WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

BACHELOR THESIS

SUPERVISOR:
Asistent Dr. Florin Roşu

GRADUATE:
Matei Ionuţ Pop

TIMIŞOARA
2025

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

SensorView

SUPERVISOR:
Asistent Dr. Florin Roşu

GRADUATE:
Matei Ionuţ Pop

TIMIŞOARA
2025

Abstract

Throughout history, the invention of automobiles has continuously changed the world, our livestyles and transportation as we know it. As technology advances, so will the automotive industry—evolving from purely mechanical systems to highly complex networks of hardware.The modern automobile is a network hosting wireless technologies like GPS, Bluetooth®, Wi-Fi®, near field communication (NFC) and cellular (4G/5G), all serving as complementary connectivity options to help enhance functionality, user safety and comfort. For a long time, radio frequency communication has been used in the automotive industry to monitor vehicle performance. RF sensors have some drawbacks: high energy consumption and the lack of bidirectional communication.

This thesis presents the development of a Windows Application in C# that collects and displays real-time data from a BLE sensor. The tool is designed to assist developers in analyzing vehicle states after prolonged automatic tests, enabling efficient diagnostics and system validation.

The application facilitates seamless communication between the BLE sensor and the computer, ensuring that developers can monitor key vehicle parameters without relying on traditional wired diagnostics like CAN Bus or OBD-II. The project integrates real-time data visualization, logging and error detection, providing an intuitive interface for the developer to efficiently debug and evaluate the performance.

This work explores the architecture of the system, including hardware components, software implementation, and data processing techniques. It discusses the role of BLE in modern automotive testing and how this approach enhances flexibility, accessibility, and accuracy in ECU diagnostics.

Through comprehensive testing and validation, the application proves to be a valuable tool in automotive software development, offering a wireless and efficient alternative for analyzing sensor data.

Contents

Chapter 1

Introduction

1.1 Motivation

Curiosity has always been a driving force, especially when it came to understanding how things work beneath the surface. Early experiences with technology naturally evolved into a deep interest in the interaction between hardware and software—an interest that eventually led to pursuing a degree in Computer Science.

Throughout university, exposure to a wide array of topics—from programming and networks to databases and artificial intelligence—helped shape a technical foundation. Among them, low-level programming in C stood out as particularly engaging. At the same time, a personal decision—buying a motorcycle—sparked an interest in the automotive industry. That combination of passion and direction led to an opportunity at Continental Automotive Romania, where the chance to work on an impactful real-world project presented itself.

Getting hands-on experience revealed just how vital embedded systems are in modern vehicles. Learning about the CAN interface and the intricate communication between sensors and ECUs highlighted the complexity and precision required to ensure vehicle safety and reliability. Each data frame on the CAN bus can carry crucial information—ranging from tire pressure abnormalities to subtle environmental parameters inside the cabin.

However, observing the post-test process uncovered a significant pain point: interpreting the raw sensor data collected after extended automated test cycles proved time-consuming and inefficient. Engineers often had to sift manually through large sets of Controller Area Network (CAN) data to identify anomalies or validate system behavior.

Recognizing this gap in the workflow, the idea for a new tool emerged—one capable of collecting, decoding, displaying, and interpreting Bluetooth Low Energy (BLE) sensor data in real time. The objective was clear: improve testing efficiency, reduce manual effort, and bring greater clarity to sensor data analysis through a user-friendly Windows Forms application. The project is not only a response to a technical challenge but also a personal contribution to improving day-to-day development processes in the automotive environment.

1.2 Objectives

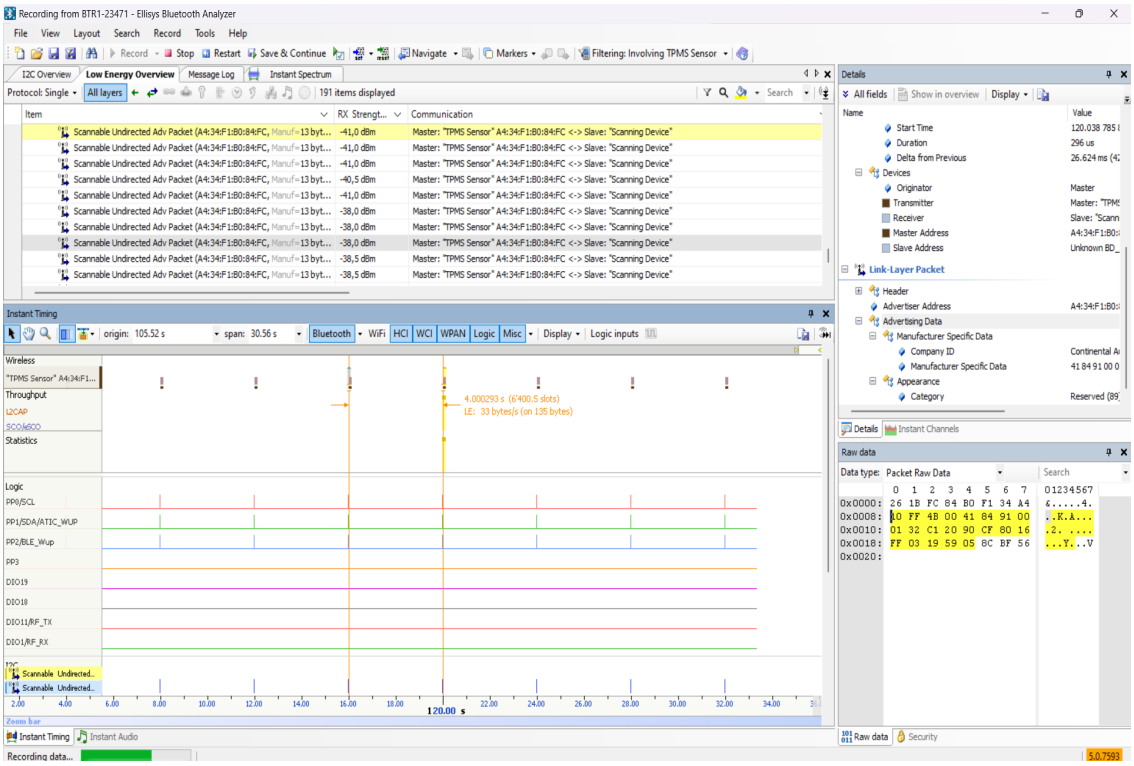

Figure 1.1: Ellisys Bluetooth Analyzer

Before explaining the objectives of the project, I would like to explain how the current process of analyzing the data works. As shown in the above picture, my colleagues use a tool that allows them to monitor the data received from the BLE sensors. This tool is not user-friendly and requires a lot of manual work to analyze the data. Each small brown sequence in the window below represents a frame. The time interval between frames is displayed by dragging the cursor between frames. On the right side, the structure of the message and the raw data appears. In order to find out what state is the car in the next things need to be analyzed: the raw data, structure of the message, structure of the package, time interval and number of frames.

The process is time-consuming and can lead to errors in interpretation. For each burst of data, my colleagues have to manually analyze each dataframe to identify the structure of the frames and finally figure out which state the sensor is in.

The main objective of the thesis is to develop an application that allows my colleagues to analyze the tests performed on the BLE sensors after long-duration automated tests. The application aims to provide a user-friendly, intuitive interface that allows users to visualize the data in real time, making it easier to identify patterns and anomalies.

This application is designed to:

- Collect data from BLE sensors in real time
- Store the data in a database for later analysis

- Allow users to filter and search for specific data points
- Provide a way to export the data in various formats
- Provide a intuitive interface for visualizing the data

1.3 Structure of the paper

This thesis is organized into nine chapters, each one building toward a deeper understanding of the project's purpose, implementation, and impact.

- **Chapter 1 – Introduction:** This chapter sets the stage with a personal look at what motivated the project, the specific goals pursued, and a roadmap of the thesis. It highlights how real-world experience and curiosity merged to spark the idea
- **Chapter 2 – State of the Art:** Offers a technical overview of existing solutions and background knowledge that informed the development of this project. It explores previous approaches to BLE sensor data monitoring, the role of ECU diagnostics in automotive testing, and technologies relevant to Windows Forms applications. Communication protocols like CAN and BLE are also introduced and discussed
- **Chapter 3 – System Architecture:** This chapter dives into the high-level architecture of the system. It outlines the physical and digital components used, explains how they interact, and walks through the flow of data between sensors, the bus, and the application
- **Chapter 4 – Implementation Details:** In this chapter is described the core of the technical contribution. The development process is broken down in detail—covering the environment setup, Bluetooth Low Energy (BLE) integration in C#, user interface features, and the logic that powers data handling and error management
- **Chapter 5 – Testing and Validation:** This chapter documents the application was tested, what scenarios were used, and how performance was evaluated. It includes feedback from developers who tested the tool and suggestions that helped refine the final product
- **Chapter 6 – Usage Guide:** A practical guide for developers who wish to use the application. It describes how to set it up, what to expect from the interface, and how to troubleshoot common issues.

