype developed up until this point. Briefly, as suggestions for future work, the following is proposed:

- Implement a feature that will allow the driver of the truck to read diagnostic data from the CAN bus, via a Bluetooth link.
- Develop the firmware with a more low level approach regarding the S132 SoftDevice protocol stack solution on NINA-B1. This will allow for improvements in throughput and a better control of the Bluetooth connection between the two modules.
- Re-work the firmware implemented in order two improve the level of Bluetooth Low Energy security that was used for this project, and check if it does not compromise the throughput of the system.
- Add a checksum byte at the end of our CAN message binary format to improve the robustness of the protocol.
- Perform a real operation test in the field and verify the correct communication between both CAN buses and measure the number of lost messages and the delay associated with the load of a real vehicle CAN bus.

Appendix A

C code segment

```
void send_CAN_msg(){

   //MSG NOT STARTED
   if(!buffer_empty(&B_U_C) && flag1==0){
      checkbyte = buffer_read(&B_U_C);
      if(checkbyte == startbyte)
         flag1=1;
   }

   //SAVE LENGTH AND ID_TYPE
   if(!buffer_empty(&B_U_C) && flag1==1){
      checkbyte = buffer_read(&B_U_C);
      if(checkbyte != startbyte){

         msg_length = checkbyte;

         if( (msg_length & 0x10) == 0)
            flag2=0;//STD
         else{
            flag2=1;//EXT
            msg_length = msg_length-16;
         }

         if(msg_length==0)
            flag3=1;//NO DATA
         else
            flag3=0;//DATA

         flag1=2;
      }
      else{
         flag1=1;
         flag2=0;
         flag3=0;
      }
```

```
}

//SAVE STD_ID_H
if(!buffer_empty(&B_U_C) && flag1==2 && flag2==0){
   checkbyte = buffer_read(&B_U_C);
   if(checkbyte != startbyte){

      STD_ID_H = checkbyte;
      flag1=3;
   }
   else{
      flag1=1;
      flag2=0;
      flag3=0;
   }
}

//SAVE STD_ID_L
if(!buffer_empty(&B_U_C) && flag1==3 && flag2==0){
   checkbyte = buffer_read(&B_U_C);
   if(checkbyte != startbyte){

   STD_ID_L = checkbyte;
   COMPLETE_STD_ID = ((uint16_t)STD_ID_H << 8) | STD_ID_L;
   flag1=10;
   }
   else{
      flag1=1;
      flag2=0;
      flag3=0;
   }
}

//SAVE EXT_ID_HH
if(!buffer_empty(&B_U_C) && flag1==2 && flag2==1){
   checkbyte = buffer_read(&B_U_C);
   if(checkbyte != startbyte){

   EXT_ID_HH = checkbyte;
   flag1=3;
   }
   else{
      flag1=1;
      flag2=0;
      flag3=0;
   }
}

//SAVE EXT_ID_HL
```

```
84    if(!buffer_empty(&B_U_C) && flag1==3 && flag2==1){
          checkbyte = buffer_read(&B_U_C);
86        if(checkbyte != startbyte){

88        EXT_ID_HL = checkbyte;
          flag1=4;
90        }
          else{
92           flag1=1;
             flag2=0;
94           flag3=0;
          }
96    }

98    //SAVE EXT_ID_LH
      if(!buffer_empty(&B_U_C) && flag1==4 && flag2==1){
100       checkbyte = buffer_read(&B_U_C);
          if(checkbyte != startbyte){
102
          EXT_ID_LH = checkbyte;
104       flag1=5;
          }
106       else{
             flag1=1;
108          flag2=0;
             flag3=0;
110       }
      }
112
      //SAVE EXT_ID_LL
114   if(!buffer_empty(&B_U_C) && flag1==5 && flag2==1){
          checkbyte = buffer_read(&B_U_C);
116       if(checkbyte != startbyte){

118       EXT_ID_LL = checkbyte;
          COMPLETE_EXT_ID = (   (  ((uint32_t)EXT_ID_HH << 24) | ((uint32_t)EXT_ID_HL
       << 16)  )  |  (  ((uint16_t)EXT_ID_LH << 8) | EXT_ID_LL  )   );
120       flag1=10;
          }
122       else{
             flag1=1;
124          flag2=0;
             flag3=0;
126       }
      }
128
      //SAVE "msg_length" DATA BYTES
130   if(!buffer_empty(&B_U_C) && flag1==10 && flag3==0 && (can_data_index<msg_length)
        ){
```

```
      checkbyte = buffer_read(&B_U_C);
      if(checkbyte != startbyte){

         can_data[can_data_index] = checkbyte;
         can_data_index++;
         if(can_data_index==msg_length){
            flag1=20;
            can_data_index=0;
         }
      }
      else{
         flag1=1;
         flag2=0;
         flag3=0;
      }
   }

   //MESSAGE WITHOUT DATA
   if(flag1==10 && flag3==1)
      flag1=20;

   //READY TO SEND STD_MO TO CAN bus
   if(flag1==20 && flag2==0){
      CAN_config_LMO(CAN1Handle, COMPLETE_STD_ID, msg_length, 1, 1, 0x00000000,11);
      CAN_transmit(CAN1Handle, 1, can_data);
      flag1=0;//READY TO READ NEW CAN MESSAGE AGAIN
   }
   //READY TO SEND EXT_MO TO CAN bus
   if(flag1==20 && flag2==1){
      CAN_config_LMO(CAN1Handle, COMPLETE_EXT_ID, msg_length, 1, 3, 0x00000000,29);
      CAN_transmit(CAN1Handle, 3, can_data);
      flag1=0;//READY TO READ NEW CAN MESSAGE AGAIN
   }
}
```

