POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

NB-IoT Ultra-low power sensing platform

Supervisors

Prof. Massimo PONCINO

Davide GHEZZI

Candidate

Miloš ĐORĐEVIĆ

December 2024

Summary

This thesis details the process of creating a low-power NB-IoT development platform that incorporates sensing abilities. It showcases the development process, starting with building a basic prototype and later leading to the actual product.

The thesis includes appropriate background to better understand the design choices and terminology used when explaining the development process. It also explores the current state of the NB-IoT market to better understand this recently developed protocol and its potential. After that, it shifts focus to exploring the already existing development platforms that are similar to the one covered by this thesis. Following this, it delves deeper into describing the actual design process, starting first by building a prototype.

The documented prototyping process begins by detailing the recently developed EVLST87M01 evaluation board used for trying out the features of the ST87M01 NB-IoT + GNSS module. After that, it shows how another development board, the X-NUCLEO-U575ZI-Q, was introduced. The introduction of this board enabled testing how the STM32U575 MCU can be used together with the ST87M01 module for NB-IoT application development. Continuing with building the prototype, the X-NUCLEO-IKS4A1 board with three different sensors was introduced. This enabled testing how the temperature and humidity, barometer, and accelerometer sensors can be used. Finally, the X-NUCLEO-PGEEZ1 was added to evaluate the benefits of EEPROM memory.

After a successful prototype was built and after evaluating all of the potential components, the thesis presents the hardware development process for the new development board. It showcases how the development process kicked off with work on the schematics. It presents how an abundance of documentation and availability of schematics from previous projects can help in cutting down development time and bugs. The next step in hardware development, which was PCB design, was outsourced and not a focus of this project; therefore, it ended up not being covered by the thesis.

Once the first PCBs were available, the thesis shows the process of board bring-up and testing. After testing, it details how the firmware development started, from the BSP to the application-level firmware examples. These examples were created to better present the capabilities of the newly developed platform. One special example was developed to support the usage of the board with STMicroelectronics's data logging infrastructure, which is used for easily collecting sensor data and saving it on the PC.

Last but not least, the thesis lays out the results obtained when evaluating the performance characteristics of the new platform to get a sense of how the newly created product compares to the existing ones.

Acknowledgements

I would first like to express my gratitude towards the team of the master program EMECS that I am a part of, specifically the coordinator, Prof. Wolfgang Kunz, whose vision brought an opportunity to many students across the globe, including myself, to expand their boundaries and knowledge, as well as an opportunity to make friendships that transcend geographical borders.

Big thanks as well to the people from STMicroelectronics, SRA lab in Agrate-Colleoni, especially my supervisor, Davide Ghezzi, whose excellent mentorship and guidance enabled me to navigate the tasks and thesis work with ease.

I am also thankful for my friends, family, and loved ones who provided both distractions when needed and encouragement.

Last but not least, I would like to express my gratitude to my PoliTo supervisor, Prof. Massimo Poncino, for his guidance, feedback, and suggestions during the course of working on this thesis topic.

Table of Contents

List of Tables

List of Figures

Acronyms

NB-IoT
: Narrowband Internet of things

DTDL
: Digital Twins Definition Language

RTOS
: Real-Time Operating System

FW
: Firmware

HW
: Hardware

PCB
: Printed Circuit Board

GNSS
: Global Navigation Satellite System

GPS
: Global Positioning System

GLONASS
: romanized: Global'naya Navigatsionnaya Sputnikovaya Sistema, lit. Global Navigation Satellite System

Galileo
: Global navigation satellite system (GNSS) created by the European Union

CMSIS	Common Microcontroller Software Interface Standard
EEPROM	Electrically Erasable Programmable Read-Only Memory
ST	STMicroelectronics (the company)
STM32	Family of 32-bit microcontrollers from ST
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
MQTT	Message Queuing Telemetry Transport
LWM2M	Lightweight Machine-to-Machine
EClib	Easy Connect library
MEMS	Micro-Electromechanical Systems
PC	Personal Computer
AT	Attention (in reference to AT commands)
GUI	Graphical User Interface

UART

Universal Asynchronous Receiver-Transmitter

USART

Universal Synchronous/Asynchronous Receiver-Transmitter

ASCII

American Standard Code for Information Interchange

LPWA

Low Power Wide Area

3GPP

The 3rd Generation Partnership Project

AWS

Amazon Web Services

HAL

Hardware Abstraction Layer

ISR

Interrupt Service Routine

GPIO

General Purpose Input/Output

LPTIM

Low Power Timer

USB

Universal Serial Bus

LTE

Long Term Evolution

PNT

Positioning, Navigation, Timing

LED

Light Emitting Diode

RISC

Reduced Instruction Set

DSP

Digital Signal Processing

PSSI

Parallel Data Input/Output Slave Interface

I2C

Inter-Integrated circuit

SPI

Serial Peripheral Interface

SDMMC

Secure Digital/Multimedia Card Interface

FDCAN

Controller Area Network Flexible Data-rate

SAI

Serial Audio Interface

SoC

System-On-Chip

RTC

Real-Time Clock

ID

Identification

PSM

Power Saving Mode

eDRX	Expanded Discontinuous Reception
SRA	System Research and Applications

Chapter 1

Introduction

With the progressive improvement in mobile connectivity and mobile network technologies, humans all across the globe can get connected with one another in a matter of seconds. The main focus of mobile connectivity for a long time was human-oriented services like phone calls, Internet connectivity, messaging, etc. Nevertheless, the idea to shift the focus from human-oriented services and introduce the interconnectedness of physical devices was there from the start.

This idea slowly developed, starting from the very first physical device connected to the Internet that would give status reports and receive and interpret commands. This first device was an experiment in the 1980s with a vending machine at Carnegie Mellon University. A decade later, a successful demonstration was performed that included a toaster connected to the Internet that could be controlled, turned on, or off by sending commands to it. Fast-forward another decade, and LG released the very first smart device, a smart fridge, on the market. Around this time, the term Internet of Things (IoT) was first coined, hinting at devices (things) being connected to the Internet, which can be controlled and monitored over the network. In 2008, the first IoT conference took place. This period was also significant because the number of IoT devices surpassed the number of people on Earth. The beginning of 2010 brought a thermostat connected to the Internet that could learn the habits of the users and automatically adjust the temperature, made by the company Nest.

This steady development of technology led to today's interconnectedness of billions of devices. These devices typically have some sensors embedded into them and some actuators as well to enable them to sense the surrounding environment and perform some actions. Several different protocols exist that enable these devices to be interconnected with each other and also connected to the Internet. Usually, these protocols are limited in range and require infrastructure to be deployed separately, such as providing separate routers and nodes when using a LoRaWAN network. One relatively new protocol aims to change these limits by providing connectivity for devices that are even located indoors without the necessity of deploying separate infrastructure but taking advantage of the existing one. This protocol is known as NB-IoT.

Since NB-IoT is a relatively new protocol, there are not many solutions available on the market for finding an NB-IoT module together with a development board option for easier prototyping. The work presented in this project aims to remedy this by using the new NB-IoT module and combining it with a low-power microcontroller and sensors to create a new NB-IoT development platform. This platform can be powered by a battery or even be battery-less, relying on solar energy and supercap technology. The hardware components used were all developed by STMicroelectronics. This was intentional to avoid problems that come with parts from different sources. One of the reasons why components made by STMicroelectronics were used is because the project was developed in collaboration with STMicroelectronics, with the SRA team in Agrate-Colleoni. Several other departments from STMicroelectronics were also involved in the development of this project. One department was responsible for the development of the ST87M01 NB-IoT module and antenna design, while another one developed a software library that enabled easier communication with the module.

The development process relied on the work developed by these different departments, combining the resulting products into a new development platform. This required working on the hardware schematics and firmware for the new platform. The PCB design and manufacturing were not in the scope of this project and were outsourced. The majority of development time was spent on firmware development. The work done on firmware development included, at first, board bring-up to test the newly developed solution. After that, the development of BSP eased the application development. With everything tested and with BSP completed, the work continued on enabling data logging support to easily collect data from the sensors. Last but not least, fully working firmware examples were developed that demonstrate the possibilities of the platform.

The work is presented in several chapters. Chapter 2 gives the necessary background on the topics relevant to this project. In Chapter 3, the current state of the market and available NB-IoT devices and solutions are presented. Chapter 4 presents the methodology used in the development process from the prototyping in the very beginning to the development of the final board. The results are presented in Chapter 5, focusing on the benchmarks and performance of the newly created solution. Finally, a conclusion is presented in Chapter 6 that gives final remarks and potential areas for improvement of the developed solution.

Chapter 2

Background

2.1 LPWAN

LPWAN stands for Low-Power Wide Area Network and is a term used to represent a group of low-power wide area network technologies. These network technologies rely on a standard LTE wireless service with the addition of protocols for machine-to-machine communication and reduced power consumption. Interconnected IoT devices have different requirements, including varying data rates, power efficiency, and signal ranges. This is where LPWANs come into play with their longer range signals and additional protocols for reduced device power consumption. Given the large number of IoT devices, data transport is optimized in the LPWAN network. This helps limit the impact on the network with massive IoT deployment, preventing congestion problems due to the high number of devices connected to the network. LPWANs were developed to meet all of these unique needs [1].

Main attributes of LPWANs summarized [1]:

- **Low-power**: little power is used due to capability of standby/sleep states, maximum output power is reduced in some cases compared to standard LTE
- **Wide area**: LPWANs range can span between 30 km to 100 km and because the low-frequency connection is used, LPWAN can support IoT devices located in challenging conditions where LTE coverage is limited
- **Data optimization**: LPWAN can accommodate data packet sizes typically from 10 to 1000 bytes at uplink speeds up to 200 Kbps

2.1.1 Different LPWAN technologies

LPWAN technologies can be divided into licensed and unlicensed spectrum. The unlicensed LPWAN technologies include Sigfox and LoRaWAN, while the licensed technologies include CAT-1, CAT-M1/LTE-M, and NB-IoT [2].

CAT-1

Introduced in 2008, LTE was the first 4G connectivity standard tailored specifically for IoT. In 2016, the single antenna version of this standard, called CAT-1 BIS, was released. This paved the way for the development of low-power, wide-area network variants optimized for industrial, commercial, and consumer-facing IoT projects. CAT-1 is a licensed LPWAN network technology with the 3GPP standards. The downlink and uplink rates of this standard make it suitable for a wide range of IoT applications. CAT-1 has a very low latency range of 10-15 milliseconds, which makes it great for video streaming [2].

CAT-M1/LTE-M

CAT-M1, also known as LTE-M, is a licensed LPWA network technology that 3GPP developed as part of the 13th edition of the LTE standard. CAT-M1 is great at transferring low to medium amounts of data over a long range. The downlink and uplink rates of this standard enable small and infrequent data transfers. This is why CAT-M1 is a great option for IoT use cases requiring low to medium data transfer rates, benefiting from reduced battery and power consumption [2].

NB-IoT

This licensed LPWAN technology is developed by 3GPP for cellular network devices and services. Compared to CAT-M1, NB-IoT focuses on indoor coverage, long battery life, and high connection density. NB-IoT limits channel bandwidth and offers very low uplink and downlink rates. It also has high latency, making CAT-M1 a better option for IoT networks where latency is crucial. NB-IoT is best suited for applications with infrequent data transfer and low to medium transmission rates [2].

LoRaWAN

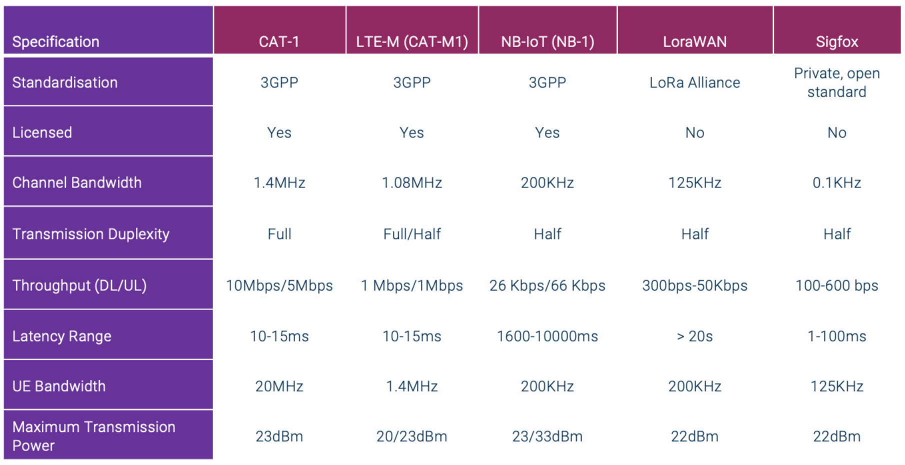
LoRaWAN is an unlicensed LPWAN technology developed by the LoRa Alliance. It is unique in that it does not rely on existing cellular infrastructure. Similar to Wi-Fi or Bluetooth, it uses independently launched infrastructure to run. Rather than finding a provider, IoT businesses are expected to build LoRaWAN infrastructure on their own to support their IoT network. LoRaWAN providers do exist but are not that common. This also brings some benefits because the infrastructure can be designed and tailored specifically, which means that it can in some situations offer better coverage and increased robustness [2].

SigFox

Sigfox is similar to LoRaWAN in terms of having to deploy a cellular infrastructure and shares the same drawbacks and benefits. However, it is a more robust non-cellular LPWAN option and a lower-cost one. It is also a proprietary technology, owned by Sigfox, meaning it can only be obtained and used through Sigfox. This can make it more limiting than other options and make it challenging to transition to a different type of connectivity in the future [2].

Comparison of different LPWAN technologies

The comparison of the aforementioned technologies is given below in Figure 2.1 and includes their channel bandwidth, transmission duplexity, downlink peak rate (marked as DL) and uplink peak rate (marked as UL), latency range, bandwidth, and maximum transmission power.



Specification	CAT-1	LTE-M (CAT-M1)	NB-IoT (NB-1)	LoraWAN	Sigfox
Standardisation	3GPP	3GPP	3GPP	LoRa Alliance	Private, open standard
Licensed	Yes	Yes	Yes	No	No
Channel Bandwidth	1.4MHz	1.08MHz	200KHz	125KHz	0.1KHz
Transmission Duplexity	Full	Full/Half	Half	Half	Half
Throughput (DL/UL)	10Mbps/5Mbps	1 Mbps/1Mbps	26 Kbps/66 Kbps	300bps-50Kbps	100-600 bps
Latency Range	10-15ms	10-15ms	1600-10000ms	> 20s	1-100ms
UE Bandwidth	20MHz	1.4MHz	200KHz	200KHz	125KHz
Maximum Transmission Power	23dBm	20/23dBm	23/33dBm	22dBm	22dBm

Figure 2.1: Comparison of LPWAN technologies [2]

Given the fact that the thesis was done in collaboration with ST and that the goal was to work on a new development platform for asset tracking, the main focus going forward was put on NB-IoT from all the listed LPWAN technologies. The low power consumption and low data rates, combined with indoor coverage and the fact that no infrastructure needs to be deployed, which keeps the costs down, made it a perfect choice for use in this project for asset tracking scenarios. This is why NB-IoT was selected as the technology on which the new development platform was to be based.

2.2 NB-IoT in more details

Whether it was browsing the internet using mobile data or making voice or video calls, people have relied on cellular networks for quite some time now. The span of these cellular networks and their coverage has grown over the years, and they now encompass the whole planet. This meant that infrastructure had to be placed all around to support connecting people on a global scale. As cellular network technology improved, increased bandwidth made it possible for more data to circulate throughout the network. However, for a long time, these networks were used solely for human-oriented services.

Taking all of these facts into consideration, around 2005, 3GPP started a research initiative. The group began researching the possibilities of reusing the already present cellular network infrastructure and defining a new communication standard for machine-oriented services. After several years of research and development, the Narrow-Band Internet of Things (NB-IoT) protocol emerged as a product of this effort. The NB-IoT protocol represents an LPWA network technology for low-data-rate applications, typically used in environmental monitoring and smart metering.

2.2.1 Technical aspects

The underlying details of the NB-IoT protocol are well abstracted in the ST87M01 module, so we do not need to be too concerned about them. One of the aims of this project was to develop a platform where NB-IoT is one part of it and to test the ST87M01 module. Therefore, broader details are omitted, but given that some background is still necessary when discussing topics where NB-IoT and/or ST87M01 come into context, some of the technical aspects are mentioned below. This is envisioned to aid in better understanding certain design choices made during the development and to avoid getting lost in the terminology.

NB-IoT network

The basic NB-IoT network consists of several parts [3]:

- **NB-IoT terminal**: An IoT device equipped with an NB-IoT modem that has a SIM card to enable access to the NB-IoT network
- **NB-IoT base station**: Mainly refers to the base station that has already been deployed by telecom operators and that supports and manages terminals attached to it
- **NB-IoT core network**: Through this network, the NB-IoT base station can connect to the NB-IoT cloud

Physical channels

Depending on the direction of data transmission between the terminal and the base station, there are two distinct classifications:

- **Uplink**: Data moves from the terminal to the base station
- **Downlink**: - Data moves from the base station to the terminal

Both uplink and downlink directions consist of several channels. These channels logically separate functions that make data transmission possible. When transmitting data, the data traffic usually consists of the data that needs to be exchanged and also includes some control signals or control data. This control data or control signals can be used to control the data exchange. The same is true for the NB-IoT protocol; besides the data itself, there are also control signals used, for example, to adjust the communication parameters between the terminal and the base station for the most optimal data transfer.

Besides the actual data and controlling the exchange of this data, another important aspect is the resources that make communication possible. If there are multiple terminals connected to the base station, the question is how to determine the resource allocation so that communication with every terminal is possible. The resources in NB-IoT refer to frequency and time. Different time slots can be allocated to each terminal, and within those time slots, the communication between the base station and a particular terminal can take place. To sum it up, in NB-IoT there are data, control, and resource channels.

The uplink channels present in NB-IoT are [4]:

- **NPRACH**: Also known as the narrowband physical random-access channel, it refers to the time-frequency resource used to transmit random access preambles sent periodically by the terminal. These preambles are used by the base station to estimate the uplink timing
- **NPUSCH**: Also known as the narrowband physical uplink shared channel, it has two formats. The first format, "Format 1," carries the actual data in the uplink, while the second format, "Format 2," carries control information

The downlink channels present in NB-IoT are [4]:

- **NBPBCH**: Also known as the narrowband physical broadcast channel, it resides in every radio frame and carries the Master Information Block, which provides information such as system bandwidth and frame number
- **NPDCCH**: Also known as the narrowband physical downlink control channel, it is used to carry downlink control information. A terminal needs to

monitor NPDCCH to obtain three types of information: scheduling NPUSCH, scheduling NPDSCH, and paging direction

- **NPDSCH**: Also known as the narrowband physical downlink shared channel, it is the core data-bearing channel used for the transmission of unicast data. It segments the data packet received from the higher layer into transmission blocks and transmits one at a time

Semi-static link adaptation

It is hard for NB-IoT to provide long-term and continuous indication of channel quality because most transmissions happen infrequently and packet sizes are small. To remedy this, NB-IoT introduces coverage levels instead of a dynamic link adaptation scheme. There are three kinds of coverage classes: normal coverage, robust coverage, and extreme coverage, which correspond to the minimum coupling losses of 144 dB, 158 dB, and 164 dB, respectively. The modulation scheme, coding mode, and the number of repetitions of data transmissions can be selected according to the coverage class [3].

Data retransmission

The NB-IoT protocol adopts a data retransmission mechanism to improve the performance of recovering the information content from the modulated carrier wave as well as to improve coverage performance. All channels support data retransmission. The retransmission count for each channel, as defined by the 3GPP, is showcased in Table 2.1 below [3].

Physical signal/physical channel name	Repetitions
Downlink: NPBCH	Fixed 64 times
Downlink: NPDCCH	[1,2,4,8,32,64,128,256,512,1024,2048]
Downlink: NPDSCH	same as above + [192,384,768,1536]
Uplink: NPRACH	[1,2,4,8,32,64,128]
Uplink: NPUSCH	[1,2,4,8,32,64,128]

Table 2.1: Retransmission amount for each channel in Nb-IoT [3]

Frame structure

The frame structure of NB-IoT is similar to LTE. As can be seen in Figure 2.2, for the downlink direction, in the time domain, a radio frame is comprised of ten subframes. Each subframe has a length of 1 ms and consists of two slots, with each slot containing seven orthogonal frequency-division multiplexed symbols. The number of these radio frames is referred to as the system frame number and can range from 0 to 1023. In the frequency domain, one physical resource block assigned to slot 0 and slot 1 for the downlink direction consists of twelve consecutive subcarriers with a spacing of 15 kHz [4].