Each chapter contributes to the overall narrative of solving a real-world problem using practical, modern software engineering—grounded in curiosity, driven by efficiency, and shaped by hands-on experience.

Chapter 2

State of the art

2.1 Existing Solution for BLE Sensor Data Monitoring

While there are some tools on the market that offer bluetooth sniffing and packet analytics, they all lack something. These tools typically feature non-user-friendly interfaces and are intended for broad usage scenarios, making them unsuitable for specialized testing workflows.

In addition, users would still need to process the data manually, which slows down the diagnostic process and increases the risk of human error.

Some of the applications introduced in [7]:

- **nRF Connect** is a mobile application. It generic tool that allows the user to scan, advertise and explore Bluetooth Low Energy (BLE) devices and communicate with them. Because it is a mobile application, it is not suitable for automotive testing;
- **BlueZ** is a Linux-based Bluetooth protocol stack. It is a powerful tool for developers, but it requires a deep understanding of the Linux operating system and Bluetooth protocols. It is not user-friendly and requires significant technical expertise to use effectively;
- **OBD Fusion® by OCTech** (formerly TouchScan) is an app that allows users to read OBD2 vehicle data directly from your Android phone or tablet. Just as **nRF Connect**, it is a phone app and it is not suitable for automotive testing. Besides this, it is designed for OBD2 data, which is not the same as BLE data and it is not free to use;
- **Acrylic Bluetooth Analyzer** is a software that allows users to scan, monitor and analyze Bluetooth Low Energy devices. Even though it is a powerful tool, it still doesn't interpret the data packages so the manual processing is still required.

2.2 Technologies for Windows form Applications

This project is developed using the .NET framework for Windows form applications. The .NET framework is a software development platform developed by

Microsoft that provides a large library of pre-built code and tools for building Windows applications. It is widely used in the industry and has a large community of developers, making it a good choice for this project because it has a lot of resources and support available.

The .NET framework provides the Windows.Device.Bluetooth namespace, which allows developers to work with Bluetooth devices in a Windows environment, which made the job easier.

Because we use Windows at work, I decided to use Windows Form Application as the user interface. Windows Forms is a UI framework for building Windows desktop applications. This framework provides a set of controls and components that can be used to create rich user interfaces.

2.3 Communication protocols

Communication protocols are sets of defined rules that outline how digital messages are formatted, transmitted, and received. They are very important in telecommunication systems and networking because they ensure that devices with different hardware, software, and architectures can understand and process data consistently and reliably.

2.3.1 Controller area network

The Controller Area Network (CAN) is a protocol message-based designed to allow ECUs (Electronic Control Units) to communicate with each other in a reliable and priority-based system. In this network, messages are called frames and are received by all nodes, but processed only by the ones with the right address. This makes a central unit unnecessary [1] .

The CAN bus standard is widely accepted and is used in airplanes, elevators, ships, household appliances and many more. It is used because it has the following benefits:

- **Efficient**: CAN messages are prioritized by IDs, which allows higher-priority messages to gain immediate bus access without delay;
- **Robust**: The protocol is highly reliable, with built-in error detection and correction mechanisms to ensure data integrity;
- **Scalable**: CAN supports multiple nodes on the same network, making it suitable for complex systems with numerous devices;
- **Cost-effective**: The simplicity of the CAN protocol reduces hardware costs and simplifies implementation;
- **Real-time communication**: The deterministic nature of CAN ensures timely delivery of critical messages, which is essential for automotive systems.

Devices on a CAN bus are called "nodes". Each node has a CPU, CAN controller and a transceiver, which adapts the signal levels of both send and received data.

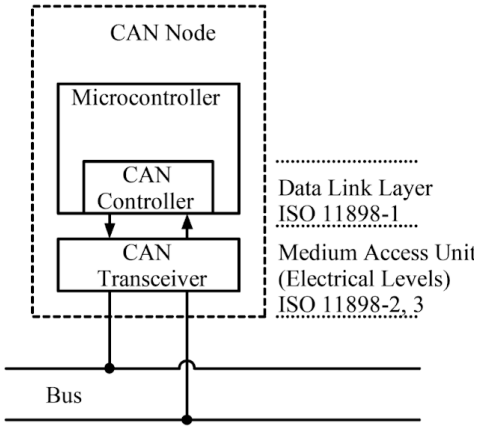

Figure 2.1: CAN node structure [1]

In the CAN protocol, transmitted messages are referred to as frames. There are four types of frames: data frames (transfer data to one or multiple nodes), remote frames (ask for data from other nodes), error frames and overload frames(report overload conditions). All these types are represented by wo variants of lengths.

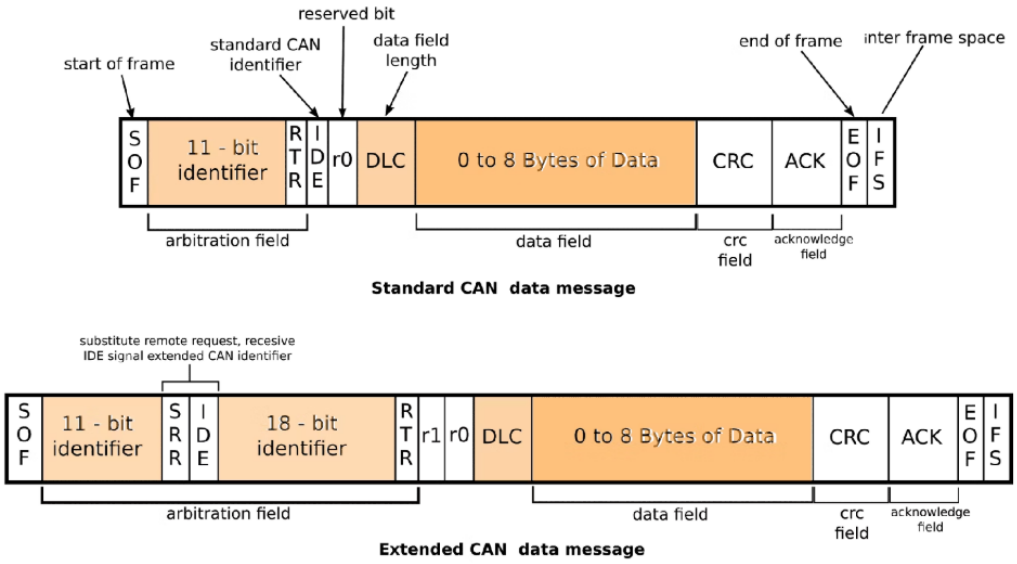

Figure 2.2: Standard and Extended frame of the CAN data message architecture [1]

2.3.2 Bluetooth and Bluetooth Low Energy

The **Bluetooth** Specification was invented in the 90s. Its name comes from the historical figure **Harold "Bluetooth" Gormsson**, a 10th century Viking king. "Bluetooth" was originally supposed to be just a codename, but the name remained.

Bluetooth is a widely adopted wireless technology operating in the 2.4 GHz ISM band, designed for low-power, short-range communication between devices[10].

It supports various use cases through standardized profiles, including audio, data transfer, and control.

Bluetooth Low Energy is suited for applications running on wearable devices, IoT sensors, and devices where power efficiency is critical and data transmission occurs in small packages. In [7], a few of the most valuable advantages of BLE are :

- Low power consumption: because its ability to place itself in sleep mode, the battery consumption is not that high;
- Low development costs: because its popularity, the BLE modules and chipsets are low cost when compared to other similar technologies on the market such as Zigbee, Z-Wave or LoRa;
- Prevalence in smartphones: most of the people in the world own a smartphone that already has the BLE hardware inside. this gives developers a much larger potential user base.

While BLE excels in some areas, it has some limitations as well:

- Range (20 meters [9])
- Limited data throughput (2.1 Mbps for Bluetooth and 1 Mbps for BLE [6])
- Gateway required for internet access (transferring data to the Internet, another BLE device that has an IP connection in required [7])

One critical component in completing this thesis is the data package structure, which defines how sensor data is organized, encoded, and transmitted between the BLE module and the application.