Listing A.1: Transmission of a CAN frame to the CAN bus.

References

[1] Nam Seok Kim and Bert Van Wee. Assessment of CO2 emissions for truck-only and rail-based intermodal freight systems in Europe. 2009. URL: http://www.tandfonline.com/doi/abs/10.1080/03081060903119584.

[2] Eelco den Boer, Sanne Aarnink, Florian Kleiner, and Johannes Pagenkopf. Zero emissions trucks - An overview of state-of-the-art technologies technologies and their potential. 2013. URL: https://www.theicct.org/sites/default/files/publications/CE_Delft_4841_Zero_emissions_trucks_Def.pdf.

[3] Luis Simoes. Relatorio de Sustentabilidade e Contas. Technical report, 2016. URL: https://www.luis-simoes.com/wp-content/uploads/2018/11/PT-Relatorio-LS-2016.pdf.

[4] Duncan MacMichael. Windows Network Architecture and the OSI Model, 2017. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/network/windows-network-architecture-and-the-osi-model.

[5] José Manuel Ramos Pinto de Sousa. Aplicação do protocolo CAN a uma rede comunicações. Master's thesis, 2000.

[6] Infineon Technologies AG. XMC1000, XMC4000 - Universal Serial Interface Channel (USIC), 2015. URL: https://www.infineon.com/dgdl/Infineon-USIC-XMC1000_XMC4000-AP32303-AN-v01_00-EN.pdf?fileId=5546d4624e765da5014ed93960ae3391.

[7] Ariel Nuñez. An Introduction to the CAN Bus: How to Programmatically Control a Car, 2017. URL: https://news.voyage.auto/an-introduction-to-the-can-bus-how-to-programmatically-control-a-car-f1b18b

[8] Ernst Christmann. Data Communication in the Automobile – Part 2: Reliable Data Exchange with CAN. 2016. URL: https://pdfs.semanticscholar.org/35fe/b36f104b4e7bb98357c7ef72d58b8118dfb2.pdf.

[9] Hugo Miguel Gomes Ribeiro. Dados CAN / OBD2 em tempo real de viaturas auto. Master's thesis, 2015.

[10] Hanxing Chen. Research on the Controller Area Network. 2009. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5116731.

[11] Luis Miguel Moreira Lino Ferreira. Implementacão de uma Camada de Aplicação sobre a Rede CAN. Master's thesis, 1996.

[12] Pat Richards. A CAN Physical Layer Discussion. 2002. URL: http://ww1.microchip.com/downloads/en/appnotes/00228a.pdf.

[13] Conal Watterson. Configure Controller Area Network (CAN) Bit Timing to Optimize Performance. 2014. URL: https://www.analog.com/media/en/analog-dialogue/volume-48/number-3/articles/configure-can-bit-timing.pdf.

[14] Karl Henrik Johansson, T Martin, and Lars Nielsen. Vehicle Applications of Controller Area Network. URL: https://link.springer.com/content/pdf/10.1007%2Fb137198.pdf.

[15] HMS Industrial Networks. Wireless technologies for industrial communication. URL: https://www.anybus.com/technologies/wireless-others/wireless-differences.