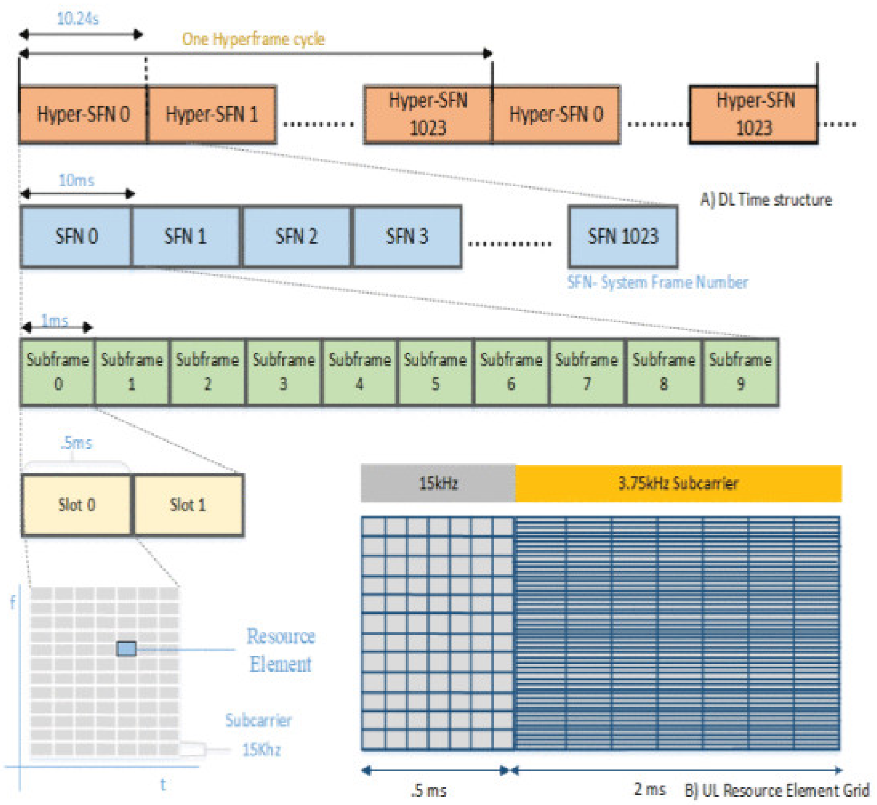


Figure 2.2: NB-IoT frame structure [4]

For the uplink direction, the resource grid has two subcarrier spacing options: 15 kHz and 3.75 kHz. In the time domain, it has the same characteristics as the downlink counterpart. The frequency domain is the same for slot 0; however, for the 3.75 kHz subcarrier spacing in slot 1, the total number of subcarriers is four times greater. The duration of slot 1 is increased to 2 ms [3].

Establishing a connection

The terminal indicates to the base station that it wants to connect to the network by sending a request. The request can be rejected or granted. If the request is granted, the connection establishment is successful. The established connection can later be completely terminated by the terminal or by the base station if it determines that the terminal is "misbehaving" (e.g., too many connection requests followed by terminating the connection in a short time period). To lower power consumption, the terminal can announce to the base station that it wants to drop the connection but stay attached and registered to that base station. This ensures quicker re-establishment of the connection during a later reconnection attempt [4].

NB-IoT deployment modes

NB-IoT currently supports only bandwidth of 180 kHz and the following types of deployment modes [3]:

- **Independent deployment**: Utilizes an independent frequency band that does not overlap with the frequency band of LTE
- **Guard-band deployment**: Utilizes the edge frequency band of LTE
- **In-band deployment**: Utilizes the LTE frequency band for deployment and occupies one physical resource block of the LTE frequency band

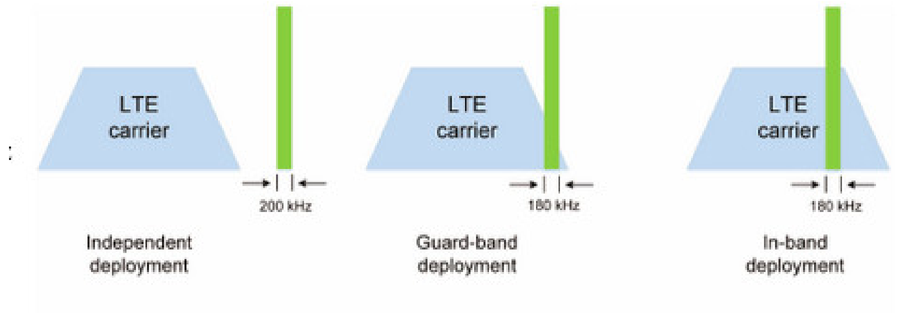


Figure 2.3: Deployment modes of NB-IoT [3]

Low power consumption

Using power saving mode and expanded discontinuous reception, longer standby time can be realized in NB-IoT, thus preserving a greater amount of energy. In power saving mode, the terminal is still registered online to the base station but cannot be reached by the base station. A terminal can switch to low-power mode by first proposing two time periods to the base station called T3324 (active timer) and T3412 (extended timer). It is up to the network and the base station to accept the proposed timer values or not. Expanded discontinuous reception further extends the sleep cycles of the terminal in idle mode. The power saving mechanism of both power saving mode and expanded discontinuous reception is presented in Figure 2.4, marked with PSM and eDRX respectively.

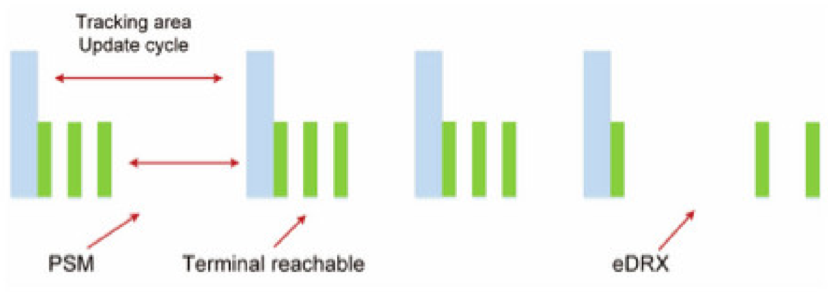


Figure 2.4: PSM and eDRX modes of NB-IoT [3]

2.3 GNSS

Slowly over time, the satellite navigation system has become an integral part of every application where mobility is an important aspect. Nowadays, everyone relies on navigational services, whether it is exploring an unknown city as a tourist, trying out a new route when driving a car, or seeing which bus to catch when commuting daily.

In general, navigational services are called by one name, GNSS, which represents a navigational system that encompasses three main satellite technologies: GPS, GLONASS, and Galileo, and recently BeiDou.

2.3.1 GPS

Developed by the United States Department of Defense, the Navstar GPS represents a space-based navigation system created to support the needs of the USA military forces and accurately determine their position, velocity, and time in a common reference system [5]. As the technology proved really useful and reliable, it was later also shared for public use.

2.3.2 GLONASS

The GLONASS is nearly identical to GPS. The GLONASS satellite-based radio-navigation system provides positioning and timing information to users. It is operated by the Ministry of Defense of the Russian Federation [5].

2.3.3 Galileo

Galileo is the European Union's GNSS, developed to provide Europe with an independent and reliable satellite navigation system. Unlike GPS and GLONASS, which were initially created for military use, Galileo was designed to serve both civilian and governmental applications from the start [6].

2.3.4 BeiDou

The BeiDou Navigation Satellite System is one of the newest GNSS constellations, developed to provide China with an independent PNT capability. It has rapidly expanded from a regional system to a fully global one [6].

2.3.5 General overview of the technology

The GNSS satellite technologies consists mainly of three segments [7]:

- **pace segment**: Consists of GNSS satellites orbiting above the Earth arranged in orbits to provide the desired coverage
- **Control segment**: Comprises a ground-based network of master control stations, data uploading stations, and monitor stations that exchange data between themselves and with the satellites, providing adjustments to the satellites' orbit parameters and onboard high-precision clocks when necessary to maintain accuracy
- **User segment**: Consists of equipment (e.g., mobile phones) that processes the received signals from the GNSS satellites and uses them to derive location and time information

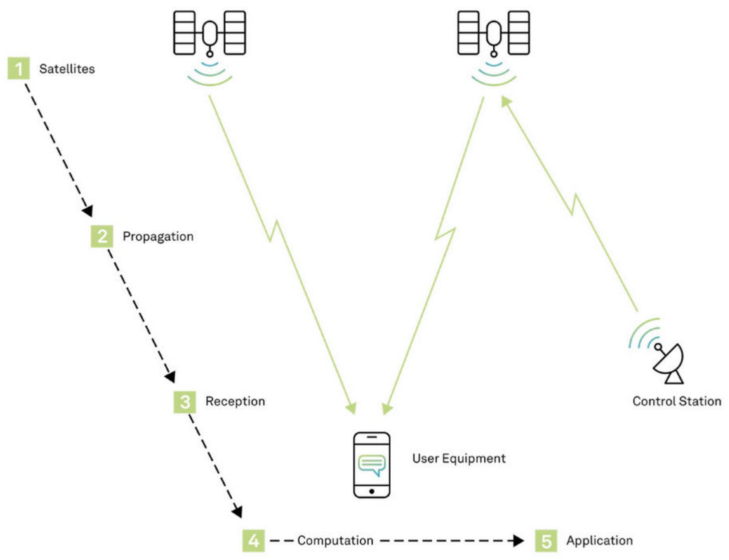


Figure 2.5: Basic GNSS [7]

From Figure 2.5 [7]:

1. At any given moment in time, satellites know their parameters that define their orbit and the time very accurately. The ground-based control stations are responsible for adjusting these parameters when necessary

2. Satellites regularly broadcast their orbit parameters and time, as well as their status, to the user equipment

3. User equipment receives the signals from multiple GNSS satellites and, for each satellite, recovers the information that was transmitted and determines the time of propagation (the time it takes the signals to travel from the satellite to the receiver)

4. User equipment uses the recovered information to compute time and position

5. User equipment provides the computed position and time to the end-user application

2.3.6 How is positional data represented

The definition of a reference coordinate system is important for the description of satellite motion. The position of the receiver is calculated with respect to the current position of the satellite at the moment of calculation. There are two ways to describe coordinates when dealing with satellite motion. The coordinate system used is a Cartesian terrestrial reference system. It is defined by convention with three axes, where the Z-axis coincides with the Earth's rotation axis, the X-axis is associated with the mean Greenwich meridian, and the Y-axis is orthogonal to both the Z and X axes, completing the right-handed coordinate system. This representation forms Cartesian coordinates (X, Y, Z). As this representation has less significance in navigation, ellipsoidal coordinates are commonly used for navigation. These encompass longitude ϕ, latitude λ, and height (h) above the ellipsoid that represents Earth's sphere, as can be seen in Figure 2.6 [5].

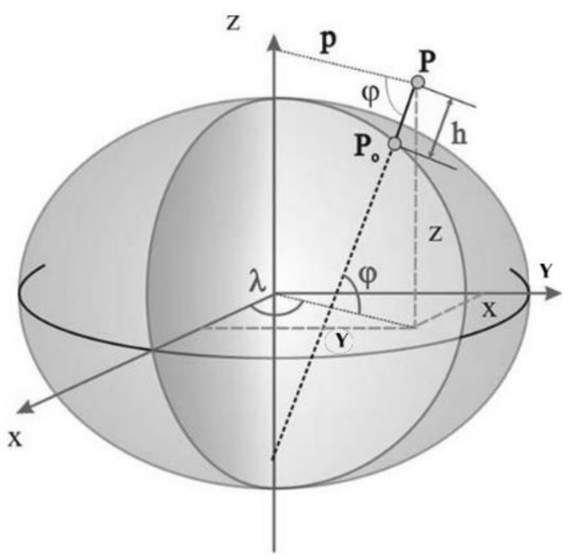


Figure 2.6: Cartesian and ellipsoidal coordinates [5]

2.4 RTOS

Operating systems that are used for the development of systems which must fulfill the requirement of responding to events within a certain defined time limit are known as real-time operating systems [8]. Depending on the severity of the outcome of the event and how critical it is, we can have systems that are [8]:

- **Soft real-time**: Missing the deadline has no serious implications regarding safety or well-being but can lead to a loss in quality or performance
- **Hard real-time**: Missing the deadline can cause serious issues, sometimes leading to life-threatening situations

Usually, in the context of smart metering, when using an RTOS, the system is considered to be soft real-time. The scenario where data should be gathered from a sensor or more than one sensor, and then sent over the network or some serial interface, has no serious implications regarding the safety of the environment it is put in or the people nearby. The worst thing that could go wrong is missing a deadline for reading or not sending the data in time. Since the data is not used to quickly react to an event, sometimes the devices are used to send one reading per day, eliminating the stress of not allowing some deadline to be missed at any cost. If we imagine a scenario of reading the temperature and missing the deadline by a couple of seconds, the value should be pretty much comparable to the one made if we stayed within the deadline, if not the same. In contrast, an airbag system has to respond within the exact time deadline measured in milliseconds for it to be effective.

But why even introduce an RTOS if deadlines are not so important? Could bare-metal programming have solved the problem? Well, yes, but with increasing the complexity of the functionalities that need to be supported and increasing the number of functionalities, it becomes harder and harder to develop and maintain the system. Since an RTOS has support for dividing and isolating the application logic into so-called "tasks", and taking care of the scheduling on its own, we can focus on the actual problems that each of the tasks needs to solve. Although we do not control the scheduling itself, we can configure it and make it deterministic. We can make the task execution order predictable.

Another very important aspect is the smart usage of idle processing time. Since the data from the sensors is read periodically, and the sending part is also executed periodically, there are periods of idleness that should be taken advantage of to minimize power consumption. In an RTOS, there is usually a default task that manages the idle periods, called the "idle task", which can be configured to do almost anything. By default, it just actively waits, but it can also be configured to put the device in low power mode. Taking into account the necessity for time management, more specifically deadlines that are not so strict, easier development, and easier management of idle periods, it was determined that the usage of an RTOS was appropriate for this project. Given that the project relies on the STM32 portfolio of devices, an instance of an RTOS that is mature, popular, and well supported on this platform, called FreeRTOS®, was used in the development of this project.

2.4.1 FreeRTOS®

FreeRTOS® is a real-time operating system that is professionally developed, strictly quality controlled, robust, well supported, and does not contain any intellectual property ownership ambiguity, which makes it free to use even in commercial applications. It is currently under the stewardship of AWS [9]. It is also well integrated and supported in the STM32 ecosystem with plenty of examples, which made it a perfect candidate to use for this project.

FreeRTOS® tasks

Tasks are implemented as C functions. Each task is a small program that has an entry point, normally runs forever in an infinite loop, and does not exit. A task must not be allowed to return from the function that implements it in any way. Each created task is a separate execution instance, and each instance has its own stack. Tasks can be in any of the following states [9]:

- **Suspended**: Task unavailable to the scheduler, excluded until explicitly made available again
- **Ready**: Task ready to be selected to run
- **Running**: Currently executing task
- **Blocked**: Waiting on some event (e.g., delay to pass, or to receive a notification)

The scheduler will, depending on the task priorities and the states they are in, change the task states accordingly. The scheduler is invoked periodically and it ignores all the suspended tasks. Tasks can be suspended by some other task or they can suspend themselves. The scheduler takes the suspended tasks into consideration again only if they are explicitly resumed (only some other task can resume another suspended task). When that happens, they are moved to a ready state. Blocked tasks that are eligible for unblocking (delay has passed or a certain event on which the task was blocked occurred) are moved to the ready state. Blocked tasks can also be suspended by some other task.

After inspecting all of the tasks in the ready state, if a task that is ready has a higher priority than the currently running task (note that only one task can be running at a time), the current running task becomes ready, and the newly selected task runs. When a task is running, it can transition to a suspended or blocked state by suspending itself or blocking, waiting on some event. It can only be put in the ready state from the running state by the scheduler. If several tasks have the same priority, the round-robin policy will be used among them. Every time there is no suitable task available for running, a special task called the idle task executes. This task has no specific functionality but can be enabled to check if it is suitable to put the system into a low-power mode [9].

FreeRTOS® synchronization primitives

FreeRTOS® supports all of the standard primitives that can be found in a typical RTOS, of which the relevant ones for this project are [9]:

- **task notifications**: An efficient synchronization mechanism that allows one task to directly notify another task and synchronize with ISRs, all without the need for a separate communication object (like a semaphore, for example)
- **message buffers**: A lighter weight alternative to queues that hold variable-length data, used to exchange data between tasks and for synchronization. One task can block on receive and wait for data from another task

Dynamic memory management

There are five available dynamic memory management schemes [9]:

- **heap_1**: Simply subdivides a byte array into smaller blocks each time a memory allocation request is made. The heap is implemented as a statically allocated array and cannot free allocated memory
- **heap_2**: Similar to heap_1, but uses a best-fit algorithm to allocate memory, and allocated memory can be freed. The best-fit algorithm ensures that the free block of memory closest in size to the number of bytes requested is selected
- **heap_3**: Represents the standard *malloc()* and *free()* functions from the C standard library. The added functionality is that when either is used, the scheduler is suspended because these functions are not thread-safe
- **heap_4**: Similar to heap_2, but uses a first-fit algorithm to allocate memory and combines (coalesces) adjacent free blocks of memory into a single larger block, which minimizes the risk of memory fragmentation
- **heap_5**: Same as heap_4, but can combine memory from multiple separated memory spaces into a single heap, useful when the RAM provided by the system does not appear as a single contiguous block in the memory map (e.g., RAM consisting of multiple memory banks that are not contiguous)

Tickless idle

If tickless idle mode is enabled, each time there is no task to be executed, once the idle task is running, a user-defined action is performed that enables the user to put the system into a low-power mode. In contrast to this first action, there is a second user-defined action available for defining what should happen after the system exits the low-power state. These actions are defined as function calls where the implementation is left to the user. Note that if the implementation is not provided, the functions will do nothing. Typically, the clock and/or some peripherals can be stopped, and a system call can be invoked to put the system into a selected low-power state when implementing the first action. It can also be defined under what circumstances the system can wake up. The second action can resume the clock and the operation of stopped peripherals.

2.4.2 CMSIS RTOS

The CMSIS-RTOS is an API for real-time operating systems that provides a standardized programming interface. This interface includes software templates, middleware, libraries, and other components. A typical CMSIS-RTOS API implementation interfaces with an existing RTOS [10]. Figure 2.7 represents the architecture of the CMSIS-RTOS API. The main advantage of this API is that the developed system can be easily used with many different real-time operating systems. In case the initially chosen RTOS needs to be swapped for another, there is no penalty and no major code refactoring required.

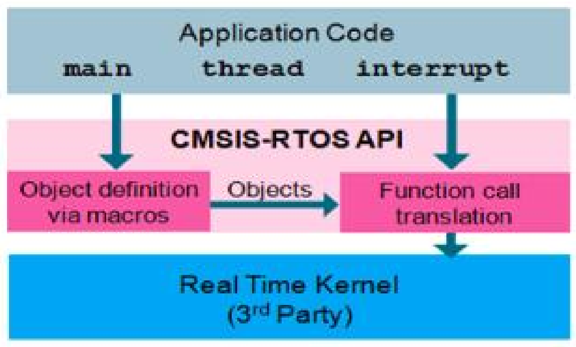


Figure 2.7: CMSIS RTOS API architecture [10]

2.4.3 FreeRTOS® on STM32

To be able to run FreeRTOS® on an ST board, and to run FreeRTOS® in general, a time source is needed because the FreeRTOS® scheduler needs to periodically check if a task of higher priority is available so that it can preempt it [9]. Then, as the application logic is divided into tasks and objects for message passing and task synchronization, memory needs to be assigned for these tasks and objects. Since low power consumption is one of the major goals of this project, the methods to exploit low power modes of the ST MCUs using FreeRTOS® are also showcased.

Time management

On the ST MCUs, there exists a dedicated timer unit called "SysTick" just for keeping track of the system time. The system time is measured in ticks, where usually by default each tick occurs every 1 millisecond, giving the ability to track elapsed time in firmware. Every millisecond, an interrupt would fire that would, in turn, increment a global variable that stores the number of ticks that have occurred since the MCU was powered on. With this simple scheme, delays could be implemented or the duration of some event could be tracked. When FreeRTOS® is enabled and used on some ST boards, it takes control of the "SysTick" timer and its configuration and initialization, and by default, sets the priority of this peripheral to the lowest.

This is not a big problem in general, but it can lead to some issues depending on the situation. If some ISR were running and it used HAL functions that rely on "SysTick" there, like functions that implement delay, it would result in a deadlock [8]. Why does this occur? Because the ISR would be of higher priority than "SysTick" timer interrupts, and the variable that keeps track of how many ticks have passed would never be updated. Thus, from the perspective of the HAL function that implements delay, the time would be standing still. The delay would never expire, and since those functions are blocking, the whole system would be stuck and come to a halt. In these cases, it is recommended to use a different timer to implement the "SysTick" functionality.

Memory management

After configuring the timer that enables FreeRTOS® to function, the next important aspect is memory management. As every object in the FreeRTOS® library, such as a task or a mutex, requires memory when being instantiated, it is crucial to decide what setup to use and how to configure this memory. The majority of the configuration is done by the STM32CubeIDE, so what is left to the user is to choose whether static or dynamic memory allocation should be used. If dynamic memory allocation is chosen, then a proper dynamic allocation scheme needs to be selected from the five supported schemes described in chapter 2.4.1.