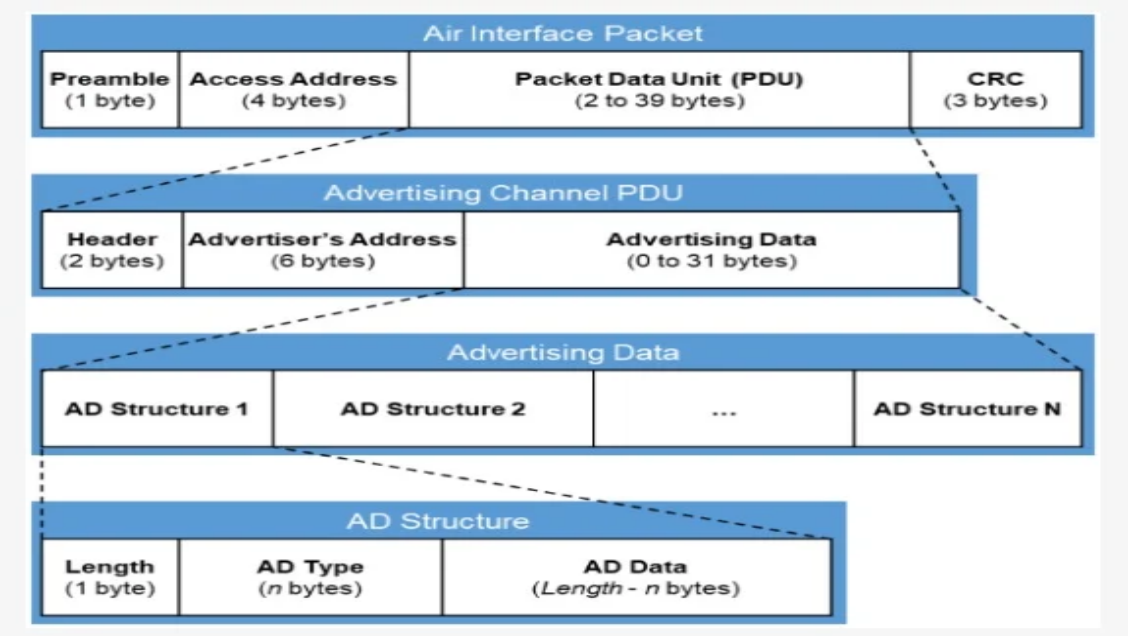

Figure 2.3: Structure of a Bluetooth data package [11]

As the picture above shows, each packet contains a byte for Preamble(used for frequency synchronization [4]), Access Address and CRC(cyclic redundancy check used for error checking), then depending on the size, there is a packet data unit(PDU) of 2 to 39 bytes. The PDU consists of a header and the sender address which share a byte and the advertising data of 0 to 31 bytes. The PDU is made of one or more structures of advertising data (length + advertising data type + advertising data).

In modern TPMS systems, tire sensors often use Bluetooth Low Energy (BLE) to transmit pressure and temperature data wirelessly. These BLE messages are received by a gateway module that serves as a bridge between BLE and the vehicle's internal CAN network. Once decoded, the gateway translates this information into CAN messages and distributes them to other ECUs that need it, such as the dashboard or body control module.

Chapter 3

System arhitecture

3.1 General Overview

The architecture of the system is centered around the acquisition, decoding, and visualization of sensor data transmitted via Bluetooth Low Energy (BLE). The data originates from embedded BLE-enabled sensors integrated into the vehicle's subsystems, such as the Tire Pressure Monitoring System (TPMS). These sensors periodically broadcast packets containing pressure, temperature, and diagnostic data.

A BLE-capable receiver module within the vehicle acts as a gateway, continuously scanning for incoming advertisements. Upon reception, the gateway decodes the BLE payload and encapsulates the relevant sensor information into Controller Area Network (CAN) frames, which are then transmitted across the vehicle's internal communication bus.

The Windows Forms application, developed as part of this project, connects wirelessly to the BLE receiver and acquires data in real-time. It parses the received information, applies interpretation logic according to the predefined data package structure, and presents it through an intuitive graphical user interface (GUI). This architecture enables developers to monitor the state of the vehicle after extended automated testing cycles without relying on manual data inspection, thus improving analysis efficiency and overall development workflow.

3.2 Hardware Components

The hardware components of the system are essential for data acquisition, communication, and integration with the vehicle's existing infrastructure. This section describes the physical devices involved in the architecture, including the BLE sensor module, the receiving gateway, and the computing platform used for running the application.

Each component plays a critical role in ensuring that sensor data is accurately collected, transmitted, and processed. The hardware ecosystem contains the following components:

BLE TPMS Sensor

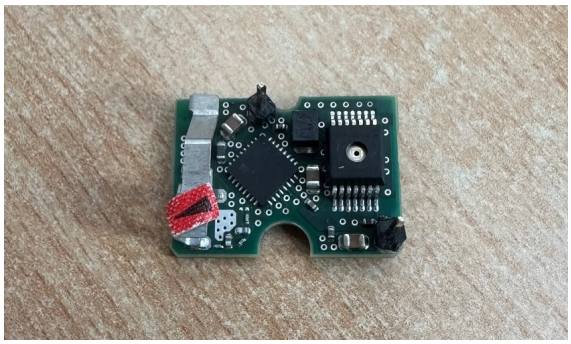

Figure 3.1: TPMS sensor module

The sensor assembly in this project consists of four primary hardware components: the ATIC (SP49-21-11), a BLE communication module, a PCB (printed circuit board) and a battery.

ANUM (Antenna Network Unit Module)

The ANUM is a BLE receiver module that acts as a gateway between the sensor and the vehicle's internal communication system. It is responsible for receiving the BLE advertisements from the sensor, decoding the data, and transmitting it over the Controller Area Network (CAN) bus.

ATIC (SP49-21-11) Module

The SP49 provides a very high level of integration, and is optimized to perform all of the functions necessary to implement a Tire Pressure Monitoring System (TPMS) sensor module.With its integrated micro controller, senors and convenient peripherals, the SP49 needs only a few external components to implement a complete TPMS sensor module. The device has been designed for lowest charge consumption making it ideal for battery powered applications.

This makes the SP49 an excellent choice for BLE applications, as its low power consumption ensures extended battery life, which is critical for wireless sensor modules. Additionally, its high level of integration reduces the need for external components, simplifying the design and minimizing the overall size and cost of the device. These features make it particularly well-suited for compact and energy-efficient BLE-based systems like TPMS.

Figure 3.2: ATIC (Antenna Telematic Interface Controller) module

As illustrated in the figure above 3.2, the chip features multiple pins, each designated for specific functionalities. The integration of these pins facilitates the efficient operation of the SP49, ensuring accurate data acquisition and reliable transmission within the TPMS sensor module.

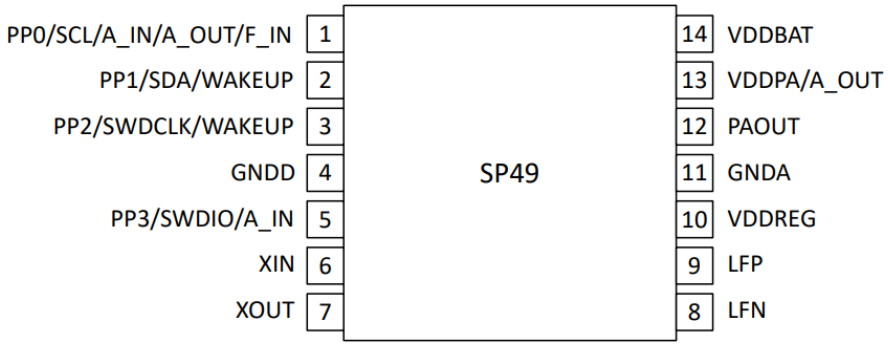

Figure 3.3: Description of the ATIC Chip pins

Figure 3.2 illustrates the pinout of the SP49 chip. The chip has a total of 14 pins, each serving a specific function:

- **Pin 1 (PP0/SCL/A_IN/A_OUT/F_IN)**: General-purpose I/O pin with multiple functions, including I^2C clock (SCL), analog input/output, sensor signal input, or external frequency reference input;
- **Pin 2 (PP1/SDA/WAKEUP)**: General-purpose I/O pin that can serve as I^2C data (SDA) or as an external wake-up source;
- **Pin 3 (PP2/SWDCLK/WAKEUP)**: General-purpose I/O used for the Serial Wire Debug (SWD) clock line or as another external wake-up input;
- **Pin 4 (GNDD)**: Digital ground reference for the internal digital circuitry;
- **Pin 5 (PP3/SWDIO/A_IN)**: General-purpose I/O line used for Serial Wire Debug data or as an analog input for the internal ADC;
- **Pin 6 (XIN)**: Crystal oscillator input pin used for the external clock source;
- **Pin 7 (XOUT)**: Crystal oscillator output pin to complete the oscillator circuit;
- **Pin 8 (LFN)**: Low-frequency (LF) receiver input, typically used for low-power wake-up signaling;
- **Pin 9 (LFP)**: Another low-frequency receiver input paired with LFN;
- **Pin 10 (VDDREG)**: Internal regulated power supply output used to power internal components;
- **Pin 11 (GNDA)**: Analog and RF power amplifier ground reference;
- **Pin 12 (PAOUT)**: Output pin for the RF power amplifier, used for signal transmission;

- **Pin 13 (VDDPA/A_OUT)**: Regulated power supply input for the RF power amplifier, also serves as an analog output for pressure or accelerometer data;
- **Pin 14 (VDDBAT)**: Primary power supply input, typically connected to a coin cell battery.