[16] André Teixeira da Silva. Módulos de Comunicação Wireless para Sensores. Master's thesis, 2007.

[17] Proemion GmbH. CANlink Bluetooth (HW4.0) - Data Sheet. URL: https://www.proemion.com/en/products/embedded-systems/wireless-can-interfaces/canlink-bluetooth.html.

[18] HMS Industrial Networks. Ixxat CANblue II - Extended User Manual. URL: https://www.ixxat.com/products/products-industrial/gateways-and-bridges/canblue-ii.

[19] HMS Industrial Networks. Anybus Wireless Bridge - Serial - Bluetooth Data Sheet. URL: https://www.anybus.com/products/wireless-index/anybus-wireless-bridge/detail/anybus-wireless-bridge---serial---bluetooth.

[20] Case GmbH. Bluetooth wireless technology Adaptor for CAN-Bus Applications Datasheet. URL: http://www.case-gmbh.de/DS4113D.pdf.

[21] Kevin Townsend, Carles Cufí, Akiba, and Robert Davidson. *Getting Started with Bluetooth Low Energy*. O'Reilly Media, Inc., 2014.

[22] Flávio Lopes de Vasconcelos. Rede de sensores Bluetooth de baixo Consumo Energético com Interface CAN. Master's thesis, 2017.

[23] Bluetooth Special Interest Group. Bluetooth Specifications. URL: https://www.bluetooth.com/specifications/gatt/.

[24] Rohde & Schwarz GmbH. From cable replacement to the IoT Bluetooth 5. 2016. URL: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma108/1MA108_4e_Bluetooth_WhitePaper.pdf.

[25] André Guedes Linhares. Performance Measurements and Analysis of Bluetooth Low Energy. Master's thesis, 2016.

[26] Bluetooth Special Interest Group. Bluetooth Core Specification Version 5.1, 2019. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification/.

[27] Mark Loveless. Understanding Bluetooth Security, 2018. URL: https://duo.com/decipher/understanding-bluetooth-security.

[28] Bluetooth low energy central tutorial. URL: https://devzone.nordicsemi.com/nordic/short-range-guides/b/bluetooth-low-energy/posts/ble-central-tutorial.

[29] Martin Woolley. Bluetooth 5: Go Faster, Go Further. URL: https://www.bluetooth.com/bluetooth-resources/bluetooth-5-go-faster-go-further/.

[30] MikroElektronika. mikroBUS Standard specifications, 2015. URL: https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf.

[31] MikroElektronika. Wireless Connectivity Boards. URL: https://www.mikroe.com/click/wireless-connectivity.

[32] Infineon Technologies AG. XMC™ microcontroller and DAVE™ software development platform, 2016. URL: https://www.infineon.com/dgdl/Infineon-DAVE-Digital-Application-Virtual-Engineer-for-XMC-MCUs-BC-v01_00-EN.pdf?fileId=5546d4624e765da5014edee86bb71ac8.

[33] Peak-System Technik Gmbh. PCAN-USB CAN Interface for USB - User Manual, 2019. URL: https://www.peak-system.com/produktcd/Pdf/English/PCAN-USB_UserMan_eng.pdf.

[34] MikroElektronika. BLE 3 Click. URL: https://www.mikroe.com/ble-3-click.

[35] u-blox AG. u-connectXpress Throughput measurements - Application Note, 2019. URL: https://www.u-blox.com/sites/default/files/u-connectXpress-ThroughputMeasurements_ApplicationNote_%28UBX-17023548%29.pdf.

[36] Infineon Technologies AG. XMC4500 (ARM Cortex-M4 32-bit processor core) Data Sheet. URL: https://www.infineon.com/dgdl/Infineon-XMC4500-DS-v01_05-EN.pdf?fileId=5546d46254e133b40154e1b56cbe0123.

[37] Infineon Technologies AG. Evaluation Board For XMC4000 Family - Board User's Manual, 2014. URL: https://www.infineon.com/dgdl/Board_Users_Manual_XMC4500_Relax_Kit-V1_R1.2_released.pdf?fileId=db3a30433acf32c9013adf6b97b112f9.

[38] u-blox AG. NINA-B1 series - Stand-alone Bluetooth Low Energy module. URL: https://www.u-blox.com/en/product/nina-b1-series.

[39] u-blox AG. u-connectXpress software User Guide. URL: https://www.u-blox.com/sites/default/files/u-connectXpress_UserGuide_%28UBX-16024251%29.pdf.