Finally, another important aspect when dealing with dynamic memory is that this memory is dedicated to the heap region. This implies that the linker script needs to be set up so that there is enough memory in the heap region for the planned FreeRTOS® objects (tasks, mutexes, etc.).

Low power mode management

It is useful, depending on the application, to put the MCU in a low power mode if idle periods are long enough [10]. This helps with lowering power consumption and, if the system runs on a battery, prolonging the battery life, thus enabling the system to run longer before requiring battery charging or replacement. As mentioned in 4.1.3, there are several low power modes available on the MCU. If any of the low power modes is entered with a configuration such that only some interrupt can cause the MCU to exit it, this could be disadvantageous depending on the situation. If the SysTick is configured to interrupt or tick every 1 millisecond, the MCU would enter some low power state but immediately exit on the next tick.

This can be remedied by lowering the tick frequency, but what if we want to avoid this problem completely? FreeRTOS® supports the previously mentioned "tickless" idle mode. In this mode, even the SysTick timer managed by FreeRTOS® itself is stopped before entering some low power state so that it cannot interfere and wake up the system. Only some other interrupt source, like an external GPIO interrupt caused by some external component (like a sensor of some sort) or an LPTIM interrupt (because LPTIM can run even when the system is in a low power state) can wake up the system [11].

When using tickless idle, an amount of ticks for which the device is going to spend in low power state should be provided in advance so that upon return, the variable that tracks the amount of total ticks can be updated. This way, a correction is made because SysTick is not active in low power mode, thus the variable containing ticks is not updated during the time spent in low power mode. Of course, an effort can be made to track this time using an LPTIM, for example, but inevitably some drift will be introduced. As it turned out, there is a small delay when stopping the SysTick and starting the LPTIM, when entering the low power mode and returning back, and also when stopping the LPTIM and enabling SysTick once again.

2.5 DTDL

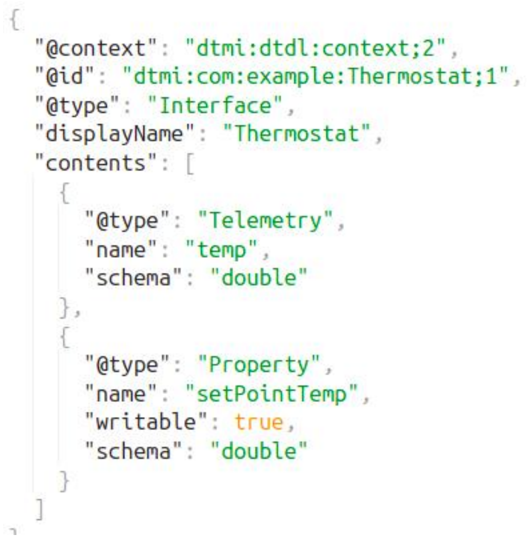
DTDL represents a language for describing digital twin models of smart devices, assets, spaces, and environments. This enables digital twin solutions to provision, use, and configure digital twins of all kinds from multiple sources in a single solution [12]. When working with sensor devices, for example, this enables a concise way of describing different sensors in one common language and format. Using DTDL to describe and model different kinds of sensors eases development in particular situations where communication with the sensors and data gathering comes into play.

Usually, one typical scenario of such communication is where the data is read from these sensors and sent to the cloud platform, after which the data is processed in the cloud and used for different purposes, depending on the needs. Since there can be multiple different sensors, DTDL makes it possible to define and represent each one of them and their capabilities in an abstract way, and most importantly, in one general format. For example, representing a temperature sensor with the capability to report temperature readings in units of Celsius. If a pressure sensor were to be included as well, DTDL could be used to easily define this sensor in a standardized way. It does not matter that the two sensors are from different manufacturers or of different types. Even if they are the same, they can still have different units for their data; every nuance can be described precisely by DTDL.

Digital twins for smart devices, assets, spaces, and environments are described using a variant of JSON called JSON-LD. JSON-LD represents a JSON format for describing linked data. Linked data, in general, is structured data that is interlinked with other data. JSON-LD is particularly useful in resource description framework systems when describing resources in a distributed, extensible way. The DTDL is made up of a set of metamodel classes that are used to define the behavior of all digital twins. DTDL provides semantic type annotations so that the semantics of the data can be reasoned about, not just the schema of the data. For example, properties that are semantically annotated as "temperature" will be reasoned about as temperature (compared, converted to like units, etc.). The main metamodel classes present in DTDL are [12]:

- **Interface**: Describes the contents (commands, components, properties, relationships, and telemetries) of any digital twin
- **Command**: Describes a function or operation that can be performed on any digital twin
- **Component**: Enables interfaces to be composed of other interfaces
- **Property**: Describes the read-only and read/write state of any digital twin
- **Relationship**: Describes a link to another digital twin and enables graphs of digital twins to be created
- **Telemetry**: Describes the data emitted by any digital twin

An example of an interface that represents a thermostat device and has one telemetry class that reports the temperature measurement, and one read/write property class that controls the desired temperature, can be seen in Figure 2.8.



```
{
  "@context": "dtmi:dtdl:context;2",
  "@id": "dtmi:com:example:Thermostat;1",
  "@type": "Interface",
  "displayName": "Thermostat",
  "contents": [
    {
      "@type": "Telemetry",
      "name": "temp",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "setPointTemp",
      "writable": true,
      "schema": "double"
    }
  ]
}
```

Figure 2.8: DTLD interface example in JSON-DL [12]

Chapter 3

Related work

3.1 State of the market and available NB-IoT solutions

Since the focus of this thesis is the NB-IoT protocol and the development of a platform based on this technology, more details are given in the continuation of this chapter on the overall state of the market and some of the already available NB-IoT solutions. The aim is to explore the pros and cons of the current solutions and see what the new development platform can bring. Another reason is to explore why NB-IoT is becoming more and more popular.

The steady growth of the NB-IoT market in the past couple of years, as evident in Figure 3.1, brings optimistic predictions regarding the value this technology will attain in the near future.

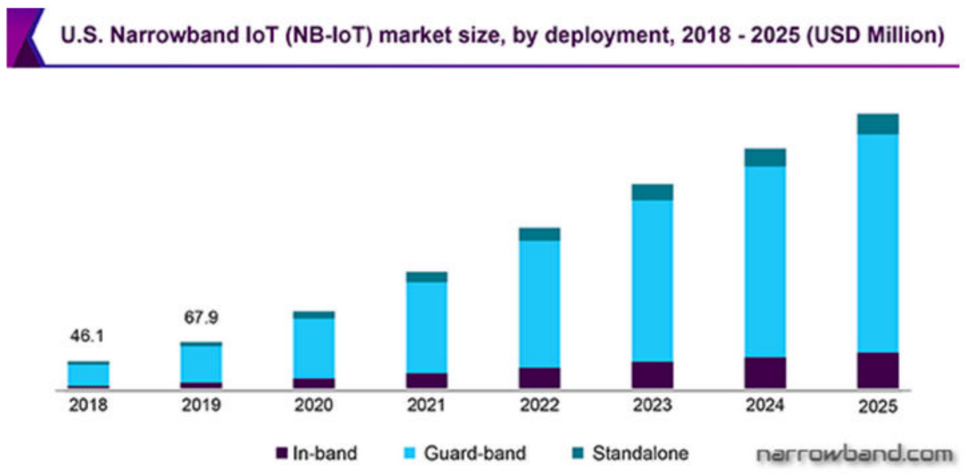


Figure 3.1: NB-IoT market size by deployment in the U.S. [13]

It is predicted that the NB-IoT chipset market will reach a market cap of 22.10 billion dollars by 2030 [13], which represents a significant increase from the estimated market value at the time of writing this. The already available network infrastructure reduces the cost of deploying NB-IoT devices, aiding in the rapid growth of the adoption of this technology. Furthermore, the fact that NB-IoT devices require low power and are usually powered by batteries means that their size is small and compact. Given also how this technology enables solid connectivity indoors, again without the need for additional infrastructure, this makes it perfect and less expensive to set up and use in various places.

But this does not come without a few drawbacks. Namely, the signal transmission frequency is low, which implies that the data transmission rates are also low. This is by design to sustain the devices running on battery and to enable this technology to be used even indoors and over larger distances. With these constraints, designers of NB-IoT devices must ensure their devices pass network approval tests and certifications, making the design cycle more difficult [13].

At this moment, several companies are driving the NB-IoT technology and its adoption forward. These companies offer dedicated NB-IoT modules, and some also provide development boards to make it easier for users to try out the NB-IoT functionalities. For distinction, these companies are referred to here as "providing" companies as they provide dedicated solutions. There are also numerous other companies that offer NB-IoT development boards and solutions, but there is a difference. Often, because of the state of the market, these other companies develop a solution that uses and relies on components from the "providing" companies. These companies are referred to here as "relying" companies.

As presented in the subsequent sections, there are not many dedicated module manufacturers or "providing" companies. Also, some of the "providing" and "relying" companies are, in turn, dependent on some other external company or companies for chip fabrication. So far, there are only a few solutions where the whole system, from the NB-IoT module to the other accompanying components offered on the NB-IoT development boards (typically a GNSS module), is developed and fabricated by one single company. The sections below provide more details on some of the available NB-IoT solutions and companies involved in the NB-IoT scene.

3.2 Quectel

Quectel is a global company providing a wide range of IoT solutions. It offers and manufactures solutions that implement cellular technologies from 2G/3G to 4G and 5G modules, modules for use in the automotive sector, and GNSS modules.

3.2.1 Quectel BC65

The BC65 is a high-performance multi-band NB-IoT module. The BC65 features an ultra-compact, unified form factor of 17.7 mm $\times$ 15.8 mm $\times$ 2.2 mm, which makes it a perfect choice for size-sensitive applications. Its small package allows it to be embedded easily and to provide reliable connectivity. The surface-mounted technology makes it ideal for durable and rugged designs. The combination of the module's very small size, ultra-low power consumption, and wide operating temperature range allows it to serve a wide range of IoT use cases [14].

3.2.2 BC65-TE-B

The BC65-TE-B is an NB-IoT development board that supports an Arduino connector interface. Designed in a 70.0 mm $\times$ 74.0 mm $\times$ 1.6 mm form factor, the BC65-TE-B can be used to develop and debug applications that communicate through the NB-IoT protocol [15].

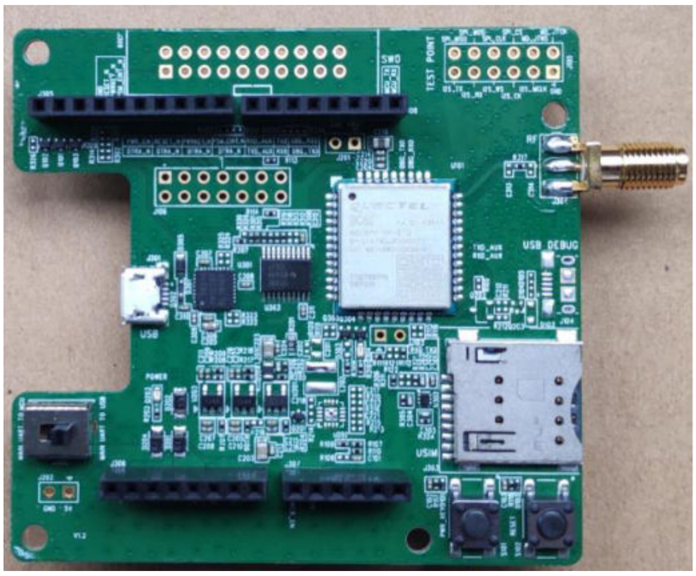


Figure 3.2: Top view of the BC65-TE-B [15]

As showcased in Figure 3.2, its prominent features are a USB connection, which enables support for AT commands using UART-over-USB, a SIM card slot, Arduino connectors, and of course, the BC65 module.

3.3 Telit

Telit started as an engineering company providing research and development services for multinational telecoms. The company manufactures and markets products for IoT and has been providing enterprise-grade IoT products and software for over 30 years.

3.3.1 ME910C1 Series

The ME910C1 is an LTE UE Cat M1/NB1 module specifically tailored for IoT applications, offering optimized power consumption and enhanced coverage. The module features a 28.2 mm x 28.2 mm x 2.2 mm LGA form factor. It supports the AT command format and several protocols typically used in IoT, such as IPv4/IPv6 stack with TCP/UDP support, LWM2M, SSL, etc. PSM and eDRX power-saving options are also supported [16].

3.3.2 Bravo evaluation kit

Bravo is an IoT evaluation kit, providing a quick and easy way to build a working proof of concept for IoT deployment. Bravo can be used as a stand-alone solution or as a non-stand-alone solution connected to Arduino or Raspberry Pi compatible devices. Bravo uses the ME910C1 module with LTE-M and NB-IoT support and 2G fallback [17].

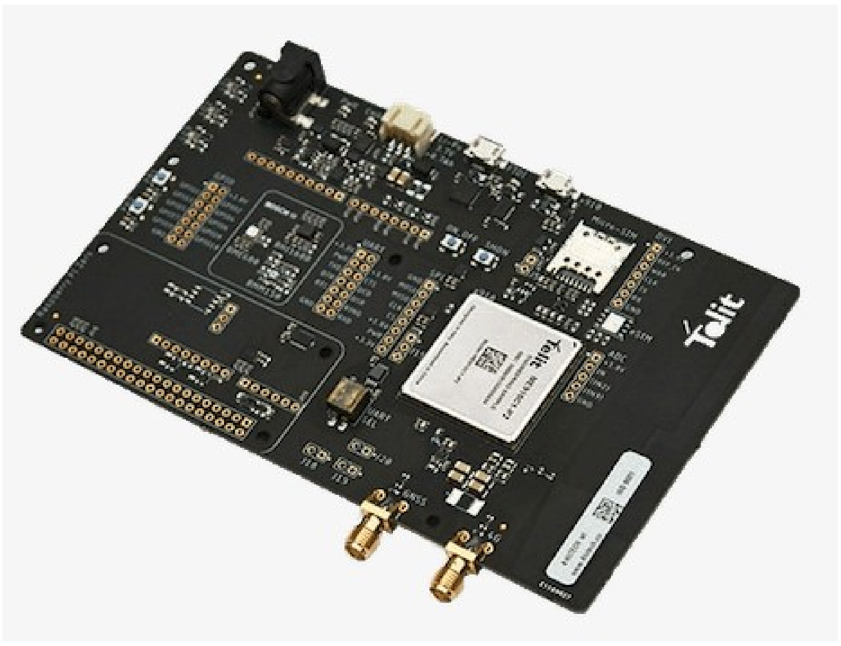


Figure 3.3: Telit Bravo development board [17]

As can be seen in Figure 3.3, the development board features a USB connection, which enables support for AT commands using UART-over-USB, a SIM card slot, Arduino connectors, and of course, the ME910C1 module. It also features two antenna connectors, one RF antenna for NB-IoT and another for GNSS. Several GPIO options, SPI and ADC peripherals, motion and environmental sensors are also included. Additionally, it supports running from a battery by incorporating a battery charging circuit.

3.4 Nordic Semiconductors

Nordic Semiconductor is a fabless semiconductor company specializing in wireless communication technology. Some of the solutions and technologies offered by Nordic include Bluetooth Low Energy solutions, ANT+, Thread, and Zigbee. In recent years, Nordic has also offered low power, compact LTE-M/NB-IoT cellular IoT solutions and Wi-Fi solutions.

3.4.1 nRF9160 SiP

The nRF9160 SiP integrates an application processor, multimode LTE-M/NB-IoT/GNSS modem, RF front-end, and power management in a 10 x 16 x 1.04 mm package. By accomplishing this, it provides the most compact solution for cellular IoT on the market. It offers a modern and powerful 64 MHz Arm® Cortex®-M33 CPU with on-chip flash and RAM exclusively for application use. The nRF9160 SiP contains application layer protocols such as HTTP(S), CoAP, MQTT, and LWM2M, built upon IPv4/IPv6, TCP/UDP, TLS/DTLS. It supports both the PSM and eDRX power-saving features, with a PSM floor current as low as 2.7 µA, and with an eDRX interval of 655 seconds, the idle average current is 6 µA for LTE-M and 9 µA for NB-IoT [18].

3.4.2 nRF9160 DK

The nRF9160 DK is a hardware development platform used to design and develop application firmware on the nRF9160 SiP. It features onboard antennas (for LTE and GNSS), a SIM card holder, and provides developers with access to all I/O pins and relevant module interfaces. Also included on this development board is the nRF52840, which serves as a board controller and network processor for Bluetooth and Wi-Fi. There is support for Arduino connectors, USB and UART over USB interfaces, as well as user buttons and LEDs [19].

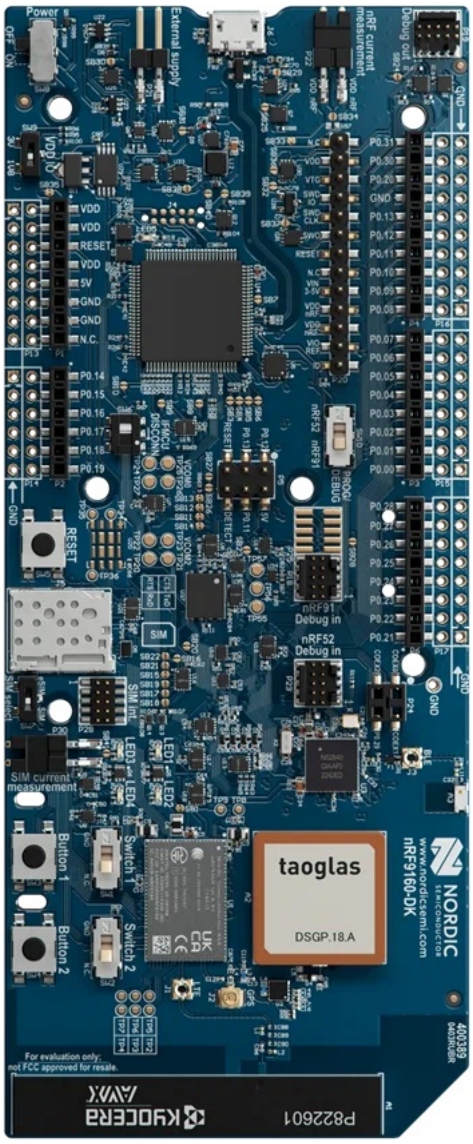


Figure 3.4: nRF9160 DK [19]

The top view of the nRF9160 DK is shown in Figure 3.4 above.

3.4.3 Nordic Thingy:91

The Nordic Thingy:91 is a battery-operated prototyping platform for cellular IoT, showcased in Figure 3.5 below. It is ideal for rapid development of prototypes for cellular IoT systems and is especially suited for asset tracking applications and environmental monitoring. It features LTE-M/NB-IoT/GPS, Bluetooth LE, and NFC passive tag antennas, with the SIM card slot used for LTE-M/NB-IoT. The development board comes with several sensors available for temperature, humidity, air quality, and air pressure. There are also color and light sensors, as well as an accelerometer. The board also features EEPROM memory and an additional module (nRF52840) that supports NFC and Bluetooth LE. It can be powered by USB or by a rechargeable Li-Po battery [20].

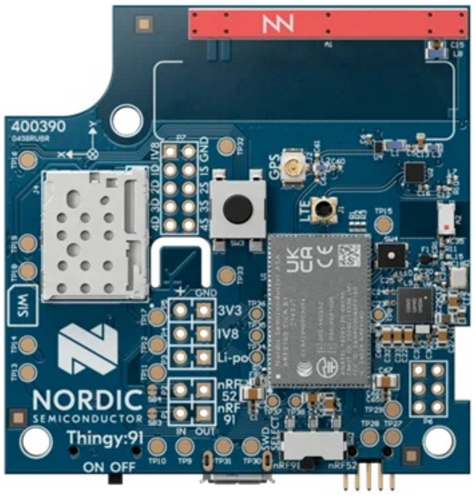


Figure 3.5: Nordic Thingy board [20]

3.5 Overall review

Out of the previously presented companies and NB-IoT solutions they offer, it can be seen that there are not many choices for users who want to leverage NB-IoT and develop solutions based on this protocol. Out of the mentioned companies, Quectel and Telit distinguish themselves as companies that possess manufacturing plants. This means that the products of these companies do not depend on third parties, at least not all products, as we will see. From design to the final solution, almost everything is done by these companies. On the contrary, Nordic Semiconductor is a fabless company, which means that all of the manufacturing is outsourced and dependent on a third party.

Starting with the BC65-TE-B development board from Quectel, this board is designed and manufactured entirely by Quectel, but it offers only NB-IoT connectivity and development of NB-IoT applications with no additional onboard sensors or typically needed GNSS support. It also lacks power supply options, providing only USB power supply and Arduino connector external power supply options.

The Bravo development board from Telit is similarly to the BC65-TE-B, designed and manufactured exclusively in-house, by Telit in this case. Nevertheless, some of the components are provided by different companies. This development board does offer onboard sensors, but these sensors are developed and manufactured by Bosch. This means that the Bravo development board depends on Bosch to provide the sensor components. Regarding the power supply options, it does provide an upgrade compared to the Quectel board in the form of a battery controller, which enables developing and testing battery-powered solutions. The module featured on the Bravo board also has support for GNSS.