BLE Communication Module

The BLE communication module is responsible for wirelessly transmitting the sensor data collected by the ATIC to the central receiver or gateway. It operates in the 2.4 GHz ISM band and is specifically chosen for its low power consumption, small footprint, and compatibility with embedded systems.

In this project, the BLE module is integrated on the same PCB as the ATIC, forming a compact and efficient sensor unit. Once the ATIC acquires the environmental parameters (such as pressure and temperature), the processed data is handed over to the BLE chip via a digital interface. The BLE chip then packages this data into advertising packets, which are broadcast periodically.

The broadcasted data can be received by a BLE-enabled gateway or diagnostic tool within the vehicle, enabling real-time monitoring and analysis without physical connections. This wireless transmission method ensures minimal power draw and supports extended battery life—an essential requirement for tire-mounted sensor systems.

CySmart BLE Dongle

Additionally for test purposes, the CySmart (a BLE host emulation tool with a BLE dongle) was used. This tool allows developers to simulate a BLE peripheral device, enabling them to test the functionality of their applications without needing the actual hardware. The CySmart tool can be used to send and receive BLE packets, making it an invaluable resource for debugging and validating the communication between the sensor module and the application.

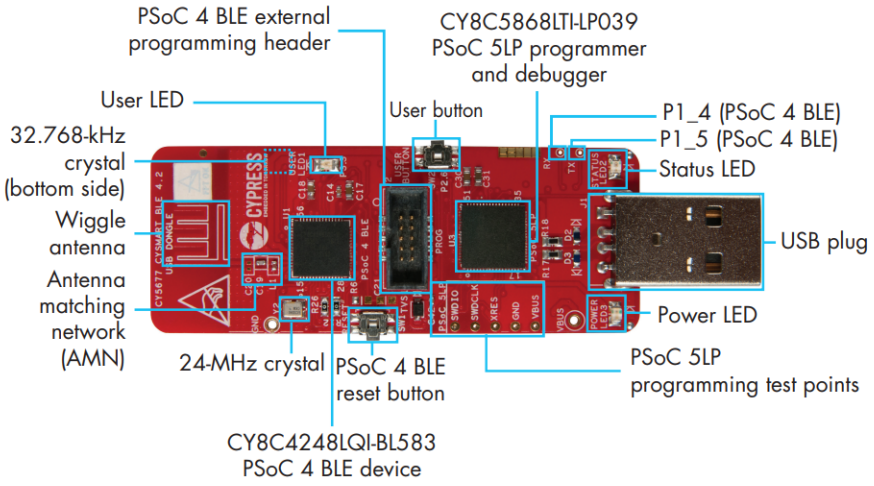

Figure 3.4: CySmart BLE dongle [5]

3.3 Software Components

The Softwares used for this project are the following:

.NET

The .NET framework is a software platform for developing Windows Form applications utilized in this project. It provides the runtime environment, a plentiful set of libraries and tools for graphical user interface (GUI) and BLE communication handling.

.NET has the following design points:

- Productivity: .NET provides a complete development ecosystem, integrating the runtime, libraries, programming language features, and development tools to streamline and enhance the developer experience;
- Safe Code: The platform emphasizes safety by default, with managed code execution, garbage collection, and type safety. However, it also allows users to write unsafe code when necessary, leaving space for the human factor in the development process;
- Static and Dynamic Typing: .NET supports both static and dynamic typing, allowing developers to choose the most suitable approach for their applications. This flexibility enables the use of dynamic features while maintaining the benefits of static typing when needed;
- Native code interop and hardware intrinsics: .NET allows efficient integration with native APIs and supports hardware instructions through low-overhead mechanisms. This enables developers to leverage platform-specific features and optimizations while maintaining the benefits of managed code;
- Cross-platform: Applications developed in .NET can run on multiple operating systems and hardware architectures. The framework also supports platform-specific optimization through targeted builds.

Visual Studio

Visual Studio is an integrated development environment (IDE) from Microsoft that provides a comprehensive set of tools for developing applications across various platforms. It supports multiple programming languages, including C#, C++, and Visual Basic, and offers features such as code editing, debugging, and project management.

The IDE chosen for this project was based on preferences and familiarity with the .NET framework. Besides the considerable capabilities of debugging, code completion, and project management, Visual Studio also provides a user-friendly interface which is very easy to work with. The extensions and plugins available make the environment very easy to customize, allowing developers to tailor the IDE to their specific needs.

PinGenerator

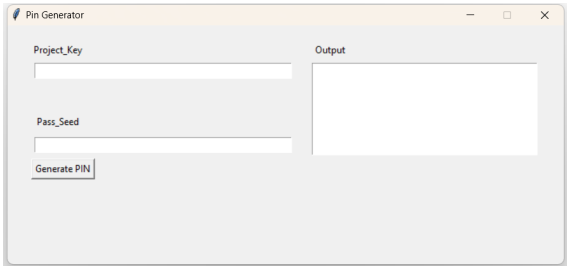

Figure 3.5: PinGenerator interface

Bluetooth pairing with the sensor module is facilitated through a unique pass seed value assigned to each sensor. This pass seed serves as an input to a sophisticated algorithm within the PinGenerator software, which processes the seed to generate a unique PIN code specific to the corresponding sensor module. This security mechanism ensures that only authorized devices possessing the correct PIN can establish a Bluetooth connection with the sensor, thereby safeguarding the sensor data from unauthorized access. The pass seed is typically embedded into the sensor during the manufacturing process, ensuring that each sensor's PIN remains permanently unique.

GeneMod

GeneMod is a software tool used for generating and managing the firmware for the ATIC module. It provides a user-friendly interface for configuring the sensor parameters, such as sampling rates, data formats, and communication settings. GENEMOD also allows developers to simulate the sensor behavior and test the firmware before deploying it to the actual hardware.

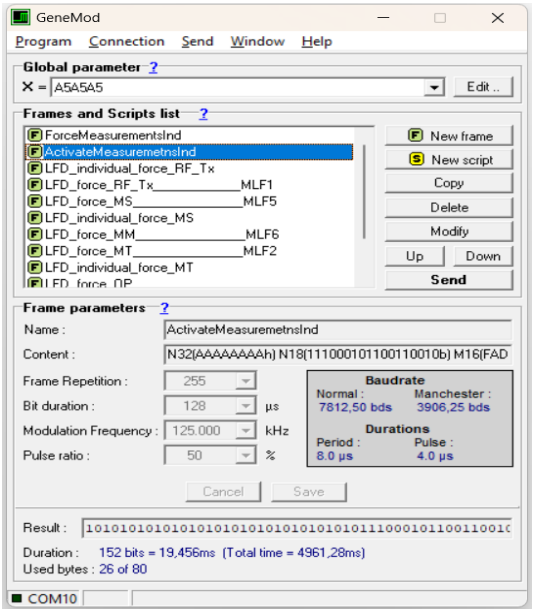

Figure 3.6: GeneMod interface

This tools is essential in the testing phase of the sensor module, as it enables developers to validate the firmware functionality and ensure that the sensor operates correctly in various scenarios without setting it up into a tire. This also helps in debugging and troubleshooting any issues that may arise during the development process. GeneMod is a tool designed to enable developers to send commands to the ATIC module via the ANUM. Through this mechanism, the operational state of the sensor can be modified—such as switching between drive mode, parking mode, or other predefined configurations.

3.4 System Architecture

For a better understanding of the system architecture, the following sequence diagram illustrates the data flow and communication between the various components involved in the system. The diagram outlines the interactions between the user, the BLE sensor, the ANUM module, the GeneMode tool and the Windows Forms application.