Nordic actually goes a step further and offers two different development board solutions, one for the NB-IoT module entirely, and another also packed with sensors, more suitable for asset tracking IoT scenarios. The former, nRF9160 DK, features GNSS support as well as a Wi-Fi interface. The latter, Nordic Thingy:91, has support for GPS, an additional microcontroller enabling Bluetooth LE and NFC support, and can be battery-operated. The SoCs provided by Nordic are developed by them, but some of the other components, as well as the manufacturing of the final products, depend on third-party companies.

The emphasis on development and manufacturing, and essential components being managed by one company, is crucial for the ability to provide support for the users and customers of the products. Some of these solutions are expected to run on average for about 10 years. If some of the parts that make up the development boards were to become discontinued, for example, sensors on the Bravo platform, the whole product would be unavailable. The remedy would include modifying the solution to include entirely different sensor parts, which would also require investing in development and redesign.

The aim of this thesis was to address all of these shortcomings by providing a new development platform that offers NB-IoT and GNSS support. The platform would also include onboard sensors and external sensor support. The possibility to be battery-operated, as well as a distinguishing factor, the possibility to be powered by an energy harvesting unit that relies on solar panels and supercapacitor technology. Finally, support for easily gathering data from the sensors and also dashboard connectivity is given in the form of firmware examples, with the aim of easing software application development. All of these design choices were made to provide maximum flexibility for the users. The plan was also to use all of the components, from the module, MCU, and sensors, that were developed and manufactured, including the final development board as well, by one and the same company. This proved to be doable as detailed in the upcoming chapter because the work in this thesis was carried out in collaboration with STMicroelectronics.

Chapter 4

Methodology

4.1 Experimental setup

This section focuses on describing available hardware components (e.g., development boards, sensors, etc.) and software components that were used in the development of this project. Since the general plan was to incorporate only the components designed and manufactured by one vendor, in this case STMicroelectronics, the focus was placed on exploring the suitable components made by this company. This restrictions was only valid for components that were included in the final development board. For other hardware components that represent some equipment (e.g., measurements, testing) and that were not a part of the final product, but could be used to help in evaluating its characteristics or for testing, this restriction was omitted.

In continuation of this section the description of relevant hardware and software components that constitute the initial development setup is given. The aim was to describe in more details which components were selected and why. Ultimately, it is also showcased how these components were used, first for prototyping, and later for transitioning from the prototype to the final product.

4.1.1 ST87 NB-IoT module

The ST87M01 is a high-performance, fully programmable, ultra-compact, low-power, certified NB-IoT and GNSS industrial module series offering worldwide band coverage [21]. Some of the relevant features include [21]:

- LTE, category NB2, Release 15
- Up to DL: 127 kbps, UL: 159 kbps
- Ultralow power mode 1.2 μA typical (0.5 μA typical in power-off)

- eDRX and PSM support
- Ultra-compact size
- Embedded IoT internet protocols
- Up to 23 dBm power-out
- Optional eSIM GSMA compliant with an additional secure element
- Optional GNSS and A-GNSS
- LTE, category NB2, Release 15

The ST87M01 module family supports multifrequency bands with extended multiregional coverage, enabling almost complete worldwide NB-IoT data communication. In addition, the presence of the GNSS receiver allows support for multiple satellite constellations to address high-accuracy localization applications. The ultra-compact module form factor in an LGA package of only 10.6 mm x 12.8 mm (with 51 pins) makes the ST87M01 family the perfect choice for size-critical applications. Furthermore, thanks to the ultra-low power consumption and industrial qualification grade over the industrial temperature range, the ST87M01 family represents the best choice for a wide range of IoT applications, ranging from smart grid, energy smart metering, smart city, factory automation, industrial IoT, asset tracking, etc. The ST87M01 family also embeds Short Message Service and internet protocols for NB-IoT products, including TCP/IP, TLS/DTLS, CoAP, LwM2M, MQTT, and HTTP/HTTPS, which enable a broad set of IoT applications. Full support of PSM and eDRX mechanisms allows the ST87M01 family to achieve an extra-long battery life on a single-cell primary battery [21].

AT commands and response codes

In order to communicate with the ST87 module, to send commands and receive responses, a command format known as AT format is used. This format uses ASCII characters and is suitable for UART interfaces. AT commands have a specific format that needs to be respected. The AT format dictates that all commands start with a prefix "at" or "AT" (case insensitive), followed by a separator, the character "+", after which the name of the command (also case insensitive) is given along with the command body. The command body depends on the type of the command [22]:

- **Parameter type commands**: This type of command may be "set" (to store a value or values in the modem's memory for later use), "read" (to determine the current stored value or values), or "test" (to determine ranges of values supported). They all have a trailing "=" after the command's name, followed by the parameters. Each of these commands has a test command associated with it (trailing "=?") to provide information about the type of its subparameters, and also a read command (trailing "?") to check the current values of subparameters

- **Action type commands**: This type of command may be execute or test, with no trailing characters present. "Execute" commands invoke a particular function of the modem (e.g., enabling the HTTP stack), while "test" commands determine if subparameters are associated with the action, and if they are, return the ranges of subparameter values that are supported or indicate an error if the command has no subparameters

Finally, to indicate the command's end, a special character is used, in this case, a carriage return. The ST87 modem supports a standard set of AT commands specified by the 3GPP and also supports proprietary commands defined by ST. The proprietary AT commands format is the same as the standard one except the delimiter "+" is replaced with the delimiter "#".

An example of a set command where "cmd" represents the name of the command:

- **AT+cmd=0**: Here, "0" is a subparameter
- **AT+cmd?**: This is a read command for checking current subparameter values
- **AT+cmd=?**: This is a test command for checking possible subparameter values

After receiving a command, the modem will, after performing the indicated action, return "OK" to indicate success, or in case of an error or wrongly typed command, return "ERROR." Depending on the command, an error response can be a bit more detailed, with a format of "+CME ERROR" immediately followed by a numerical error code.

The above-mentioned responses are synchronous because they occur only after a command has been issued. Another important type of response is an unsolicited response code. These are present because of the need to send out asynchronous responses in some situations. For example, when sending a UDP packet, an unsolicited response code can be enabled to asynchronously indicate that the UDP packet was sent by the modem. These responses follow a simple format of "+" followed by the name of the response code and optionally can contain some additional information, displaying certain parameters. If that is the case, a trailing "=" is present with a comma-separated list of parameters.

An example of an unsolicited response code that notifies of a UDP packet being sent by the modem would be **+NBSEND**.

4.1.2 EVLST87M01 development board

The EVLST87M01 board is an evaluation board for the ST87M01 module. The board is designed as an X-NUCLEO Shield form factor compatible with the Arduino connector [23]. Figure 4.1 showcases the block diagram of the evaluation board.

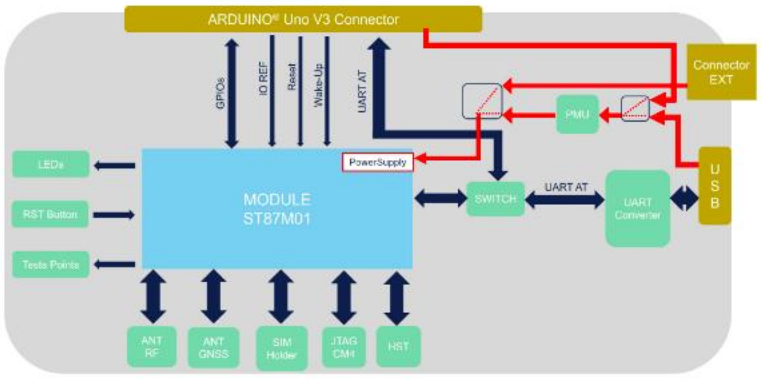


Figure 4.1: EVLST87M01 block diagram [23]

As can be seen from the figure above, the board can be powered either by USB, from the Arduino shield, or from some other external power supply. It features a UART switch that enables communication over USB or through the Arduino shield pins designated for UART. There is also the possibility to attach two antennas: one for the NB-IoT and another one for the GNSS.

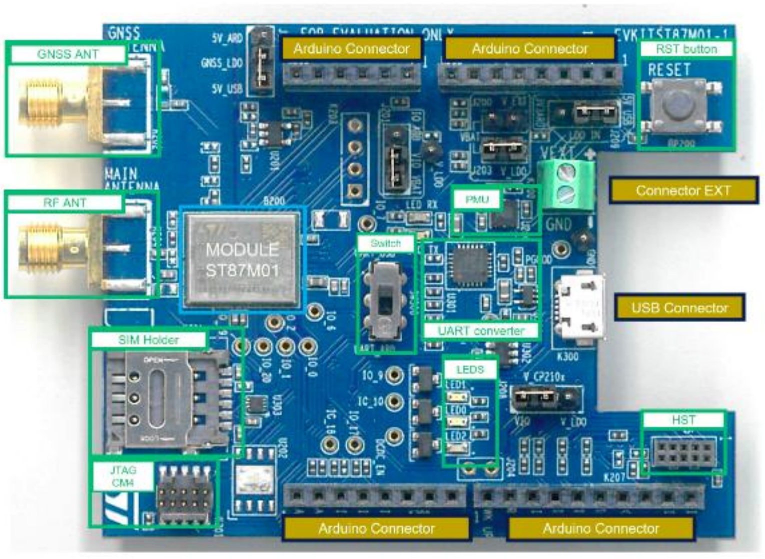


Figure 4.2: EVLST87M01 board [23]

Figure 4.2 provides an overview of the actual development board and marks the specific connectors and parts.

ST87Mxx GUI

The ST87Mxx GUI is a desktop application that provides a graphical interface to control and monitor the STM87M01 module [24].

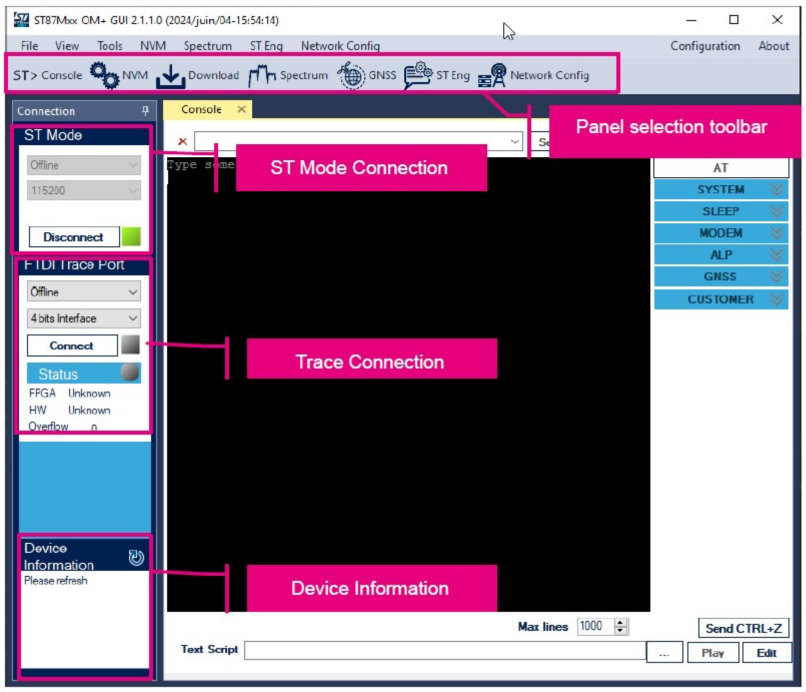
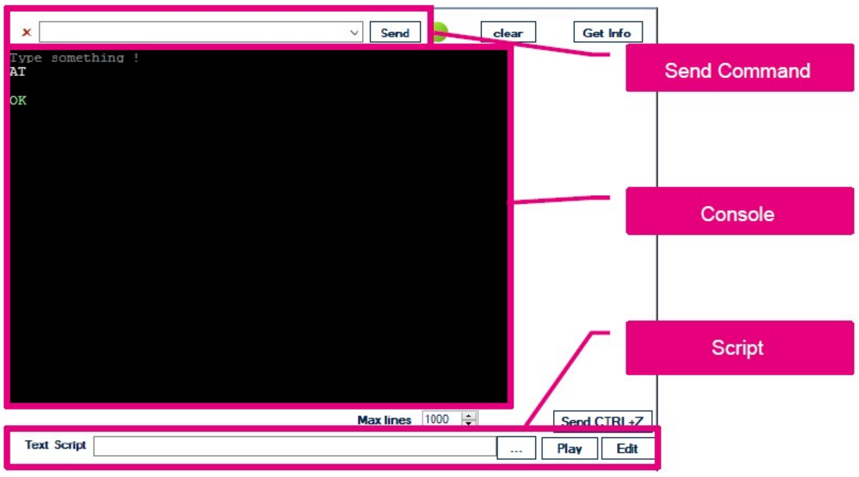


Figure 4.3: ST87Mxx GUI [24]

Figure 4.3 represents the application window that enables users to select and establish a connection with the ST87 module connected to the PC. In the panel selection toolbar, several different functionalities are presented. It is possible to select the console view to be able to send and receive AT commands. NB-IoT network configuration parameters are also available, along with views of the GNSS data, RF spectrum, and the possibility to download (to flash) new firmware onto the module. On the left, the connection panels are present, providing more information about the connected USB devices. There is also a device information section in the bottom corner that gives more information about the connected module, firmware version, RF parameters, etc.



```
Type something !
AT

OK
```

Figure 4.4: ST87Mxx GUI command window [24]

The command console view, as seen in Figure 4.4 above, gives users an interface for sending AT commands to the module as well as for showing the response messages from the modem. Commands to be sent can be inputted in the input box on top. Right below is the console view window, and at the bottom, there is an option to load a script, which is a regular file that has several AT commands one after another that are read and executed in the given order.

Easy Connect library

The Easy Connect library represents a software library that offers an API to interact with the ST87 module, thus enabling easier access to the module's resources and making the configuration of the module as convenient as possible [25]. The Easy Connect library handles the communication with the ST87 module via UART in an AT commands format. Standard AT commands are supported as well as extended commands defined by ST specific to the module. Since the handling of the commands is done by the library, the user does not need to worry about parsing responses or creating commands to send. This is all abstracted in the form of "sequences."

These so-called sequences handle the logic for sending a specific set of commands in a specific order to the module and parsing the adequate responses. The user is responsible for starting a sequence via an appropriate API call, except in the case of a "cold-start" sequence, which runs at the very beginning, checks if it needs to perform an update of supported configuration parameters for the module, and then either performs the configuration update or skips it if it is not needed. Some of the implemented sequences in the library are:

- **GNSS sequence**: Getting a position fix
- **TCP sequence**: Sending data over TCP
- **UDP sequence**: Sending data over UDP
- **HTTP(S) sequence**: Sending HTTP(S) requests

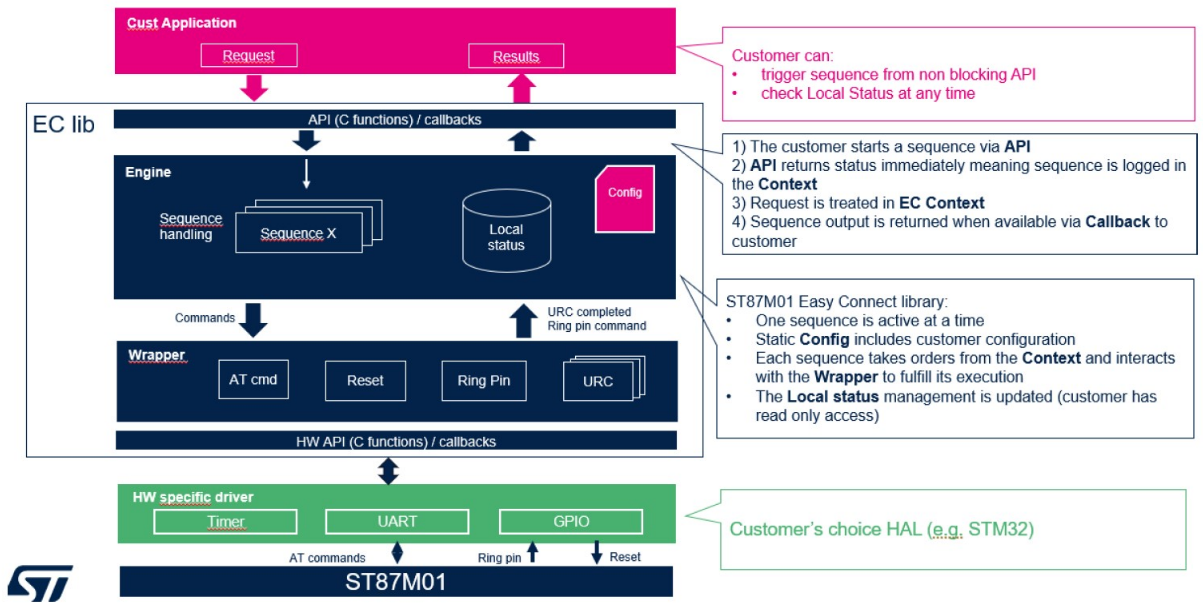

Cust Application
Request
Results
Customer can:
• trigger sequence from non blocking API
• check Local Status at any time
EC lib
API (C functions) / callbacks
Engine
Sequence handling
Sequence X
Local status
Config
1) The customer starts a sequence via API
2) API returns status immediately meaning sequence is logged in the Context
3) Request is treated in EC Context
4) Sequence output is returned when available via Callback to customer
Commands
URC completed
Ring pin command
ST87M01 Easy Connect library:
• One sequence is active at a time
• Static Config includes customer configuration
• Each sequence takes orders from the Context and interacts with the Wrapper to fulfill its execution
• The Local status management is updated (customer has read only access)
Wrapper
AT cmd
Reset
Ring Pin
URC
HW API (C functions) / callbacks
HW specific driver
Timer
UART
GPIO
Customer's choice HAL (e.g. STM32)
AT commands
Ring pin
Reset
ST87M01

Figure 4.5: Architecture of the Easy Connect library

Figure 4.5 showcases the architecture of the library. The library serves as a middleware between the hardware and the application logic. It relies on a timer peripheral for managing the timeout when communicating with the modem, and a UART peripheral to manage the communication and data exchange as already mentioned. The exposed API provides a simple interface with all of the complex logic abstracted away and handled by the library in the form of sequences.

4.1.3 STM32U5 Nucleo

The STM32U5 Nucleo board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features provided by the onboard STM32U5 microcontroller. The ST Zio connector, which extends the Arduino connectivity, and the ST morpho headers provide an easy means of expanding the functionality of the Nucleo open development platform with a wide choice of specialized shields. The STM32U5 Nucleo board does not require any separate probe as it integrates the ST-LINK debugger/programmer. The board also comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube MCU Package, which greatly eases the software development process [26].

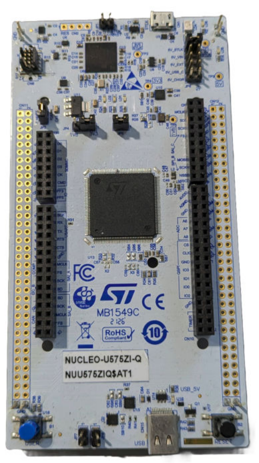


Figure 4.6: STM32U575 Nucleo development board

Common features of the board include [26]:

- STM32 microcontroller
- 3 user LEDs
- 2 user and reset push-buttons
- 32.768 kHz crystal oscillator
- Serial Wire debugging interface
- ST Zio expansion connector including Arduino and ST morpho expansion connector

- Flexible power-supply options: USB connector or external sources

STM32U575 microcontroller

The microcontroller featured on the STM32U5 board used in this project is the STM32U575 microcontroller. The STM32U575 belongs to an ultra-low-power microcontroller family (STM32U5 series) based on the high-performance Arm® Cortex®-M33 32-bit RISC core. It can operate at a frequency of up to 160 MHz. The Cortex®-M33 core features a single-precision FPU that supports all the Arm® single-precision data-processing instructions and all the data types. The Cortex®-M33 core also implements a full set of DSP instructions and a memory protection unit that enhances application security.

The MCU embeds high-speed memories, 2 Mbytes of flash memory and 786 Kbytes of SRAM, and an extensive range of enhanced I/Os and peripherals connected to three APB buses, three AHB buses, and a 32-bit multi-AHB bus matrix. The MCU offers one fast 14-bit ADC (2.5 Msps), one 12-bit ADC (2.5 Msps), two comparators, two operational amplifiers, two DAC channels, an internal voltage reference buffer, a low-power RTC, four 32-bit general-purpose timers, two 16-bit PWM timers dedicated to motor control, three 16-bit general-purpose timers, two 16-bit basic timers, and four 16-bit low-power timers. The MCU also features standard and advanced communication interfaces such as [27]:

- Four I2Cs
- Three SPIs
- Three USARTs
- Two UARTs
- One low-power UART
- Two SAIs
- Digital camera interface
- Two SDMMCs
- One FDCAN
- One USB OTG full-speed
- One USB Type-C /USB Power Delivery controller
- One generic synchronous 8-/16-bit PSSI

A comprehensive set of power-saving modes allows the design of low-power applications. Many peripherals (including communication, analog, timers, and audio peripherals) can be functional and autonomous down to Stop mode with direct memory access, thanks to low-power background autonomous mode support.