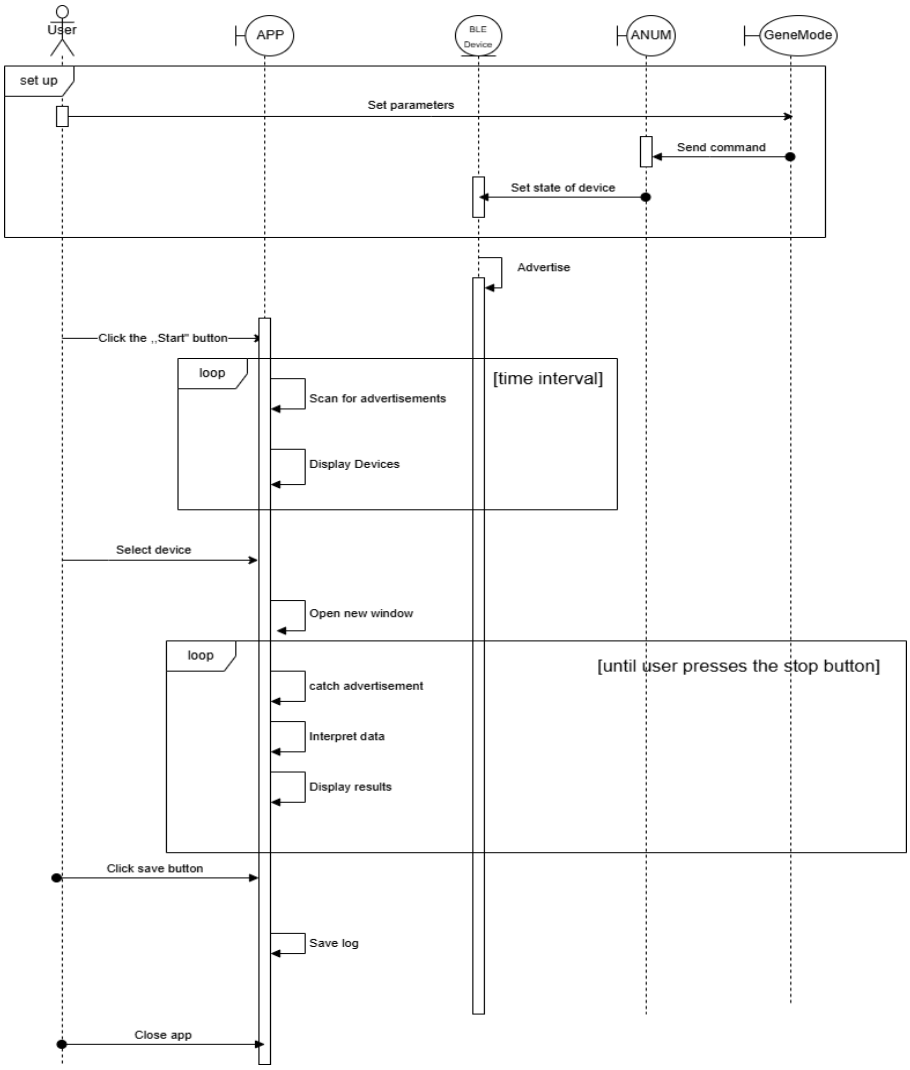

Figure 3.7: System architecture sequence diagram

The user needs to set first the parameters of the GeneMode tool, in order to send the data to the sensor module, through the ANUM device. When the state is set, the sensor will start advertising and that is when the Windows Forms application will start scanning for the BLE packets. When an advertisement packet is received, the device that sent it will be displayed in the device list. The user can then select the device and a new window will open.

The new window is called "BurstAnalyzerForm" and it is meant to display data. The following information can be found in the BurstAnalyzerForm: the device name, the MAC address, the temperature, pressure and speed (if available), the Bluetooth packages received and the state of the car. In this form, the user can also choose to stop or start the scanning process, as well as connect or disconnect, pair or unpair device. The user can save the data in a txt file, in order to save it for later analysis. The data is saved in a specific format, which can be easily interpreted.

3.5 Data Flow and Communication

The data flow and communication between the various components of the system are crucial for ensuring that the sensor data is accurately collected, transmitted, and processed. The following diagram describe the data flow and communication protocols used in the system.

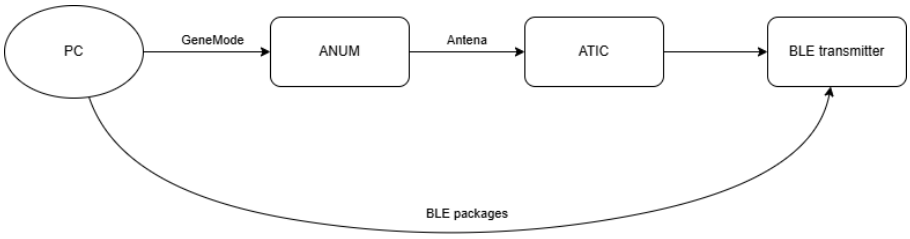

Figure 3.8: Data flow diagram

In this project, the data flow begins with the PC that uses the GeneMod tool to set the parameters for the ANUM to transmit. The ANUM then sends the data to the ATIC chip, which is responsible for collecting the sensor data. The ATIC chip processes the data and sends it to the BLE transmitter. Then the data is transmitted wirelessly to the PC's Bluetooth receiver and gets interpreted by the application.

For a more detailed understanding of the system's behavior another diagram is presented below, which illustrates the data flow in detail.

Figure 3.9: Detailed data flow diagram

The diagram above 3.9 illustrates the detailed data flow within the system. As the diagram shows, the data flow begins with the GeneMod tool, which is used to set the parameters for the ANUM. The ANUM then sends the data to the Sensors BLE receiver chip, which is responsible for collecting the sensor data. The ATIC

chip processes the data and starts advertising the data. The application then scans for the BLE packets and collects the data. When the data is collected, it is analyzed and interpreted by the application. The data is then displayed in the user interface, where the user can see the sensor data and the state of the car.

Chapter 4

Implementation Details

4.1 BLE Communication in C#

The implementation of BLE (Bluetooth Low Energy) communication in C# involves establishing a connection with BLE devices, exchanging data, and managing the communication process. This section provides an overview of the key components and steps required to achieve seamless BLE communication.

We will explore the libraries and tools used, the process of discovering BLE devices, connecting to them, and handling data transmission. Additionally, we will discuss the challenges encountered during the implementation and the solutions adopted to address them.

First of all, we need to set up the development environment. This includes installing the necessary libraries and tools for BLE communication in C#. The most commonly used library for this purpose is the Windows.Devices.Bluetooth namespace, which provides a comprehensive set of APIs for working with Bluetooth devices [2]. Some of the classes delivered by this namespace include:

- **BluetoothLEDevice**: a BluetoothLEDevice object represents a Bluetooth Low Energy device and provides methods for connecting to it, retrieving its properties, and managing its services;
- **BluetoothDeviceId**: a BluetoothDeviceId object represents the unique identifier of a Bluetooth device. It is used to identify and connect to specific devices;
- **BluetoothConnectionStatus**: a BluetoothConnectionStatus enumeration represents the connection status of a Bluetooth device. It can be used to check if a device is connected, disconnected, or in the process of connecting.

Bluetooth Low Energy (BLE) communication in C# is implemented using the Windows.Devices.Bluetooth and Windows.Devices.Bluetooth.Advertisement namespaces, which are part of the Windows Runtime (WinRT) API set. These APIs provide the necessary tools to scan for BLE advertisements, connect to peripheral devices, and interact with their services and characteristics.

The communication process typically begins with the BluetoothLEAdvertisementWatcher class, which enables the application to listen for BLE advertisement packets broadcasted by nearby devices. The watcher can be configured with custom filters through its AdvertisementFilter property or by subscribing to specific

manufacturer data, UUIDs, or device names. This allows the application to limit the scan to relevant devices, improving efficiency and reducing interference from unrelated broadcasts.

When an advertisement is detected, the Received event is triggered, and the application can access detailed information about the broadcast through the BluetoothLEAdvertisementReceivedEventArgs object. This includes properties such as BluetoothAddress, signal strength in dBm, and the actual advertisement data structure. The device's address serves as a unique 64-bit identifier used to differentiate between BLE peripherals within range.

After identifying a device of interest, a connection can be initiated by creating a BluetoothLEDevice object using the FromBluetoothAddressAsync method. This object represents the remote device and provides access to its basic properties, such as name, device ID, and connection status, as well as its services and characteristics. Through the GetGattServicesAsync method, the application can enumerate the device's GATT (Generic Attribute Profile) services, and for each service, GetCharacteristicsAsync allows access to its associated characteristics.

Each GATT characteristic is represented by a GattCharacteristic object, which exposes methods for reading and writing data (e.g., ReadValueAsync and WriteValueAsync) and subscribing to notifications. These characteristics are defined by UUIDs and often encapsulate sensor readings or control commands. Data exchanged through characteristics is typically represented as binary buffers, which are decoded using standard byte manipulation techniques such as DataReader or BitConverter.

The BLE communication model in C# is inherently asynchronous, using the async/await pattern to ensure non-blocking operations. This design allows BLE interactions to run smoothly alongside the graphical user interface, maintaining responsiveness and enabling real-time updates. Robust error handling and connection status checks are essential when managing devices that may frequently connect and disconnect in dynamic environments.

4.2 Data Processing

As discussed in the previous section 4.1, the data received from the BLE device is in the form of a byte array. This data must be processed in order to extract the meaningful information. The processing involves converting the byte array into a more usable format, such as a string. This is done in the following manner:

```
1     var dataArray = new byte[data.Length];
2     using (var reader = DataReader.FromBuffer(data)) {
3         reader.ReadBytes(dataArray);
4     }
5     string dataHexString = BitConverter.ToString(dataArray);
```

Listing 4.1: Reading and converting BLE data to string

Before explaining the identifying process, the structures need to be explained. The information regarding these structures was obtained from internal requirement documentation provided by the development team. Due to confidentiality constraints, this document cannot be included or published . There are three important setups:

- Data Setup
- Scan Response and Advertisement structure
- Payload

Each Data Setup is composed by one or more advertisements/scan responses and each Advertisement structure contains the length, type, payload and occasionally the company id. For example,the structure `SHORT_ROLL` , which is a setup advertised in bursts of 3 frames, each frame containing 2 structures.

Table 4.1: Structure of `RF_ADV_PER_SHORT_ROLL`

PDU_type	`TPMS_SIGN + APPEARANCE`
Burst_number	Infinite
Burst_interval	X seconds
Frame_number	X
RSP_type	DEV_NAME

Note: The detailed structure presented above is based on internal documentation and is included here strictly for technical illustration.

Table 4.2: Structure of the `S_AD_TPMS_SIGN` field

Field name	**Size**	**Description**
PDU length	X bits	X + Length of Payload
AD type	X bits	`NC_AD_TYPE_MANUF`
Company ID	X bits	Company Identifier
Payload	variable	`C_BT_TYPE_STD_AUTH`

Note: The detailed structure presented above is based on internal documentation and is included here strictly for technical illustration.

Table 4.3: Structure of the `S_AD_APPEARANCE` field

Field name	**Size**	**Description**
PDU length	X bits	X
AD type	X bits	NC_AD_TYPE_APPEARANCE
Payload	X bits	NC_CODE_TPMS_SENSOR

Note: The detailed structure presented above is based on internal documentation and is included here strictly for technical illustration.

The payloads represent the most critical part of the transmitted data, as they contain the essential information. This is done by a special data transmission protocol which uses a specialized field structure to efficiently package and transmit sensor telemetry and device status information. Although the precise bit-level layout is proprietary, its functional design can be described.

The structure begins with header information, including compliance flags and a unique frame identifier for packet management. This is followed by data specifying the wireless transmission parameters, such as the communication channels being used for the broadcast.

The core of the data payload is dedicated to sensor readings. It is designed to carry telemetry from multiple environmental and motion sensors, including measurements for acceleration, atmospheric pressure, and temperature. The raw sensor data is encoded into a compact format, with the protocol defining the valid operational ranges and specifying distinct values to indicate sensor errors or out-of-range measurements.

To ensure reliability and system awareness, the structure incorporates several layers of diagnostic and integrity checks. A liveness counter increments with each transmission cycle, providing a constant confirmation of the device's operational state. Furthermore, a checksum field is included, calculated from the critical sensor and status data, to guarantee data integrity upon receipt. The final section of the field consists of a series of status flags, providing a consolidated, at-a-glance summary of the system's health. These flags indicate the current operational mode, the validity of the internal system clock, the presence of major errors, specific alerts related to sensor readings, and the device's battery status.

In code, an example would look something like this:

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct C_BT_TYPE_STD_P {
    public ushort CompanyID;
    public byte BeaconFlag_FrameID;
    public ushort ChannelUsed_Acceleration;
    public ushort Pressure;
    public byte Temperature;
    public byte PressureCritical_AliveCounter;
    public byte Checksum;
    public byte TPMSmode_RTCstatus_ErrorStatus
    _PressureAlert_BatteryStatus;

}
```

Listing 4.2: Example of structure definition

The use of `StructLayout` attribute with `LayoutKind.Sequential` ensures that the fields are laid out in memory in the order they are defined. The `Pack = 1` option specifies that the fields should be packed with a one-byte alignment, which is important for ensuring that the data structure matches the expected format of the BLE advertisement.

Different value types are use to store data, such as `byte`, `ushort`, and `int`. The `byte` type is used for single-byte values, while the `ushort` type is used for two-byte

values and `int` type is used for four-byte values. This way the parsing process of the data is easier and more efficient.

The structures have methods that allow for easy access to the individual fields. Manipulating the bits in the `ushort` or `int` data types is done using bitwise operations. For example:

```
1    public bool GetPressureCritical() {
2        return (PressureCritical_AliveCounter & 0x80) != 0;
3    }
```

Listing 4.3: GetPressureCritical method

In the example above, the battery status is extracted using a bitwise AND operation with the value `0x01`. This operation isolates the least significant bit (LSB) of the byte named `TPMSmode_RTCstatus_ErrorStatus_PressureAlert_BatteryStatus`. The LSB represents the binary flag for the battery status, where `1` indicates a low battery condition and `0` indicates a normal battery state.

This approach is a common technique when working with compact status bytes, where each bit holds a distinct piece of information [8]. In this specific case, the LSB is used, while in other methods (e.g., pressure critical or TPMS mode), the most significant bit (MSB) is used to encode additional flags or state indicators.

```
1    public bool GetBatteryStatus() {
2        return (TPMSmode_RTCstatus_ErrorStatus_
3        PressureAlert_BatteryStatus & 0x01) != 0;
4    }
```

Listing 4.4: GetBatteryStatus method

Some methods have a specific algorithm in order to get the correct values. As the Figure **??** shows, the `Pressure` field is a 16-bit value that represents the pressure relative to 100 kPa. To convert this value into a more understandable format, we need to divide it by 16. This is done in the following method:

```
1    public double GetPressure() {
2        if (Pressure == 0xFFFF) {
3            return double.NaN; \\check for error
4        }
5        return 100.0 + (Pressure / 16.0);
6    }
```

Listing 4.5: GetPressure method

The project include a class for each structure and setup that is used in the communication.

4.3 Error Handling and Debugging

In this thesis, the way the errors are handled is by the `try-catch` and `try-catch-finally` statements. The `try-catch` statement is used to handle errors that might occur during the execution of the code inside the try block, and the `try-catch-finally` is used to ensure that cleanup or finalization code is always executed, regardless of whether an exception was thrown or not. This approach

guarantees that critical operations such as releasing resources, closing connections, or stopping background processes are reliably completed, even in the presence of runtime errors [3].

One specific example would be the following:

```
private async void disconnect_button_Click(object sender,
    EventArgs e) {
     try {
          sensor.Dispose();
          status_textbox.AppendText("Disconnecting from device
              ...\n");
          Cursor = Cursors.WaitCursor;
          await WaitForDisconnectionAsync();
          if (sensor.ConnectionStatus ==
              BluetoothConnectionStatus.Disconnected) {
               status_textbox.AppendText("Disconnected from
                  device.\n");
          }
          else {
               status_textbox.AppendText("Disconnection failed.\
                  n");
          }
     }
     catch (Exception ex) {
          status_textbox.AppendText($"Error: {ex.Message}\n");
     }
     finally {
          Cursor = Cursors.Default;
     }
}
```

Listing 4.6: Error handling

In the code above 4.6, because the `Dispose` function is dangerous from a error throwing point of view, the statement `try-catch-finally` is used. Because the code from the try block can throw different errors, in the catch block the errors are displayed in order for the user to see exactly what the problem is. After the process is done, regardless of whether the process completes successfully or encounters an error, the code within the `finally` block is always executed, which in this case is turning the cursor format from loading to default.

Chapter 5

Testing and validation

5.1 Testing and Validation

The test process typically involves initializing the application, triggering a specific LF signal using the ANUM, observing the BLE advertisement, and verifying the parsed payloads against expected values. In the same requirements document referenced in Section 4.2, the expected responses to each LF (low frequency) signal are specified. This allows the received values from the BLE device to be compared against the predefined expectations for each field.

For example, when the ANUM is used to transmit a signal indicating that the vehicle is in motion, the sensor module transitions to drive mode. In this state, the sensor is expected to broadcast an advertisement every 4 seconds, containing two payloads: `TPMS_SIGN` and `APPEARANCE`. If the displayed values match the expected values, the test is considered successful. If there are discrepancies and the test fails, then the specific field that caused the failure is identified and corrected.

Through this structured approach, both expected and edge-case behaviors of the sensor can be validated. The application enables rapid testing and debugging by providing immediate feedback when field values deviate from the specification, supporting faster iteration during development.

5.2 Performance Metrics and Optimization

One of the most critical aspects of this system is the reliability and accuracy of both LF and BLE signals. The entire system acquisition and processing chain is based on the assumption that the signals are received correctly. If even a single BLE advertisement is missed or an LF trigger is not interpreted correctly by the sensor, the system may fail to detect a mode transition or receive incomplete data.

One of the hardware change that was made to the sensor module is changing its coil to a bigger one. This modification was made to increase the sensitivity of the sensor module to LF signals, therefore insuring that the sensor changes its mode when the LF signal is received from the ANUM.

Another hardware addition was a new antenna for the bluetooth adaptor that the laptop uses. This antenna is connected to the laptop via a dongle and it is used to increase the sensitivity of the bluetooth signal. In this manner, the laptop is able to receive the BLE advertisement from the sensor module smoothly.

Software-wise, a special function was created. Listing 5.1 defines the LE_SCAN_REQUEST structure, which is used to configure Bluetooth Low Energy (BLE) scanning behavior at a low level. The structure includes fields for specifying the scan type (active or passive), the scan interval, and the scan window. These parameters are expressed in 0.625 ms units and are critical for adjusting the timing and energy consumption of the scan process.