Some independent power supplies are supported, such as an analog independent supply input for ADC, DACs, OPAMPs, and comparators, a 3.3 V dedicated supply input for USB, and up to 14 I/Os that can be supplied independently down to 1.08 V. A VBAT input is available for connecting a backup battery to preserve the RTC functionality and to backup registers and 2-Kbyte SRAM [27].

Power Management

The STM32U575 features flexible power control options, which increases flexibility in power mode management and further reduces overall power consumption. The STM32U575 supports dynamic voltage scaling to optimize power consumption in Run mode. The voltage from the main regulator that supplies the logic can be adjusted according to the system's maximum operating frequency. The consumption is even lower when frequency and voltage are decreased. The STM32U575 supports several low-power modes, with the main focus on the so-called Stop modes [11]:

1. **Stop 0**: The voltage regulator is configured in main regulator mode, all clocks and oscillators are disabled except the internal or external low-speed oscillator (RTC can be powered as well by either of the two), the brown-out reset is enabled, and most of the peripheral clocks are gated off
2. **Stop 1**: Similar to Stop 0 except that the power consumption is lower as the main regulator is stopped and replaced by the low-power regulator
3. **Stop 2**: Similar to Stop 1, but here most of the peripherals are put into a lower leakage mode. Some peripherals with LPBAM capability can switch on. All SRAMs and register contents are preserved, but the SRAMs can be totally or partially switched off to further reduce consumption
4. **Stop 3**: Similar to Stop 2, but in this mode, the I/Os are in a floating state by default, and functional peripherals and sources of wakeup are reduced

Each mode can be configured in many ways, providing several additional sub-modes. In addition, it offers support for a battery backup domain, called VBAT [11].

Low-power background autonomous mode

The Low-power background autonomous mode is an operating mode available in the STM32U5 product series that allows peripherals to be functional and autonomous independently from the device power modes without any software running. While the Cortex-M33 is asleep, some parts of the microcontroller remain active and can perform background tasks that do not require software assistance. The two DMA controllers, LPDMA and GPDMA, not only transfer data but can also be initialized to access control registers in order to implement complex sequences involving peripherals and memory. For example, a timer can trigger a periodic task that consists of acquiring samples from an ADC, moving these samples to memory, and monitoring the sampled signal to detect any abnormal condition [11].

4.1.4 One example of how DTDL is used at ST: Platform Virtualization

In a scenario where some boards (development boards or custom solutions) that run specific (different) firmware could interact with different applications through different communication channels, as displayed in Figure 4.7, it is complex to implement end-to-end applications without a common standardized strategy. In general, without a common approach, each developer could define a custom protocol, but this could lead to certain problems. Re-writing code and errors when re-inventing methodologies could increase the overall complexity of the system. By introducing the concept of Platform Virtualization, which is made up of two different aspects, this can be avoided [28]:

- **Virtual devices**: Represent the abstraction of a physical device and describe its capabilities and behaviors
- **Standard protocols**: Allow avoiding the creation of custom application-based protocols and reusing the same commands/messages in different protocols over different communication channels

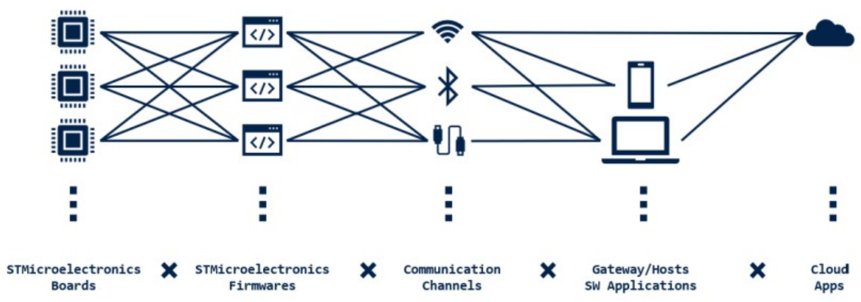

STMicroelectronics Boards
STMicroelectronics Firmwares
Communication Channels
Gateway/Hosts SW Applications
Cloud Apps

Figure 4.7: Interconnecting heterogeneous devices [28]

Platform Virtualization is built on top of the following principles:

- **Device template**: Standard description of a device and its capabilities, specified through the DTDL language

- **Device catalog**: Centralized database to collect device definitions (templates), with a unique key for identifying an entry consisting of board and firmware ID

- **Protocols**: "Azure PnP" messages when communicating with the cloud over the Internet, and "PnP-like messages" when communicating with the local device (over USB, for example)

Architecture

The complete architecture is displayed in Figure 4.8. "Azure PnP" messages are used when the application (desktop, web, or mobile) is communicating with the cloud, in this case, the Azure cloud platform. Once the data is in the cloud, it can be processed and stored for later use, usually for training neural network models, but that is just one of the possible use cases. In order for the application to know the capabilities of the sensor (or sensors) with which it communicates and gathers data from, a device template is required. This device template can be obtained from the cloud or from local storage. Once the sensor communicates the board and firmware ID to the application, the proper GUI of the application is loaded.

The exchange of data between the application and the sensor (or sensors) is facilitated using "PnP-like messages." This lets users correctly interface with the sensor and control it. A device template represents a model of a specific device/sensor and is defined by means of the DTDL. It is made up of a list of interfaces, where each interface can define a collection of device properties, commands, or telemetry types. As an example, it is possible to imagine a device model as a list of sensor interfaces, each of which defines its properties (e.g., output data rate), and also its telemetries and commands [28].

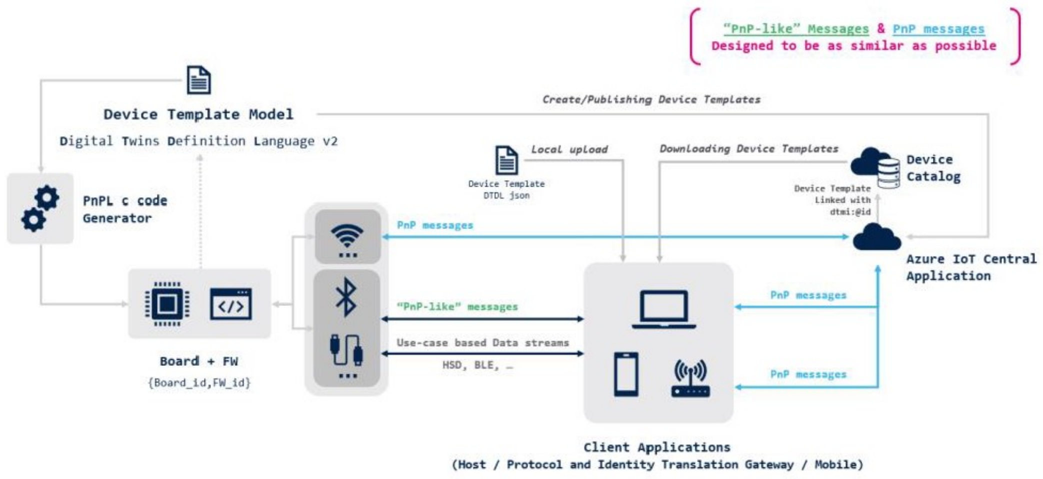

"PnP-like" Messages & PnP messages
Designed to be as similar as possible
Device Template Model
Digital Twins Definition Language v2
Create/Publishing Device Templates
Local upload
Device Template DTDL json
Downloading Device Templates
Device Catalog
Device Template Linked with dtmi:@id
PnPL c code Generator
PnP messages
Azure IoT Central Application
"PnP-like" messages
Use-case based Data streams
HSD, BLE, …
Board + FW
{Board_id,FW_id}
PnP messages
PnP messages
Client Applications
(Host / Protocol and Identity Translation Gateway / Mobile)

Figure 4.8: Platform Virtualization architecture [28]

Azure PnP messages

Azure PnP messages are DTDL format messages structured as a set of elements that define [29]:

- **Properties**: Represent the read-only or writable state of a device or other entity
- **Telemetry**: The data emitted by a device, whether the data is a regular stream of sensor readings, an occasional error, or an information message
- **Commands**: A function or operation that can be performed on a device

Azure PnP-like messages

These messages have been defined with the purpose of creating a common language to configure a device (sensor), retrieve information or data from it, or control its behavior in a way that could be "communication interface-independent." Another important point considered in the design process of these messages was to create them as similar as possible to the Azure PnP messages. This constraint was introduced to minimize the effort of converting between the two formats. When communicating with the sensors, getting telemetry data, or sending a command and then communicating back with the cloud, minimal conversion between messages is necessary. There are several types of PnP-like messages available [28]:

- **SET property**: These messages allow the update of one or more properties (defined as writable) of a device model component

- **GET property**: These messages are used to get the current values of specific properties from the board
- **Command**: A function or operation that can be performed on a device, with the same function as in the regular Azure PnP messages, just in a slightly different message format
- **Telemetry**: When a device model component includes a telemetry field, these messages are used to send relevant data or events from a device to the application (web, mobile, desktop)

In Figure 4.9, a basic exchange of Azure PnP-like commands between a desktop application and a sensor can be seen.

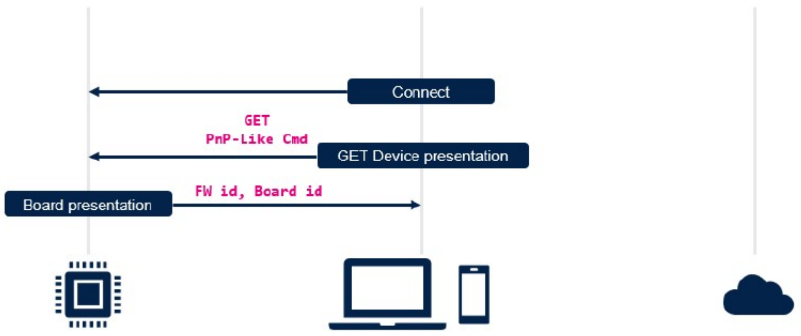


Figure 4.9: Azure PnP-like messages exchange between a desktop application and a sensor connected via USB [28]

One of the goals of firmware development was to make it easier for users to obtain the data from the sensors. This meant creating a firmware that supports the ST's Platform Visualization infrastructure. More precisely, the desktop data logging application and the web based dashboard. The following two subsections give more details about these two components.

4.1.5 ST High Speed Datalog

The ST High Speed Datalog is a sensor data capture and visualization toolkit. The toolkit consists of firmware and desktop applications. The firmware is structured into a set of layers. At the lower level, the HAL interfaces with the hardware and provides the low-level drivers and the hardware interfacing methods used by the upper layers (application layer, for example). A BSP is also provided, which deals with the board-specific peripherals and functions (LED, user button, etc.). The application level of the firmware is based on an eLooM framework.

eLooM stands for embedded light object-oriented framework. It is an application framework designed for STM32 devices, for soft real-time, event-driven, multitasking, and low-power embedded applications. The framework relies on the presence of an RTOS, specifically important are the tasks that provide logical abstraction for offered functionalities. Each firmware module implements or extends services, classes, and objects made available by the eLooM framework [30]. An important eLooM module in this firmware is the SensorManager module. SensorManager is an eLooM-based application-level module that interfaces with sensors and offers their data to other application modules. It is implemented as an acquisition engine that [31]:

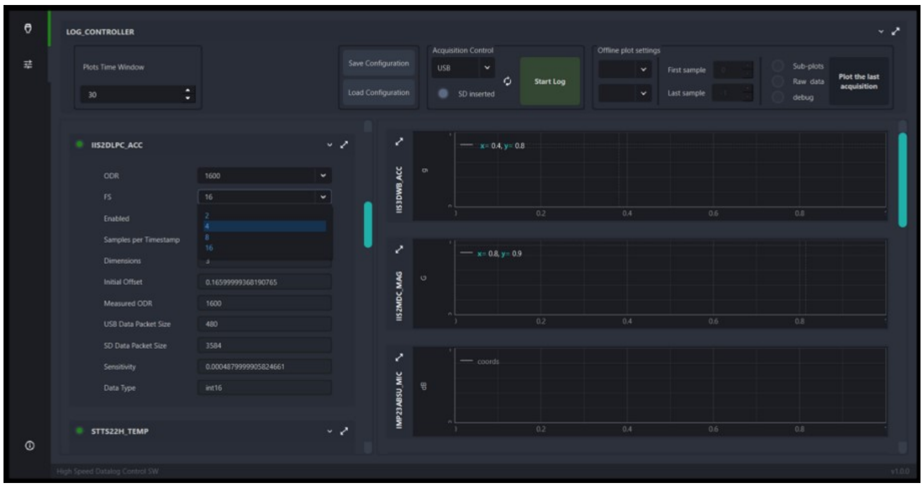
- Provides a task for each sensor that needs to be managed (this task controls the configuration and data acquisition from the sensor)
- Orchestrates task accesses to sensor bus data
- Dispatches events to notify when data has been collected from the sensors and is available to be sent to the PC

The firmware generally manages the sensors and gathers data. As soon as the data from the sensors is available, it is sent to the PC where it is saved and can also be visualized by the desktop application.

As far as the desktop application goes, it enables establishing a connection with the board, and once the connection is established, it provides the possibility to, through the GUI [30]:

- Enable or disable a specific sensor on the board
- Change output data rate, full scale, and other relevant parameters of the sensors
- Start and stop logging data and saving it on the PC

Figure 4.10 below demonstrates the GUI.



Plots Time Window: 30

Acquisition Control: USB

IIS2DLPC_ACC

ODR	1600
FS	16
Enabled	
Samples per Timestamp	
Dimensions	3
Initial Offset	0.16599999368190765
Measured ODR	1600
USB Data Packet Size	480
SD Data Packet Size	3584
Sensitivity	0.0004879999905824661
Data Type	int16

FS options: 2, 4, 8, 16

Figure 4.10: Datalog desktop GUI application [30]

4.1.6 DSH-ASSETRACKING

Devices that keep track of the environment in which they are placed, using a sensor or, in most cases, several sensors (temperature, humidity sensor, etc.), are capable of outputting data but not showing it in a manner that is "user-friendly". Usually, these devices have enough processing power to gather the data and then send it over some interface, whether it is a point-to-point protocol or a network interface. In general, users are interested in the data produced by these devices and how that data changes over time. For this purpose, a device with adequate processing power, like a PC or a cloud-based server, is included to accumulate the data and visualize it in a user-friendly manner.

The DSH-ASSETRACKING dashboard is a cloud application powered by Amazon Web Services. It provides a functional and intuitive interface created for the purpose of data collection, visualization, and analysis. It supports several different motion (e.g., accelerometer) and environmental (e.g., temperature) sensor data types as well as the capability of showing position data (e.g., GNSS). It can be used to plot and graph real-time or historical position data and sensor values and to monitor operating conditions such as running temperature and events. It provides the following features [32]:

- Registering a new device
- Connecting the new device to the cloud application
- Checking the device status and its network connection
- Viewing telemetry data sent by the device

- Selecting a device and which data to plot/show on the chart
- Filtering data by choosing a specific time window
- Geolocating the device

Figure 4.11 presents the main page overview of the dashboard application. There are several tabs that can be accessed, for monitoring the devices and creating them, to checking the telemetry from sensors and data about the position of the device.

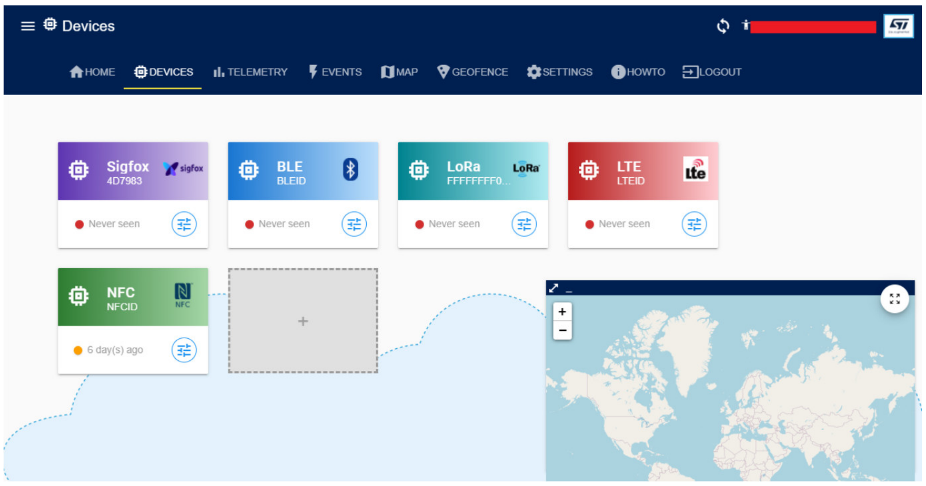


Figure 4.11: Main page of the DSH-ASSETRACKING web dashboard [32]

Once the telemetry tab is selected, a telemetry page renders, enabling users to select which device and which data they want to show on the graph, as can be seen in Figure 4.12. A filter is also available that can be configured to select the desired time window. Only the data from this time window will be shown.

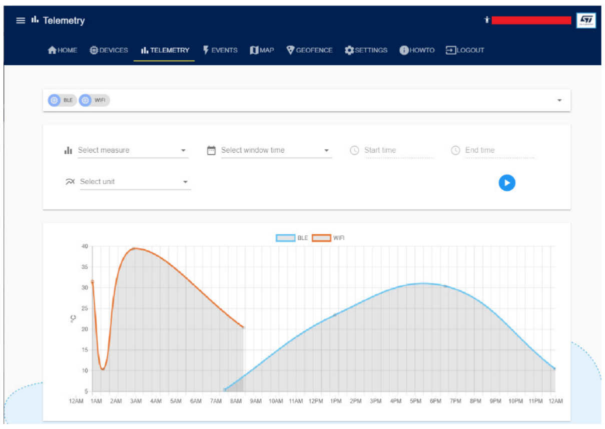


Figure 4.12: Telemetry page of the DSH-ASSETRACKING web dashboard [32]

4.1.7 Sensors

To get a sense of the environment and track what is happening and how physical quantities change, people have relied on analog sensors, and with the development of electronics, on digital sensors as well. An example of tracking how physical quantities in the environment change would be adjusting the heater during winter months. The temperature sensor that comes with the heater provides insight on how to adjust the heater's output. This is just one simple scenario of how sensors can be useful. There are many more physical quantities that can be measured by sensors, such as humidity, atmospheric pressure, acceleration, and magnetic field strength. The usage of a specific type of sensor depends on the situation.

For the purposes of asset tracking, for example, it is beneficial to get a sense of the environment in which the current asset is residing. This could involve tracking physical quantities such as temperature, humidity, and even placing an accelerometer to determine the orientation of the asset or its movement. This helps to determine how the environment affects a certain asset and to evaluate the state of the asset in certain environmental conditions. Since asset tracking is of interest for this project, the choice of sensors was directly influenced by this fact. To be more precise, the following digital sensors were used:

- **LIS2DUXS12**: An accelerometer, used for determining movement and orientation
- **LPS22DF**: - A barometer, used for determining atmospheric pressure
- **SHT40**: A temperature and a humidity sensor

LIS2DUXS12

The LIS2DUXS12 is a smart, digital, 3-axis linear accelerometer designed to combine the lowest supply current possible with features such as always-on antialiasing filtering, a machine learning core with adaptive self-configuration, and an analog hub sensing channel. The LIS2DUXS12 has user-selectable full scales of $\pm2g/\pm4g/\pm8g/\pm16g$ and is capable of measuring accelerations with output data rates from 1.6 Hz to 800 Hz. The device has a dedicated internal engine to process motion and acceleration detection, including free-fall, wake-up, single/double/triple-tap recognition, activity/inactivity, and 6D/4D orientation [33].

LPS22DF

The LPS22DF is an ultracompact, piezoresistive, absolute pressure sensor that functions as a digital output barometer. The LPS22DF provides lower power consumption. The sensing element, which detects absolute pressure, consists of a suspended membrane manufactured using a dedicated process developed by ST. The LPS22DF package is holed to allow external pressure to reach the sensing element [34].

SHT40

The SHT40 is a digital sensor for measuring relative humidity and temperature at different accuracy classes. It was designed to support maintaining an ultra-low power budget (0.4 µW). The power-trimmed internal heater can be used at three heating levels, thus enabling sensor operation in demanding environments [35].

4.1.8 Power Profiler Kit II

The Power Profiler Kit II is an affordable, flexible tool that measures the real-time power consumption of any external board. It achieves this by either providing power to the external board or acting as an ampere meter. The range of measured current goes from 500 nA to 1 A. The hardware is delivered with a GUI application that simplifies the usage of the Power Profiler Kit [36].