```
[StructLayout(LayoutKind.Sequential)]
private struct LE_SCAN_REQUEST {
    internal int scanType;
    internal ushort scanInterval;
    internal ushort scanWindow;
}
```

Listing 5.1: `LE_SCAN_REQUEST` structure for configuring BLE scan parameters

Listing 5.2 presents the method responsible for initiating a custom BLE scan using low-level Windows APIs. The method first acquires a handle to the system's first Bluetooth radio using BluetoothFindFirstRadio, then constructs a LE_SCAN_REQUEST instance with the desired scan parameters (using the value 29 for both `scanInterval` and `scanWindow` makes the scanning time 5 times more frequent). These parameters are passed to the system driver via a DeviceIoControl call using a custom I/O control code. To prevent blocking the main thread, the scan is started asynchronously in a dedicated background thread. This implementation enables more precise control over BLE scanning compared to high-level managed libraries.

```
public static void StartScanner(int scanType, ushort
   scanInterval, ushort scanWindow) {
    var thread = new Thread(() => {
        BLUETOOTH_FIND_RADIO_PARAM param = new
           BLUETOOTH_FIND_RADIO_PARAM();
        param.Initialize();
        IntPtr handle;
        BluetoothFindFirstRadio(ref param, out handle);

        uint outsize;
        LE_SCAN_REQUEST req = new LE_SCAN_REQUEST {
            scanType = scanType,
            scanInterval = scanInterval,
            scanWindow = scanWindow
        };