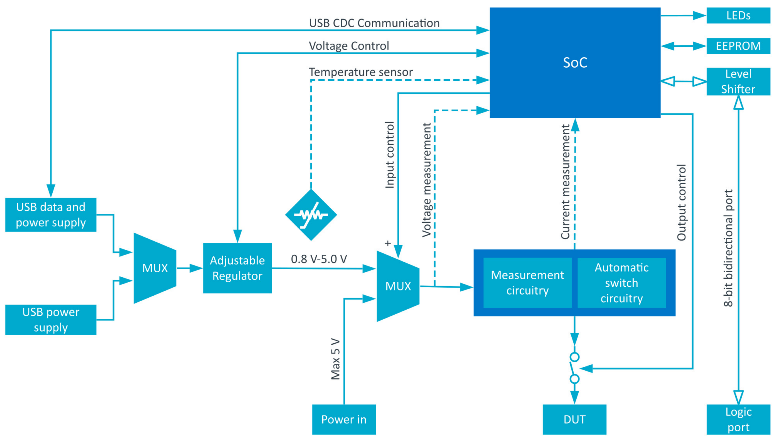


Figure 4.13: The Power Profiler Kit II block diagram

The figure 4.13 above gives an overview of the architecture of the device. The current measurement is performed by the nRF52840 SoC (referred to as just SoC in the figure above), which uses its ADC to measure a voltage drop over a series of measurement resistors. Resistor values are used to calculate the power consumption. Five different measurement ranges are supported, which are managed by an automatic switch circuitry [36].

The board for which the power consumption is to be estimated can be powered by the Power Profiler Kit device (Source Meter mode) or from an external power source (Ampere Meter mode). Figure 4.14 showcases the connections between the PC, the Power Profiler Kit, and the device for which the power consumption is to be estimated in Ampere Meter mode.

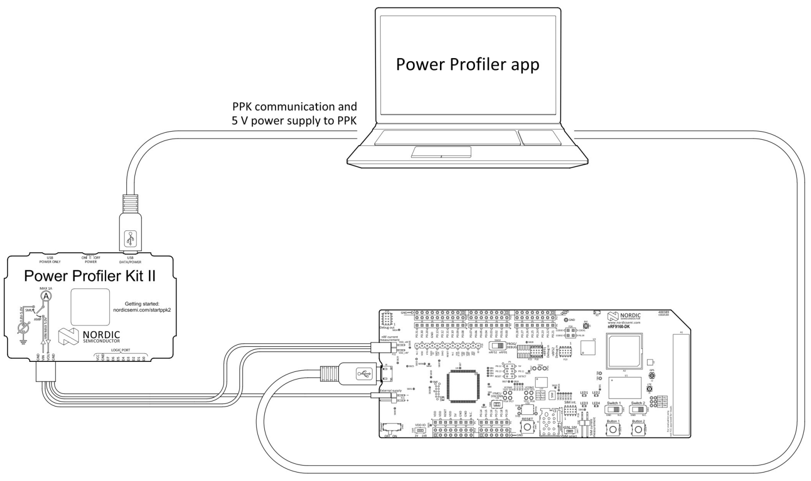


Figure 4.14: The Power Profiler Kit II Ampere Meter mode

This tool proved valuable for measuring power consumption for different firmware examples.

4.2 Prototyping

The initial work was started with a setup that consisted of several development boards listed below:

- **NUCLEO-U575ZI-Q**: STM32 Nucleo development board with STM32U575 MCU [26]
- **EVLST87M01**: Evaluation board for low-power NB-IoT module with GNSS capability [21]
- **X-NUCLEO-PGEEZ1**: Evaluation board featuring SPI EEPROM memory [37]
- **X-NUCLEO-IKS4A1**: Environmental sensors and motion MEMS evaluation board [38]

The evaluation boards were stacked on top of one another and connected via Arduino connectors as displayed on figure 4.15.

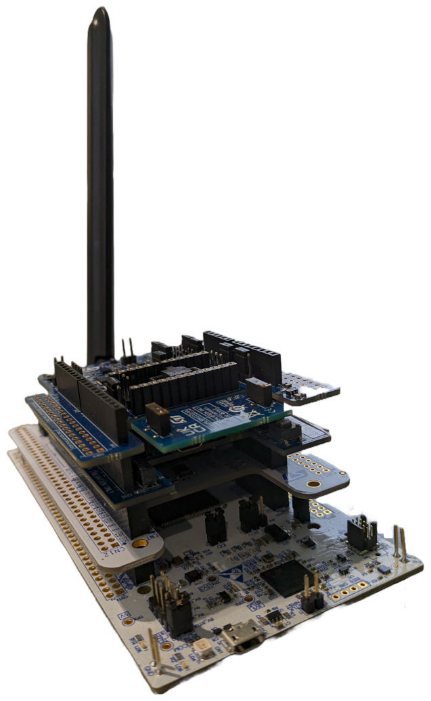


Figure 4.15: Development boards stacked one onto another via Arduino connectors

From bottom to top, the "stack" consists of the following development boards:

- NUCLEO-U575ZI-Q
- X-NUCLEO-PGEEZ1
- EVLST87M01
- X-NUCLEO-IKS4A1

This was the latest setup used for firmware development before the first STEVAL-NBIOT prototype was manufactured. Initially, the prototyping work was started by using only the EVLST87M01 evaluation kit, which was connected to the PC via USB.

4.2.1 Trying out the features of the EVLST87M01

Testing out the NB-IoT features and gaining some practical experience with the protocol proved to be straightforward. This was all thanks to the EVLST87M01 evaluation/development kit in combination with the official GUI application made to ease its use. Figure 4.16 represents the setup that was used for trying out the example explained in this section. The setup consisted of the EVLST87M01 development board with an LTE antenna attached. The EVLST87M01 was connected to the PC via USB (note that the cable connections were omitted from the figure).

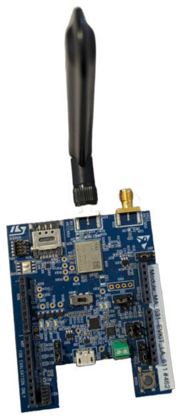


Figure 4.16: EVLST87M01 development board with LTE antenna

After connecting the evaluation kit to the PC and with the help of a reference command manual, tinkering with the module began straight away. The very first test scenario involved sending a payload using the UDP protocol, which was then echoed back by the server. The payload could be anything, but in this case, an ASCII payload was selected with the content "Hello World." This scenario could be perceived as a sort of standard "Hello, World!" example. Usually, a "Hello, World!" program just prints "Hello, World!" and is used to illustrate an example of basic syntax for some programming language. Here, it signifies a basic example or scenario of how to use a module.

The steps for the scenario that was tested included waiting first for the module to establish a connection to the NB-IoT base station. After the connection was established, a UDP socket had to be created. Using this socket, only a single AT command was necessary to initiate a packet transfer. What can be observed is that this abstraction with AT commands and socket creation is similar to the standard network programming paradigm that also makes use of sockets.

4.2.2 Using EVLST87M01 together with the NUCLEO-U575ZI-Q

After getting to know how the EVLST87M01 is used and some of its features, it was time to try controlling and communicating with it using another MCU. The MCU used was the STM32U575, as part of the NUCLEO-U575ZI-Q development board showcased in figure 4.17. Since this development board features Arduino connectors, which are also available on the EVLST87M01, they were used to interconnect these two boards, most importantly their UART peripherals. The EVLST87M01 was placed on top of the NUCLEO-U575ZI-Q development board, as shown in figure 4.17.

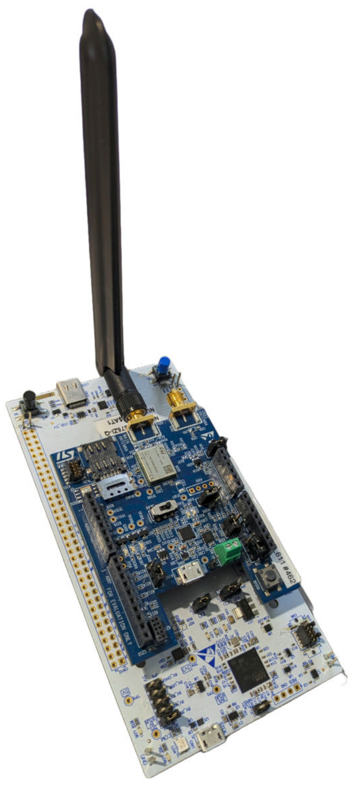


Figure 4.17: EVLST87M01 placed on top of the NUCLEO-U575ZI-Q Nucleo development board

Since, like in the previous scenario with the GUI application, the module expected AT commands over UART, the steps to send the network packet remained the same. The steps here refer to the AT commands. The only difference now was that instead of typing the AT commands and sending them over the GUI, the commands were created programmatically in firmware and sent from the STM32U575 microcontroller using its UART peripheral.

At first, the firmware example used for this scenario, of sending a UDP packet and receiving the echo from the server, was done without using additional libraries. Shortly after that, the first release of the Easy Connect library was available internally at the company, making it possible to interact with the EVLST87M01 in an easier way. With the help of the library, only a couple of API calls were necessary to reproduce the scenario; the Easy Connect library took care of the rest. All the tedious command assembling and response parsing of data from UART were hidden away from the user.

4.2.3 Sensing the outside world (X-NUCLEO-IKS4A1)

With the basic scenario being successfully realized in ways that were previously described, it was time to introduce the possibility of sending some concrete and useful information over the network. What is meant here is that the arbitrarily chosen ASCII data was now to be replaced by more useful data, in this case, data from the sensors.

The tested scenario that included data from the sensors was not so different from the one already tried, here referred to as the "Hello, World!" example. Namely, the data was still sent in ASCII format using the UDP protocol, and the server was echoing back the message. But the actual payload content now depended on which sensor was active. Depending on that, it could include temperature, humidity, atmospheric pressure, or accelerometer data, and sometimes even a combination of these. The development board of choice that included all of the sensors that can provide the aforementioned data and was used for this part of the development was the X-NUCLEO-IKS4A1. The actual development board can be seen in figure 4.18 below.

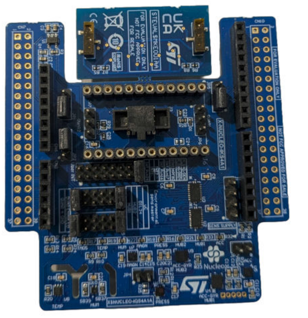


Figure 4.18: X-NUCLEO-IKS4A1 development board

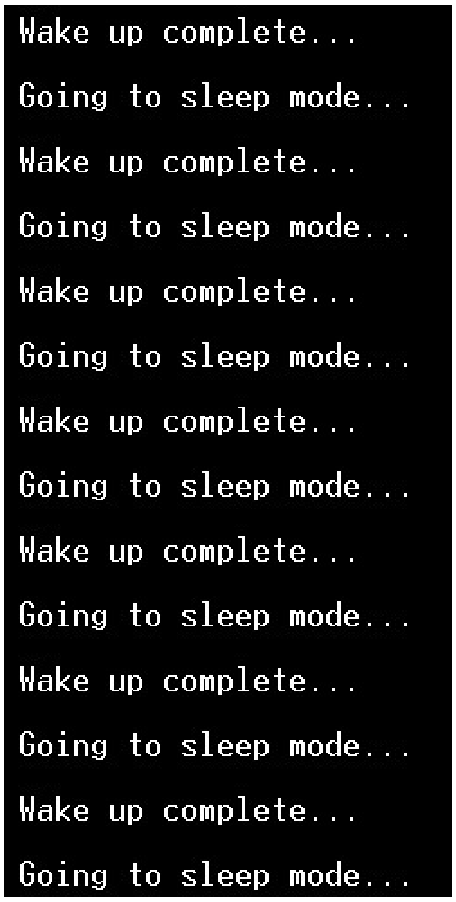
The concrete example presented here details how the accelerometer was configured to recognize and notify about the tap events to the STM32U575 MCU. Similar to the tap-to-wake function on mobile phones, the idea here was to detect either a single, double, or triple tap, send the payload to the server, and get the echo back. While waiting for the tap event from the accelerometer, the STM32U575 was supposed to be in a low-power mode. The firmware relied on the Easy Connect library to communicate with the module and was organized into several FreeRTOS tasks. One task was responsible for running the Easy Connect library main scheduler, another for configuring and reading the accelerometer data, while the third one was used to receive the sensor data and initiate the UDP transfer.

The first step was, of course, to decide which low-power mode to use and to explore how to put the MCU in this mode. The state that provided the lowest power consumption but had a certain amount of flexibility, such as the usage of standard GPIO pins for external interrupt events and being able to provide power to GPIOs even when sleeping, was STOP2 mode. The STM32 HAL library also provided a simple function call that enabled users to put the MCU into this mode. Upon making this decision, the next step was to determine how to use FreeRTOS to detect when the MCU could go to sleep and how to configure it for maximum power savings. For this purpose, the tickless idle feature of FreeRTOS was taken advantage of. After enabling this feature, two functions had to be defined to specify what should happen upon entering and exiting the sleep mode.

Every time no task other than the idle task was running, FreeRTOS would invoke the function that was supposed to put the system into low-power mode. This function was defined to print a message over the UART-to-USB interface of the ST-LINK debugger to ease the tracking of the system execution. Then it would stop the SysTick timer and enter the STOP2 mode, configuring it in such a way that only some interrupt could wake the system up.

The second function that ran after the system woke up reconfigured all the system clocks, because they are stopped when entering the STOP2 mode, resumed the SysTick timer, and printed a message about the wake-up event, the same way as the first function, using the debugger.

One small hurdle that was encountered at first was that when the system went to sleep, it would immediately wake up, as demonstrated by the trace in figure 4.19.



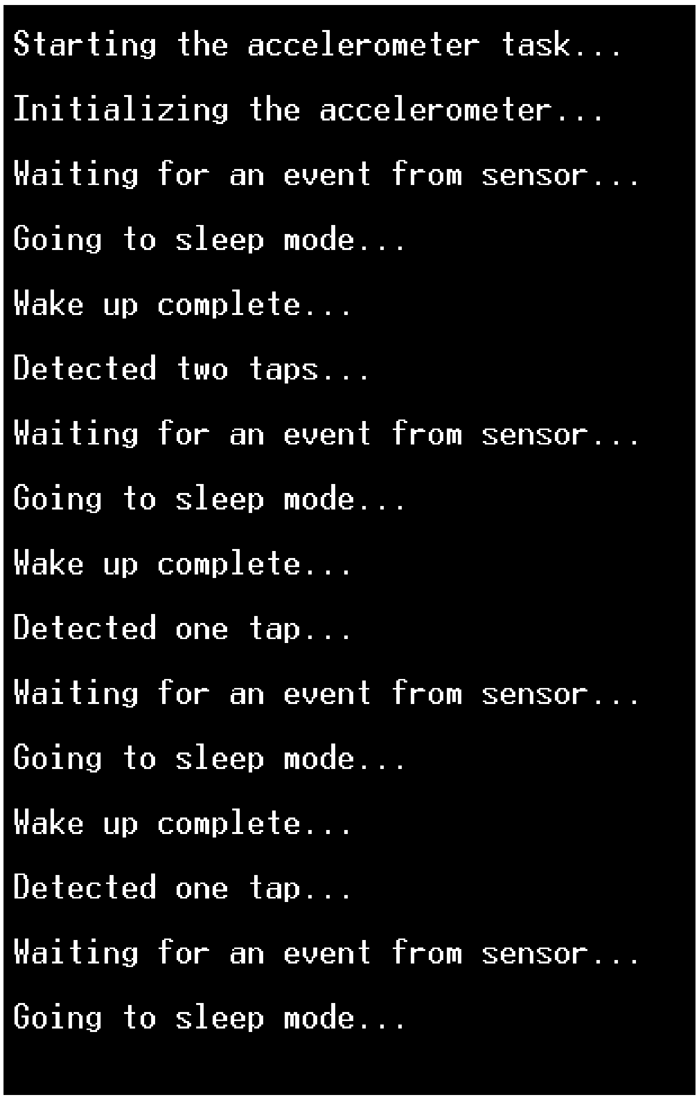
```
Wake up complete...

Going to sleep mode...

Wake up complete...

Going to sleep mode...

Wake up complete...

Going to sleep mode...

Wake up complete...

Going to sleep mode...

Wake up complete...

Going to sleep mode...

Wake up complete...

Going to sleep mode...

Wake up complete...

Going to sleep mode...
```

Figure 4.19: Sleep/wake-up trace IRQ pending problem

Upon inspecting the core registers of the MCU, it was observed that the SysTick interrupt was indeed disabled, but the previous interrupt was still pending. Instead of stopping and clearing the pending interrupt, only the former was accomplished. The MCU would go to sleep, but because of the pending interrupt, it would immediately wake up. This was remedied by clearing the pending interrupt after stopping the SysTick.

After successfully putting the MCU to sleep and waking it up, it was time to set up an interrupt source that could actually wake it up. The previous example would only put the MCU to sleep indefinitely. For this purpose, the accelerometer was configured to recognize the tap events—all three of them—on the Z axis, meaning that the board would lay flat and could be tapped on top. When the event was recognized by the accelerometer, the interrupt line between the sensor and the MCU would go high, signaling that an event had occurred. This would raise an interrupt to the STM32U575 while in STOP2 mode, after which the MCU would wake up, initialize the system clocks again, and resume the SysTick timer.

In the callback for the external GPIO interrupt, for the pin on which the sensor's interrupt line was connected, the accelerometer task was resumed. Upon resuming, this task would read the event source from the sensors and, depending on the tap event, send an ASCII message to the task that initializes the UDP transfer. Before actually incorporating the UDP and Easy Connect main scheduler tasks, figure 4.20 showcases the flow indicating the system going to sleep and waking up when there is an event from the sensor.

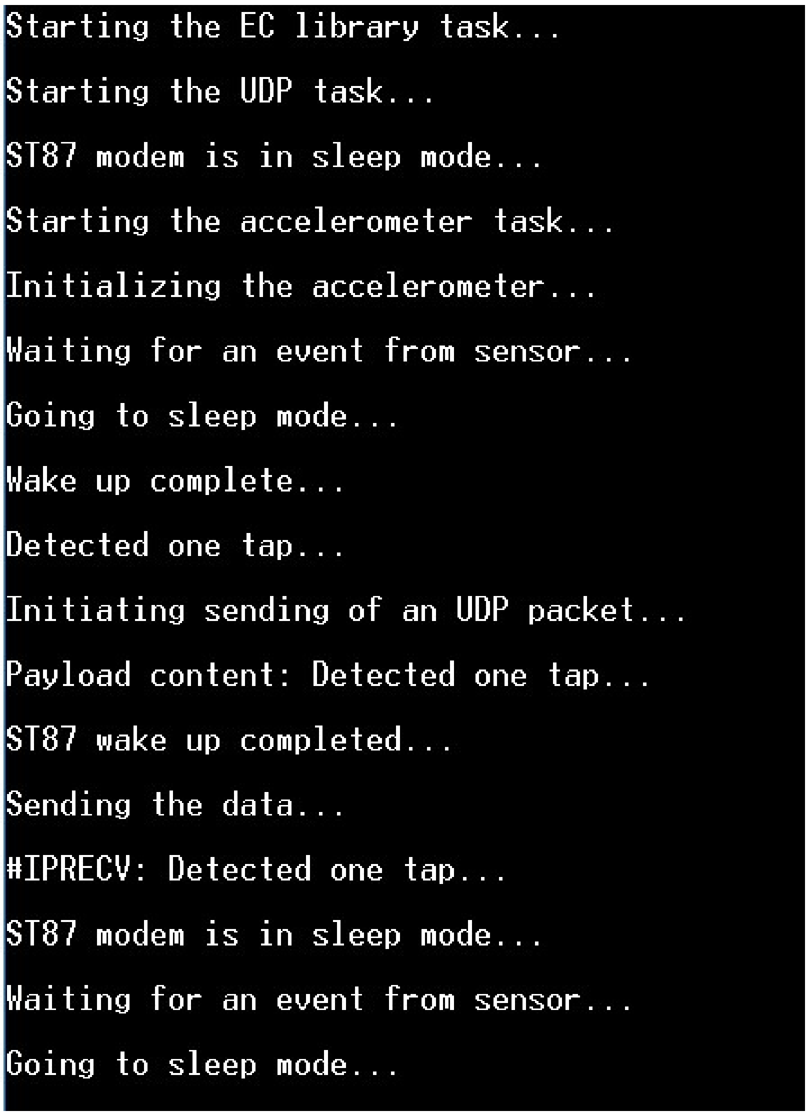


```
Starting the accelerometer task...

Initializing the accelerometer...

Waiting for an event from sensor...

Going to sleep mode...

Wake up complete...

Detected two taps...

Waiting for an event from sensor...

Going to sleep mode...

Wake up complete...

Detected one tap...

Waiting for an event from sensor...

Going to sleep mode...

Wake up complete...

Detected one tap...

Waiting for an event from sensor...

Going to sleep mode...
```

Figure 4.20: Sleep/wake-up trace with accelerometer sensor tap detection

The UDP transfer task would initialize the transfer using the Easy Connect library and pass control to the task that runs the main scheduler of the Easy

Connect library. This task was responsible for actually executing the UDP transfer sequence and dealing with all of the responses from the modem. Figure 4.21 provides an overview of how the example operates. The MCU would go to sleep, and when woken up by the event from the sensor, it would initiate the UDP data transfer with the payload indicating the event type: single, double, or triple tap.



```
Starting the EC library task...

Starting the UDP task...

ST87 modem is in sleep mode...

Starting the accelerometer task...

Initializing the accelerometer...

Waiting for an event from sensor...

Going to sleep mode...

Wake up complete...

Detected one tap...

Initiating sending of an UDP packet...

Payload content: Detected one tap...

ST87 wake up completed...

Sending the data...

#IPRECV: Detected one tap...

ST87 modem is in sleep mode...

Waiting for an event from sensor...

Going to sleep mode...
```

Figure 4.21: Sleep/wake-up trace with accelerometer sensor tap detection and UDP data transfer

4.2.4 Including memory (X-NUCLEO-PGEEZ1) in the project

Since the solution was supposed to primarily run on a battery and/or a solar panel, this meant that the energy available to perform certain operations was unpredictable. Do we have enough energy in the battery to send data frequently? What if sometimes, after waking up, we can only gather data from the sensors? These questions meant that the data gathered could, in some situations, be lost. If we imagine a scenario where, after sleep, we wake up but have enough energy only to read from the sensors and nothing else, we could choose to skip this operation completely. But if we were eager to perform it, how do we prevent the loss of data since we cannot send it?

In these sorts of situations, the thought-out solution was to save the data in some memory location and retrieve it later once the conditions were more ideal, in this case, when there was enough energy to send the data over the network. The type of memory used in this case was EEPROM, and the development board tried out was the X-NUCLEO-PGEEZ1 development board equipped with the M95P32 low-power EEPROM memory, as shown in figure 4.22.

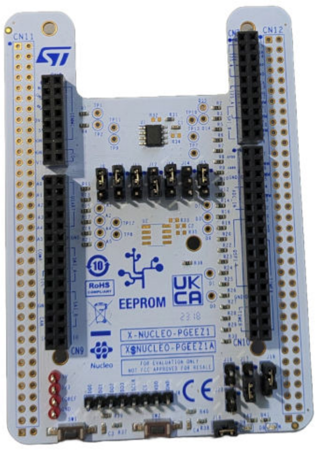
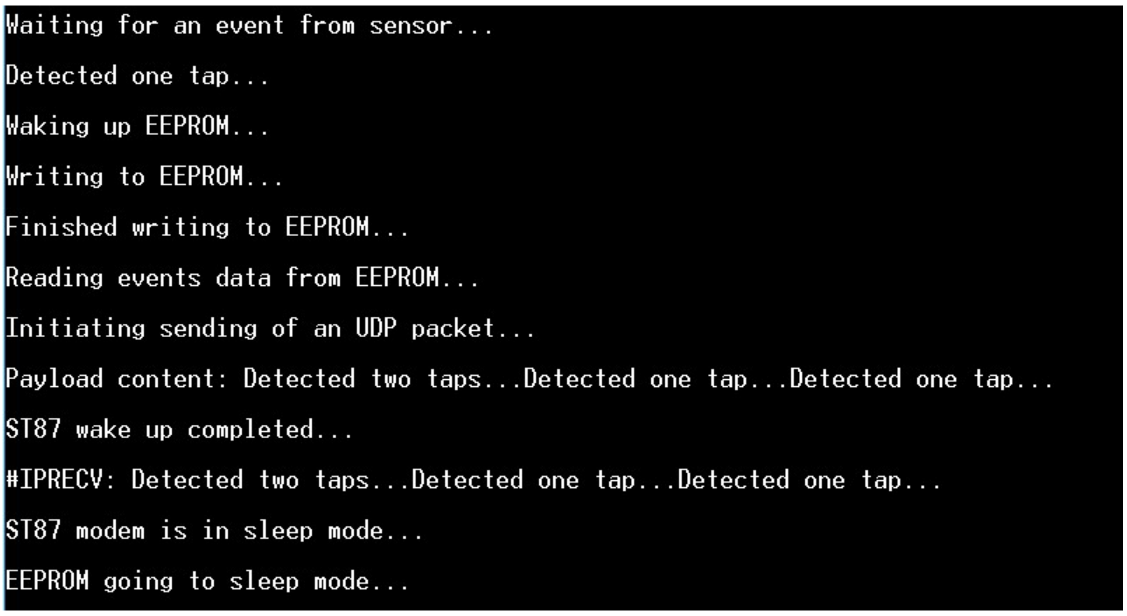


Figure 4.22: X-NUCLEO-PGEEZ1 development board

The firmware was modified accordingly. The flow of the scenario was changed so that between data gathering from the sensors and data sending over the NB-IoT network, the EEPROM was now included. Each time some data was gathered, it would be written to the EEPROM first. The sending of that same data was delayed until later when a certain number of data records had been written to the EEPROM. After that number was reached, the data was read from the EEPROM memory and sent in bulk. It should be noted that there was no logic for determining whether or not there was enough energy for a certain task, as all of the examples were run from the USB power supply. However, this setup was created to make it easier to implement the actual scenario discussed at the beginning of this subsection later.

The previous example that included tap detection was extended. A new task was added that would save the ASCII description of the tap event to the EEPROM memory. The accelerometer task now sent the ASCII data to the EEPROM task, not to the task responsible for starting the UDP transfer. After three events were received, the EEPROM task would read the previous events from the EEPROM component and send the ASCII content of the three events to the task that initiates the UDP transfer at once. The rest of the logic remained the same. The figure 4.23 provides a trace displaying how the flow was altered.



```
Waiting for an event from sensor...
Detected one tap...
Waking up EEPROM...
Writing to EEPROM...
Finished writing to EEPROM...
Reading events data from EEPROM...
Initiating sending of an UDP packet...
Payload content: Detected two taps...Detected one tap...Detected one tap...
ST87 wake up completed...
#IPRECV: Detected two taps...Detected one tap...Detected one tap...
ST87 modem is in sleep mode...
EEPROM going to sleep mode...
```

Figure 4.23: Sleep/wake-up trace with accelerometer sensor tap detection, EEPROM and UDP data transfer

4.3 STEVAL-NBIOT

After extensive prototyping with a solution that implemented a full expected scenario of sensing and sending data over the network, the work continued, starting with the hardware development for the new platform named STEVAL-NBIOT and later included firmware development as well.

4.3.1 Hardware

The hardware development began with work on schematics. Since access to similar projects was available, it turned out that some of the schematic parts could be reused from those previous projects. Reusing the schematics led to faster development time and lowered the potential for introducing bugs. Another useful fact that eased the development was the extensive documentation provided by STMicroelectronics for their products. A lot of information was neatly documented, mainly in the datasheet for some device or part that was used. The datasheet mainly showcased the necessary electronic components that had to be included for the specific part to function properly. Since all of the used components (microcontroller, sensors, etc.) were made by STMicroelectronics, this proved to be really valuable. The PCB design process based on the schematics was outsourced to an external contractor and is not discussed here; only the commentary on the layout of the final PCB is given.

Schematics

The work on schematics was broken down into five different parts:

- Power management, which included battery management
- STM32U575 interconnection with other components, EEPROM, user buttons, and LEDs
- ST87M01 interconnection with other components and antennas
- Sensors
- USB-C and external 34-pin connector

Figure 4.24 gives a block diagram representation of the power management for the STEVAL-NBIOT board. Based on the design, it is possible to choose between power supply from the USB or from the battery. In order to manage the charging of the battery when connected to the USB and switch between the charging and discharging modes, the STBC02 battery controller circuit was chosen to take care of this part. Next, there are two linear-dropout regulators, one providing current up to 150 mA and another one providing current up to 1 A.

It can also be observed from Figure 4.24 that the regulator providing less current is responsible for powering the STM32U575, the EEPROM, and the sensors, as these components do not have high power consumption. As the sensors and the EEPROM have very low-power requirements, they were connected to the standard GPIO of the STM32U575 MCU. This enabled easily controlling the power state of these components from the firmware, allowing them to be powered on or off as needed. The other linear-dropout regulator, which goes up to 1 A, powers the ST87M01. This was intentionally separated to achieve better power consumption and meet the power requirements, as it was observed that the ST87M01 can require current ranging from 500 mA to 1 A when transmitting data over the NB-IoT network. Based on the situation, when needed, the ST87M01 can activate the regulator that goes up to 1 A and otherwise use the lower output current one.

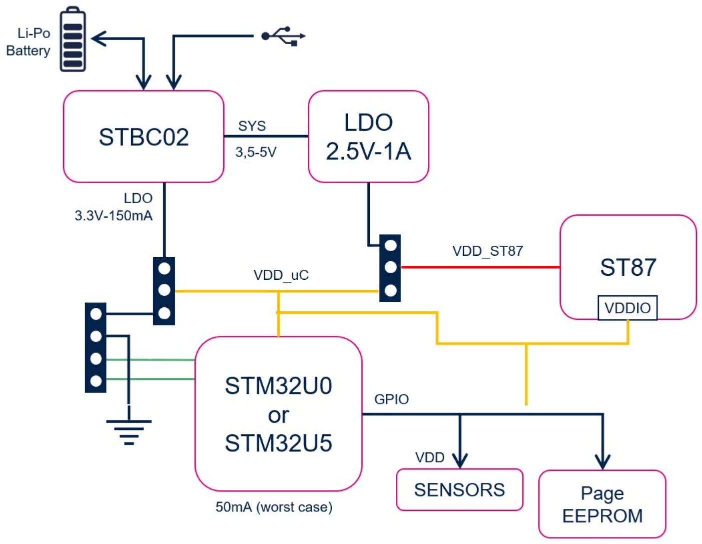


Figure 4.24: STEVAL-NBIOT power management block diagram [39]

The STM32U575 required interconnecting with all of the other relevant components. These components ranged from the ST87M01, interconnected through UART, to the battery controller, in order to track the battery status, to the sensors over standard communication protocols, here I2C and SPI (more on this later in this section). EEPROM interconnection over SPI is also included in this section, but it should be noted that the SPI peripheral used is different from the one for sensors, and finally all relevant GPIOs. Reset and user buttons, for some level of interaction, as well as two LEDs for providing visual information to the users, are provided and their exact functions can be defined by the firmware. For better accuracy, there are two external oscillators included on the board: one low-speed oscillator of 32.768 kHz and one high-speed one of 16 MHz.

The main focus of the ST87M01 schematics work was on antenna interconnection and design, as well as the SIM card slot interconnection. The work was heavily influenced by the suggestions and advice from the ST87M01 and RF antenna design department, which also provided the PCB antenna design. Figure 4.25 below demonstrates the PCB antenna design. Nevertheless, it is useful when talking about the schematics as well. The main point that should be noted here is that there are two PCB antennas, one for NB-IoT and another for GNSS. For both antennas, surface mount connectors are available but not mounted by default. The user has the option of using the PCB antennas or mounting an external antenna depending on the usage choice. Another thing, more relevant to the PCB itself, is that the antenna design had an impact on the dimensions of the board as it required a large amount of clearance (space on the PCB with no other electrical components).

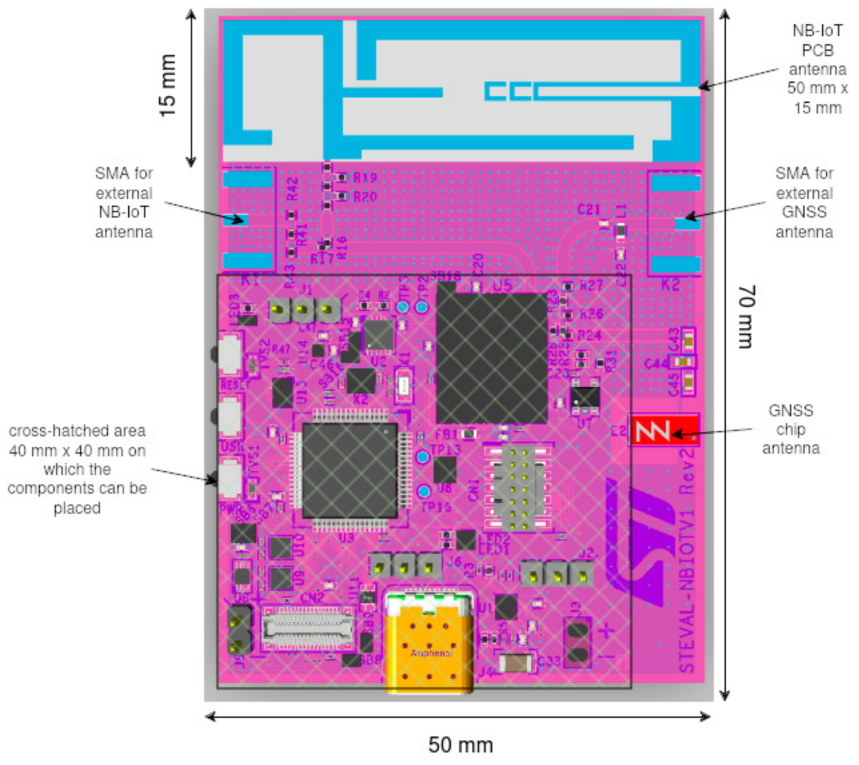

15 mm
NB-IoT PCB antenna 50 mm x 15 mm
SMA for external NB-IoT antenna
SMA for external GNSS antenna
70 mm
GNSS chip antenna
cross-hatched area 40 mm x 40 mm on which the components can be placed
50 mm

Figure 4.25: STEVAL-NBIOT antenna desing [39]

As far as the sensors go, the SHT40 temperature and humidity sensor is connected to the STM32U575 via I2C. The other two, the LPS22DF barometer and the LIS2DUXS12 accelerometer, are connected via SPI and have dedicated chip select GPIO pin connections as well. Another interesting feature of these two sensors is the interrupt connections, which are routed to the GPIO pins of the STM32U575 that were configured as external interrupt GPIOs. This enables these two sensors to signal to the MCU that some event has happened. Usually, this event signifies that there is some data ready; in the case of the accelerometer, for example, it can also signify a detection of a tap or an orientation change event. Either way, the important thing to notice is that these events can be used to wake up the MCU.

At the very end, the external connectors were defined. Since the board features a small form factor that does not permit the usage of Arduino connectors or Morpho connectors, a special small-dimension 34-pin external connector was used. Some of the peripherals were exposed through this external connector. The idea was to enable connecting other sensors as well, since the board only includes three sensors. The other prominent feature is the USB-C connector that enables connection to the PC and serves as a power supply option as well.

Figure 4.26 showcases the overall block diagram of the STEVAL-NBIOT development platform consisting of the previously discussed parts.

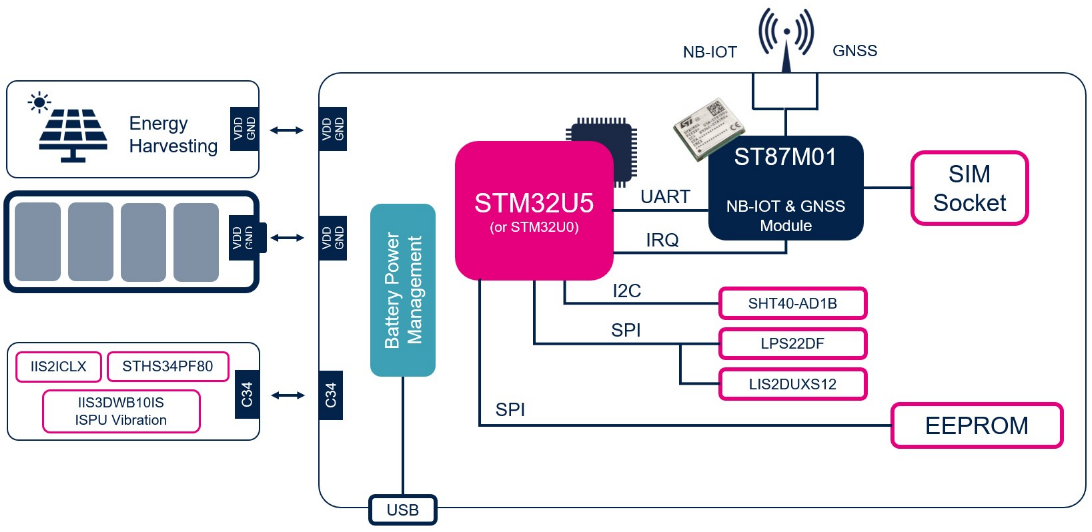


Figure 4.26: STEVAL-NBIOT block diagram [39]

It turned out that the approach of reusing parts of the schematic mentioned in the beginning can significantly reduce hardware-related bugs, but it can also lead to some bugs that are hard to spot in the development phase. Luckily, only one case of such a hardware bug was encountered. The bug ended up being a mistake in the bill of materials, where the wrong capacitance was used for the external low-power oscillator. It was confirmed that this issue was introduced by reusing the component from the previous project without validating the value of the capacitance. However, even though this minor bug was encountered, it proved that this approach was good because no other significant problems were encountered, which led to fast progression of development.

Parallel to the development of the new platform, work on the expansion board that would enable more power supply options, specifically options that enable the use of energy harvesting technologies, took place. The development of this expansion board was primarily done under guidance from ST's department in Catania. At the time of writing, it is still a work in progress and has not been tested with the STEVAL-NBIOT platform. Nevertheless, a brief explanation is given. As can be seen in Figure 4.27, the developed expansion board includes the possibility of using a solar panel and harvesting the solar energy, accumulating it into the supercap. The purpose of the supercap is to serve as an energy buffer.

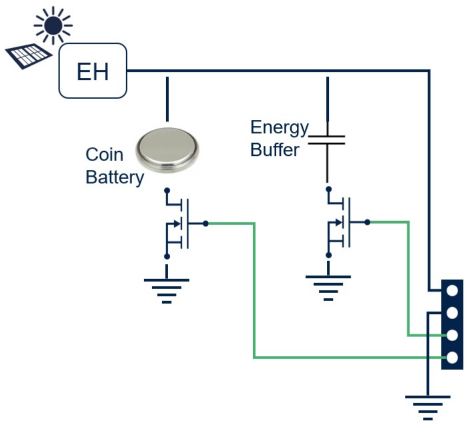


Figure 4.27: Expansion board for STEVAL-NBIOT that enables energy harvesting support [39]

PCB

The PCB design was not the focus of this thesis and was outsourced. Nevertheless, a short overview of the PCB for the STEVAL-NBIOT is given below. Figure 4.28 provides an overview of the top side of the PCB and enumerates the present components. What can be seen is that a large part of the PCB is clear of any components, and the majority of the components are concentrated in one area. This was directly caused by the PCB antenna design and requirements. The final PCB dimensions were 50 x 70 mm, with the useful area populated by the components being 40 x 40 mm.

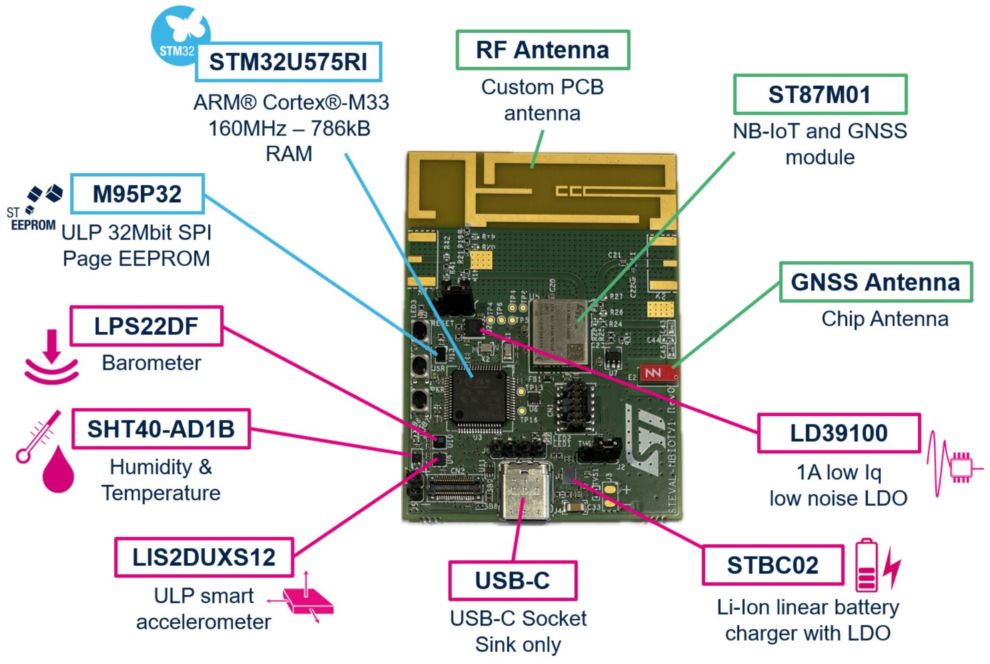


Figure 4.28: STEVAL-NBIOT PCB top view [39]

In contrast, Figure 4.29 provides an overview of the bottom side of the PCB, with the major component being the SIM card slot. Again, a significant area is devoid of any components because of the PCB antennas.