        DeviceIoControl(handle, 0x41118c, ref req, 8, IntPtr.
           Zero, 0, out outsize, IntPtr.Zero);
    });
    thread.Start();
}
```

Listing 5.2: Low-level BLE scan initialization using `DeviceIoControl`

5.3 Developer Feedback

Working on this project has truly felt like a roller coaster — an unpredictable, thrilling ride through the intertwined worlds of hardware and software. There were moments of genuine excitement, quickly followed by deep frustration and trial-and-error loops that tested both my patience and my perseverance. From the very beginning, it was clear that this wasn't going to be a simple software assignment. This was an embedded system — a living, breathing mix of signals, sensors, timings, and tolerances.

One of the most unexpectedly difficult challenges was positioning the sensor module correctly in relation to the antenna. Even a slight misalignment would cause the sensor to ignore the LF signals completely, throwing off the entire validation flow. It was a surprisingly sensitive setup — something you don't read about in specifications, but only learn through hours of hands-on testing. The laptop itself became a source of resistance. Long BLE sniffing sessions and repeated scanning processes caused it to overheat more than once, reminding me that the physical limitations of the tools we use are just as real as the bugs in our code.

Then there was the ANUM — a tool that, while functional, felt like it belonged to another era. Its interface was clunky, and its feedback minimal. With each successful mode switch it triggered, I began to understand its value, not just technically but also as a symbol of the broader system I was interfacing with — a real-world automotive stack with legacy tools and evolving technologies coexisting.

Throughout this journey, I quickly learned that asking for help wasn't a weakness, but a necessity. Whether it was a question about how the ANUM worked, or understanding why the BLE payload looked the way it did, I often turned to my colleagues in the office. Their guidance, patience, and willingness to explain even the smallest details played a crucial role in moving the project forward. I came to rely on their experience, and those moments of collaboration reminded me that engineering is rarely a solo effort — it's built on shared knowledge and teamwork.

Yet, despite — or perhaps because of — all these hurdles, the satisfaction I felt when the system worked as intended was unparalleled. Seeing a properly structured BLE advertisement pop up on my screen, knowing it came from a real sensor triggered by a physical LF pulse, and watching my application interpret it in real time, felt incredibly rewarding. Every obstacle made those moments of success ten times more meaningful. It wasn't just about writing code or wiring hardware; it was about creating something that lives in the space between them — and making them speak the same language. This experience didn't just teach me technical concepts; it taught me how to troubleshoot, adapt, persist, communicate, ask for help and — most importantly — celebrate the small victories that come after real effort.

Chapter 6

Usage Guide

6.1 Set Up

The system setup for each test session requires several steps to be followed in order to ensure proper signal transmission, reception and interpretation. This process is described in the following steps:

1. Connect the external BLE antenna to the laptop via USB
2. Power the sensor on by connecting the battery to the sensors pins. When connecting the battery, the user must be careful to connect the positive and negative terminals correctly. A simple way to remember the correct wiring is to note that the antenna is positioned near the negative terminal, which, in this case, corresponds to the blue wire.
3. Plug in the ANUM device to the laptop via USB.
4. Open the GeneMode and select the correct COM port for the ANUM device. Select the parameters for the test session and start sending LF packets.
5. Start the SensorView application on the laptop and start the scanning process. Make sure the bluetooth is enabled on the laptop. When the name of the device appears in the list, select it and start the data analysis process.
6. Connect/Disconnect the device as needed. In cases where a PIN is needed to complete the pairing process, the relevant considerations and implementation details have been discussed in previous sections 3.3.
7. When the process is over, the data can be saved locally.

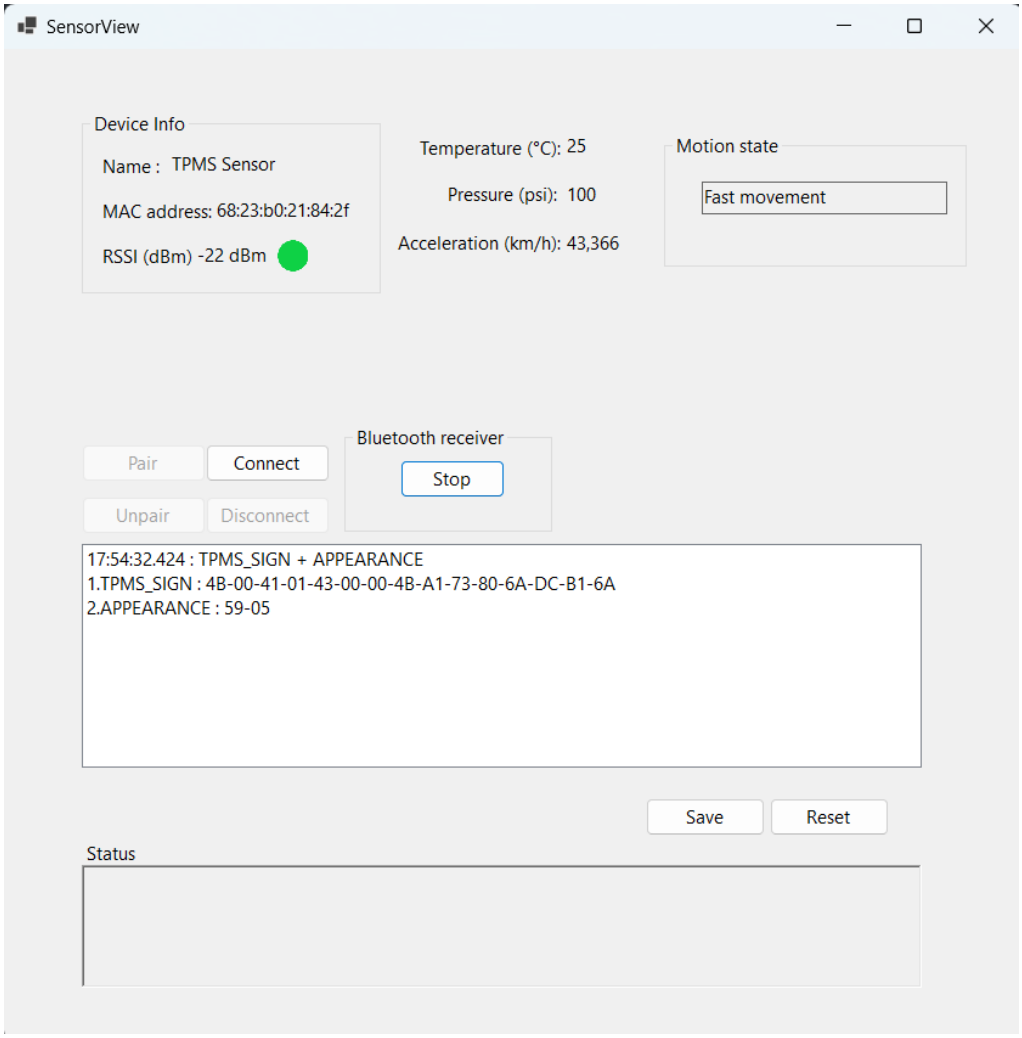

Figure 6.1: User interface of the SensorView application

Figure 6.1 displays the interface of the application *SensorView*. A key feature of this interface is the inclusion of the Received Signal Strength Indicator (RSSI), shown alongside the device name and MAC address. The RSSI value, expressed in dBm, provides immediate insight into the strength and reliability of the Bluetooth connection. An indicator is used to visually represent the strength of the signal (red for weak, yellow for moderate, and green for strong), allowing users to quickly assess the connection quality. This is very useful for troubleshooting lost packets or connection issues, as it helps in identifying whether the problem lies with the signal strength or other factors.

The upper section of the interface also includes fields for displaying decoded sensor values such as temperature in °C, pressure in psi, acceleration in hm/h, and motion state. These values are populated dynamically as valid data is received from the BLE device. The user can start or stop the Bluetooth receiver using the controls in the middle section.

The console area logs all incoming data, including timestamps and raw or partially decoded payloads, helping developers trace and interpret the structure of each received advertisement. Lower, controls for saving or resetting the session allow for organized testing and review. At the bottom, the status box displays the states that the application is going through, such as scanning, connected, disconnected,

saving and so on. This provides a clear overview of the application's current state and helps users understand the flow of operations.

6.2 Troubleshooting

In 6.1, the steps for setting up the system are described. However, there are some common issues that can arise during this process. The next sections will describe these issues and how to resolve them. Here are some common issues that can arise during the setup process and their solutions:

- The ANUM can be assigned a port number bigger than 13, which is the maximum on the GeneMode application. In this case, the user must change the port number manually from Device Manager. To do this, the user must open Device Manager, select the ANUM device, right-click and select Properties. In the Properties window, select Port Settings, then Advanced. In the Advanced Settings window, select the COM Port Number in the drop-down menu that suites best.

- The ANUM device does not appear at all in GeneMode. In this case, the user must check if the ANUM device has the propper drivers installed. To do this, the user must open Device Manager and check if the ANUM device appears in the list. If it does not appear, the user must install the drivers by right-clicking on the device and selecting Update Driver, then 'Browse my computer for drivers', then 'Let me pick from a list of available drivers on my computer'. In the list, select the driver version `3.3.17.203`. The driver will be installed and the ANUM device should appear in the list.

- If the sensors name does not appear in the list of devices in the SensorView application, the state of the sensor must be checked. A Troubleshooting session would need to be performed. First check if the battery is connected properly (care for the positive and negative terminals). If the battery is connected and the sensor is still not visible, check the advertising using the CySmart 3.2. If the sensor is visible in CySmart, then the problem is with the SensorView application. If the sensor is not visible in CySmart, then the problem is with the sensor. In this case, the user must check if the sensor got the right LF packets from the ANUM device (this can be solved by ensuring proper alignment between the sensor and the antenna).

The troubleshooting process is a necessity in this project because the setup is very sensible and everything is based on the accuracy. It is important to understand what could go wrong in the process and how to fix it. Documenting known limitations and their corresponding fixes not only improves system reliability but also provides a clear path for future maintenance and optimization.

Chapter 7

Conclusion

7.1 Summary and findings

The purpose of this project was to design and implement a desktop application capable of receiving, interpreting, and displaying BLE advertisement data from a sensor module used in automotive testing scenarios. The solution was developed in C#, using a Windows Forms interface and a low-level Bluetooth scanning mechanism to ensure higher precision and reduced packet loss compared to high-level APIs.

The project explored both hardware and software aspects, including the impact of antenna placement, BLE sensitivity, and LF signal accuracy on the overall reliability of the system. A custom structure was used to parse BLE packets and extract fields such as pressure, temperature, and battery status in real time. Furthermore, the application supported developer needs by offering immediate feedback, customizable scanning parameters, and structured data visualization.

Key findings include the importance of low-level BLE scanning via DeviceIoControl, the sensitivity of sensor response to LF signal alignment, and the challenges of integrating legacy tools like the ANUM. Despite these complexities, the system achieved its intended goal: it provided a working, testable environment for monitoring BLE sensor states accurately and efficiently.

7.2 Limitations

In the domain of Radio Frequency (RF) communication, accuracy is a crucial requirement. The effectiveness of any BLE based application is totally influenced by the ability of reliably receiving and interpreting the BLE advertisement packets. When signal reception is not 100% accurate, it introduces uncertainty that complicates the software development process. In this case, the project development faced several problems related to the accuracy of the BLE signal reception, which were not fully resolved. The main issues included: inconsistent packet reception, missed frames and data loss due to environmental factors such as signal interference, attenuation, and general RF surrounding radio frequency activity.

This problems made it impossible to interpret the BLE advertisement packets all the way to the end. Losing frames meant losing track of the frames count which led to incorrect burst size calculations. Another issue was receiving the frames late,

which made it impossible to determine the bursts themselves because the frames time stamps were exceeding the burst threshold.

In order to solve these problems, a larger external antenna was connected to the laptop to improve the receiving range and sensitivity and the sensor module was modified by switching the LF coil with a bigger one in order to enhance the LF signal strength. Unfortunately, these changes did not fully resolve the issues, and the application still faces challenges in accurately interpreting the BLE advertisement fully.

7.3 Possible improvements

While the application performed reliably under test conditions, several improvements can be considered in future versions. First, the user interface could be enhanced to support live plotting, better error messaging, and configuration presets. Second, the application could be extended by adding a functionality to change the GATT services of the device. Third, more accuracy and better hardware would be useful in order to improve the overall performance of the system.

Additionally, expanding the application to support multiple sensor modules in parallel or integrating it with vehicle CAN diagnostics could further increase its relevance and usefulness in broader automotive test environments. Porting the tool to a cross-platform framework such as .NET MAUI or Qt could also enhance portability and adoption across different test setups.

Bibliography

[1] Dewesoft. "what is can bus?". =https://dewesoft.com/blog/what-is-can-bus, 2023. Accessed: 2025-04-01.

[2] Microsoft Documentation. Windows.devices.bluetooth namespace. https://learn.microsoft.com/en-us/uwp/api/windows.devices.bluetooth?view=winrt-26100&redirectedfrom=MSDN.

[3] Microsoft Documentation. Exception handling statements - throw, try-catch, try-finally and try-catch-finally. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/exception-handling-statements, 22-04-2023.

[4] Eric Peña Erick Reyes, Mary Grace Legaspi. Understanding the architecture of the bluetooth low energy stack. https://www.analog.com/en/resources/technical-articles/understanding-architecture-bluetooth-low-energy-stack.html, 2024-11-12.

[5] Infineon Technologies AG. Cy5677 cysmart ble 4.2 usb dongle. https://www.infineon.com/dgdl/Infineon-Quick_Start_Guide-UserManual-v01_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0efc004c11f6, 2015.

[6] Link Labs. Bluetooth vs. bluetooth low energy: What's the difference? [2023 update]. https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy, 2023-05-17.

[7] Mohammad Afaneh. "bluetooth low energy (ble): a complete guide. https://novelbits.io/bluetooth-low-energy-ble-complete-guide, September 6, 2022. Accessed: 2025-04-08.

[8] Robert Sheldon. most significant bit (msb). https://www.techtarget.com/whatis/definition/most-significant-bit-or-byte, 07-2022.

[9] TP-Link. General questions about wi-fi range of tp-link wireless product. https://www.tp-link.com/ar/support/faq/2291/, 2021-09-24.

[10] Wikipedia contributors. Bluetooth — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bluetooth&oldid=1284372619, 2025. [Online; accessed 8-April-2025].

[11] Chi Yi Lin You Wei Lin. An interactive real-time locating system based on bluetooth low-energy beacon network. https://www.mdpi.com/1424-8220/18/5/1637, 2018-05-21.