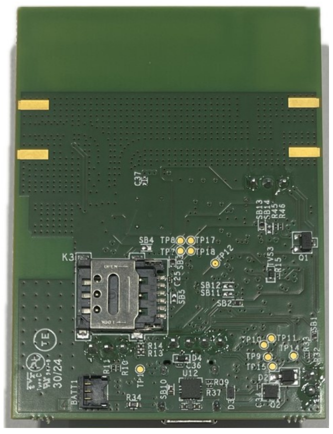


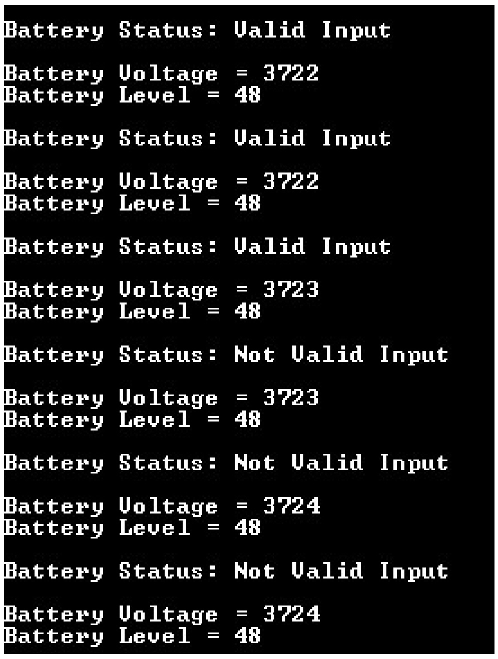
Figure 4.29: STEVAL-NBIOT PCB bottom view [39]

4.3.2 Firmware

The impact of a successful prototype was evident in the official firmware development for a new platform. Namely, the developed firmware example for a scenario that involved reading data from the sensor and sending it over the network was made to work on the new board with only a few modifications. The necessary changes involved different pins and different instances of the peripherals. For example, using the I2C1 peripheral instead of I2C2, or even using the same peripheral but mapped to different pins. The rest of the logic that involved peripheral device drivers and application logic, as well as Easy Connect library configuration, remained the same. No issues were encountered during this process, and the development continued with working on a BSP for the device.

The sensor drivers were already available because ST provides drivers for all of the sensors it manufactures. The general software structure for the BSP was based on the existing one from a previous project. In the end, it involved support for all of the required peripherals (LPUART, I2C, SPI, ADC, etc.) and components (battery charger, EEPROM) as well as clock and pin configurations. The idea was to have an API that would enable quick and easy development of application logic. This phase was also a sort of testing phase. Covering all of the firmware for the peripherals and components also included testing if these components and peripherals worked properly. In the process of porting and trying to run the example that included sleep/wake-up, EEPROM, and tap events from the accelerometer from the prototyping stage, several parts of the system were tested. Later, as we will see with other firmware examples, the other two sensors were tested out. In the process of trying out the communication with the ST87 module, EEPROM, and the sensors, the peripherals used in communicating with them were verified to function correctly. One of the small issues with using the sensors was forgetting to actually set the GPIO pin used as a power supply for these components to high, which of course had an easy fix of configuring the pin properly.

Talking about the pin configuration, several minor issues resulted from configuring the wrong pins or providing the wrong configuration for certain pins. There were instances of not enabling the pull-up resistor in firmware for certain pins, or not enabling the clock for certain peripherals. This led to some of the peripherals not functioning properly during this bring-up phase. For example, a wrong pin configuration for the battery charging controller led to inaccurate reporting of the battery status. The battery charger was outputting a signal to this pin, and depending on the sampled frequency of the signal, information about the current battery status could be obtained. The information included whether the battery was being charged, if it was discharging, or if some error was detected. The wrong pin configuration led to inaccurate battery status being reported, as can be seen in Figure 4.30.



```
Battery Status: Valid Input

Battery Voltage = 3722
Battery Level = 48

Battery Status: Valid Input

Battery Voltage = 3722
Battery Level = 48

Battery Status: Valid Input

Battery Voltage = 3723
Battery Level = 48

Battery Status: Not Valid Input

Battery Voltage = 3723
Battery Level = 48

Battery Status: Not Valid Input

Battery Voltage = 3724
Battery Level = 48

Battery Status: Not Valid Input

Battery Voltage = 3724
Battery Level = 48
```

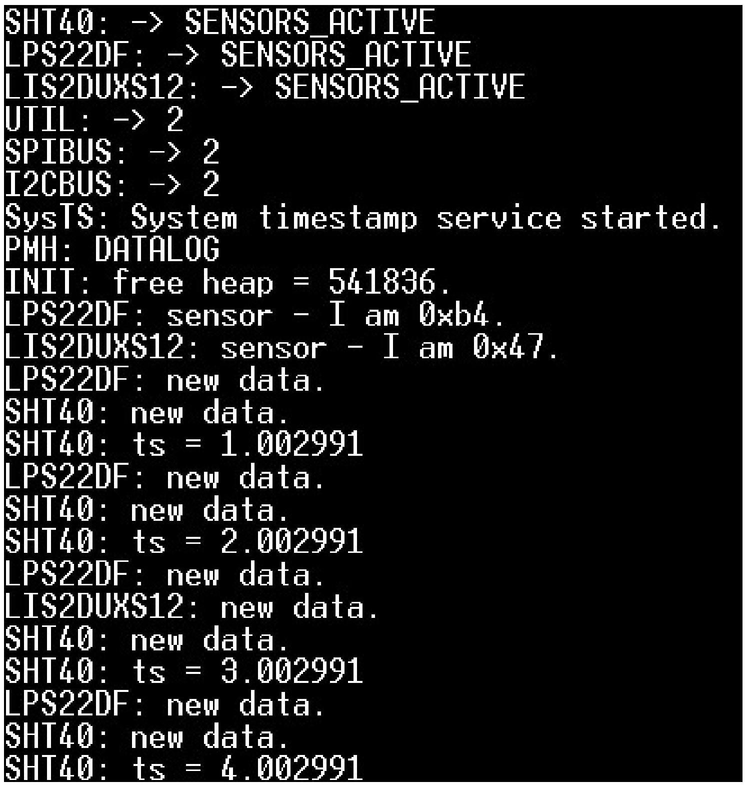
Figure 4.30: Trace showcasing inaccurate battery status because of the wrong pin configuration

After ensuring that everything worked properly, it was time to move to the firmware development to enable integrating the device with ST's data logging desktop platform.

Firmware that supports ST's data logging platform

Since this was a new product, it initially lacked the proper description of the device model in the DTDL language, which represents the backbone of ST's data logging platform. To remedy this, the first step was to devise a device model by describing the new device and all three sensors that it possessed. Based on this device model, the firmware code was generated with support for Azure PnP-like messages over USB. The missing parts of the generated code that needed to be filled with information about the sensors (such as sensors' output data rate) and platform information (name of the platform, board, and firmware ID identifier) were filled after this step.

Code reuse was also taken advantage of again. The standard firmware structure used for this purpose, based on the eLOOM platform, was adapted from a previous project. It already had support for the LPS22DF and SHT40 sensors. To be more precise, the RTOS tasks that communicate with sensors over SPI and I2C, gather data, and send it over USB were already available. The LIS2DUXS12 task had to be implemented but not from scratch, as there was already an available task for a LIS2DU12 sensor that served as a base. The main obstacles faced here were getting to know the firmware structure used for data logging applications. This required understanding the eLOOM software platform, how it was organized, and the parts that make it up. Another hurdle was the generated code from the DTDL device model. After generating the code, certain parts are left to be implemented by the user and filled with the proper information. The missing parts included the basic information about the board and the mapping of the sensor parameters from the Azure PnP-like message format to the format used by the firmware.

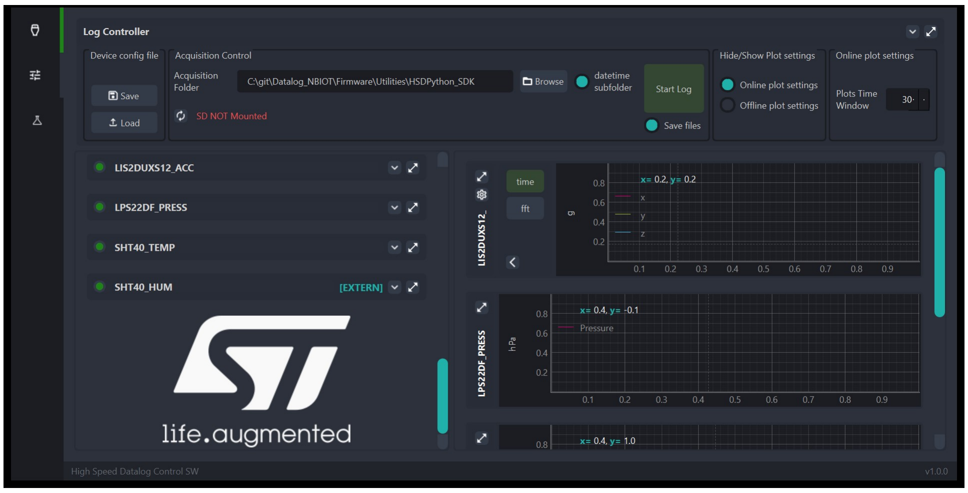


```
SHT40: -> SENSORS_ACTIVE
LPS22DF: -> SENSORS_ACTIVE
LIS2DUXS12: -> SENSORS_ACTIVE
UTIL: -> 2
SPIBUS: -> 2
I2CBUS: -> 2
SysTS: System timestamp service started.
PMH: DATALOG
INIT: free heap = 541836.
LPS22DF: sensor - I am 0xb4.
LIS2DUXS12: sensor - I am 0x47.
LPS22DF: new data.
SHT40: new data.
SHT40: ts = 1.002991
LPS22DF: new data.
SHT40: new data.
SHT40: ts = 2.002991
LPS22DF: new data.
LIS2DUXS12: new data.
SHT40: new data.
SHT40: ts = 3.002991
LPS22DF: new data.
SHT40: new data.
SHT40: ts = 4.002991
```

Figure 4.31: Datalog firmware trace showing the sensors working correctly

The resulting trace over the USB can be seen in Figure 4.31, and it shows all three of the sensors being correctly initialized and the generated events by those same sensors. Here, the generated events represent the availability of new data samples.

The final step was to integrate the firmware with the desktop GUI application. The previously created device model was linked with the desktop application and included the expected board and firmware ID. This was done to enable the desktop application to load the proper GUI elements. By recognizing the board as being the STEVAL-NBIOT based on the device model stored on disk, it loads the GUI elements for the three sensors available on the board. Figure 4.32 gives an overview of the desktop application GUI after connecting the STEVAL-NBIOT and before starting the acquisition. It can also be observed that only the corresponding GUI elements based on the sensors present on the board are loaded.


Acquisition Folder: C:\git\Datalog_NBIOT\Firmware\Utilities\HSDPython_SDK

Plots Time Window: 30

Figure 4.32: STEVAL-NBIOT data logging desktop GUI overview for the STEVAL-NBIOT

Firmware that supports ST's asset tracking dashboard

Figure 4.33 represents the overall goal to be achieved, in which the new STEVAL-NBIOT platform is able to support both ST's data logging and dashboard infrastructure. The firmware support for the data logging functionality was laid out and presented in the previous section. The focus of this section will be the development that enabled the device to connect and send sensor data to the dashboard.

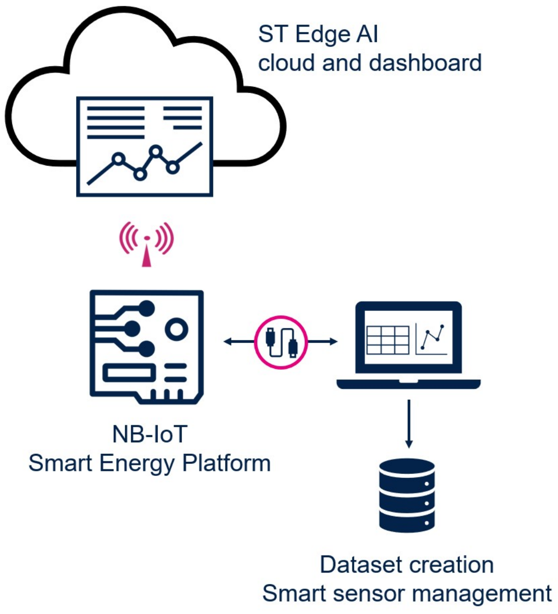


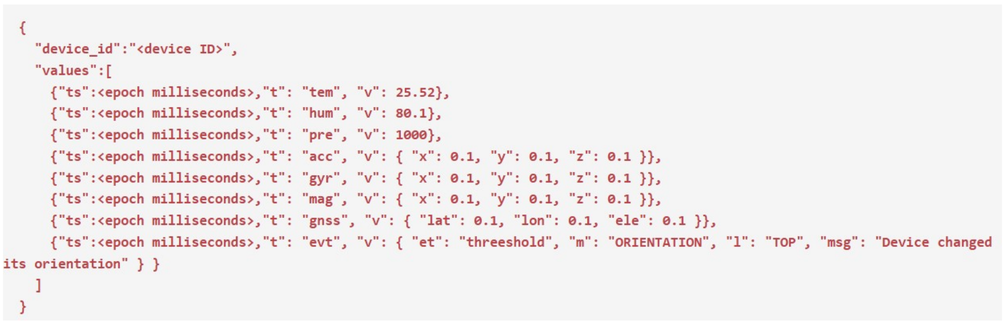
Figure 4.33: STEVAL-NBIOT data logging and dashboard support

The first required step was to prepare and provision the necessary certificates onto the ST87M01 module. The dashboard itself is hosted on Amazon's AWS platform, so it required getting the root certificate from Amazon and storing it in the flash memory of the ST87M01. This way, because HTTPS was used, the device would only accept connections with Amazon servers. The dashboard server was not configured to ask for the device certificate needed for the server to correctly confirm the authenticity of the device. Nevertheless, the firmware of the ST87M01 modem required both the server and the device certificate to be present in order for the HTTPS protocol stack to function properly. This was resolved by loading a self-signed device certificate.

Regarding the firmware work for the application logic running on the STM32U575 of the STEVAL-NBIOT board, the Easy Connect library provided the required abstraction and sequence for HTTPS, so the main bulk of the work was on getting the data from all of the sensors and creating a correct HTTP header and body.

The architecture of the firmware consisted of five FreeRTOS tasks. Each of the three sensors had a task associated with it that was responsible for initialization and data gathering. One task was used to coordinate and schedule the collection of data from all of the sensors in one place. This task would resume each of the three blocked sensor tasks in sequence, one after another, and get the data from each. Once the data was collected, this task would send the data gathered from all of the sensors to the task responsible for running the main scheduler of the Easy Connect library and creating the HTTPS POST request.

The creation of the HTTPS header required manually setting the "Authorization" API key used to enable the interaction with the dashboard and sending data to it. In any other case, the server would not accept the requests. The next step was in creating the properly formatted JSON body of the request. The dashboard had support for several types of data coming from the sensors, and it required that this data be sent in a specific format. This format was defined as a JSON object with several key-value pairs. A more detailed look is given in Figure 4.34 below.



```
 {
   "device_id":"<device ID>",
   "values":[
     {"ts":<epoch milliseconds>,"t": "tem", "v": 25.52},
     {"ts":<epoch milliseconds>,"t": "hum", "v": 80.1},
     {"ts":<epoch milliseconds>,"t": "pre", "v": 1000},
     {"ts":<epoch milliseconds>,"t": "acc", "v": { "x": 0.1, "y": 0.1, "z": 0.1 }},
     {"ts":<epoch milliseconds>,"t": "gyr", "v": { "x": 0.1, "y": 0.1, "z": 0.1 }},
     {"ts":<epoch milliseconds>,"t": "mag", "v": { "x": 0.1, "y": 0.1, "z": 0.1 }},
     {"ts":<epoch milliseconds>,"t": "gnss", "v": { "lat": 0.1, "lon": 0.1, "ele": 0.1 }},
     {"ts":<epoch milliseconds>,"t": "evt", "v": { "et": "threeshold", "m": "ORIENTATION", "l": "TOP", "msg": "Device changed
its orientation" } }
    ]
 }
```

Figure 4.34: ST's dashboard JSON request body expected format

The first pair consisted of a key "ts" with a value that represented a UNIX timestamp. This timestamp generally represents the number of seconds passed since January 1, 1970, but in the case of the dashboard, it is used in units of milliseconds. The second piece of data in the JSON object was the key "t" with a value that was a string depicting the type of data; in the case of temperature, it is represented as "temp". Finally, the actual value of the reading was given as a third key-value entry of the JSON object. The key "v" had a value representing the measurement from the sensor. The value here is just a floating point number, but for more complex sensors, like the accelerometer, it represented a JSON object with key-value pairs depicting the axes of the accelerometer and their values. The body also required setting the correct device ID, which was a unique name selected when registering the device on the dashboard. The registration step required only providing the user-friendly name and some unique identifier.

The RTC peripheral was used to set up the correct date and time when the board was powered up. The correct date and time were obtained from the ST87M01 module. The date and time string was parsed, and the RTC was initialized to the correct value.

The data gathering was organized in a periodic manner; the initially tested period duration was 10 seconds. In between, the STM32U575 would be put to sleep in STOP2 mode in the same manner as the prototype example. In order to be able to track time even when in STOP2 mode, the LPTIM was used in LPBAM mode. The timer was configured with the proper period duration, and before the device was to enter STOP2 mode, the LPBAM mode was enabled and the LPTIM was started to initiate the counting. Once the LPTIM counted to the specified period, it would issue an interrupt request that would wake up the system and resume the task responsible for collecting and scheduling data gathering from the sensors.

Chapter 5

Results

5.1 What is included in this chapter

The most important factors of the new hardware platform were tested, and the results are detailed in the following sections. The first two factors included the GNSS and NB-IoT performance. The antenna performance evaluation was done for both NB-IoT and GNSS and is detailed in one section. There is a dedicated section on the performance of the GNSS protocol, going into more details about the performance such as the precision of the evaluated position, etc. The third important factor that was tested was the power consumption, determining which components have the biggest. The tested scenario includes gathering data from the sensors, then sending data over UDP using NB-IoT network, and getting the echo from the server. Last but not least, regarding the work on firmware, an overview of the data logging and dashboard was given while the data gathering by the board was active. This was performed to showcase the features offered by the developed firmware examples.

5.2 NB-IoT and GNSS antenna evaluation

The tests were carried out in a special environment called an anechoic chamber. It represents a shielded space designed and constructed to suppress electromagnetic waves coming from the outside in order to reduce outside noise and interference. Inside this controlled environment, it is possible to measure the performance and test RF devices. The setup that was used can be seen in Figure 5.1.

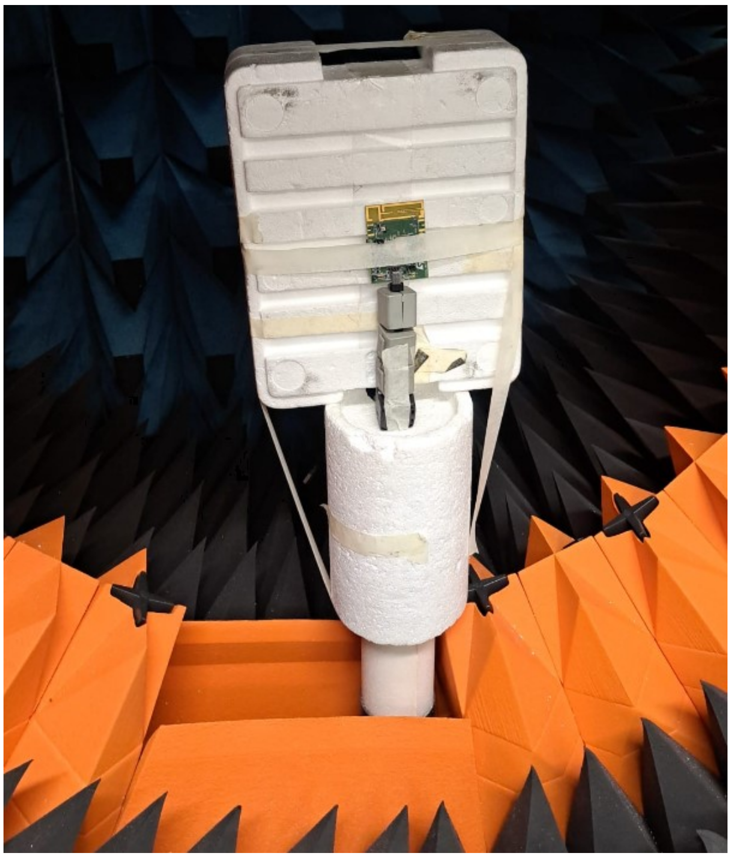


Figure 5.1: Test environment for evaluating the NB-IoT antenna performance of the STEVAL-NBIOT

The NB-IoT frequency bands dedicated for Europe are B3 (1800 MHz band), B8 (900 MHz band), and B20 (800 MHz band). This is the reason why the antenna performance was evaluated for these bands specifically. To evaluate the performance, an RF antenna efficiency measure was used. This measure tells us how much of the power that was fed into the antenna is transmitted (emitted) out. An ideal antenna has an efficiency of 100%, which means that all of the power fed into the antenna is emitted from it. In practice, this is impossible to achieve because some amount of power is lost due to resistive elements. In general, a good antenna emits between 50% to 60% of the power supplied to it. The measured efficiency of uplink frequency ranges for each band can be observed in Figure 5.2 below.

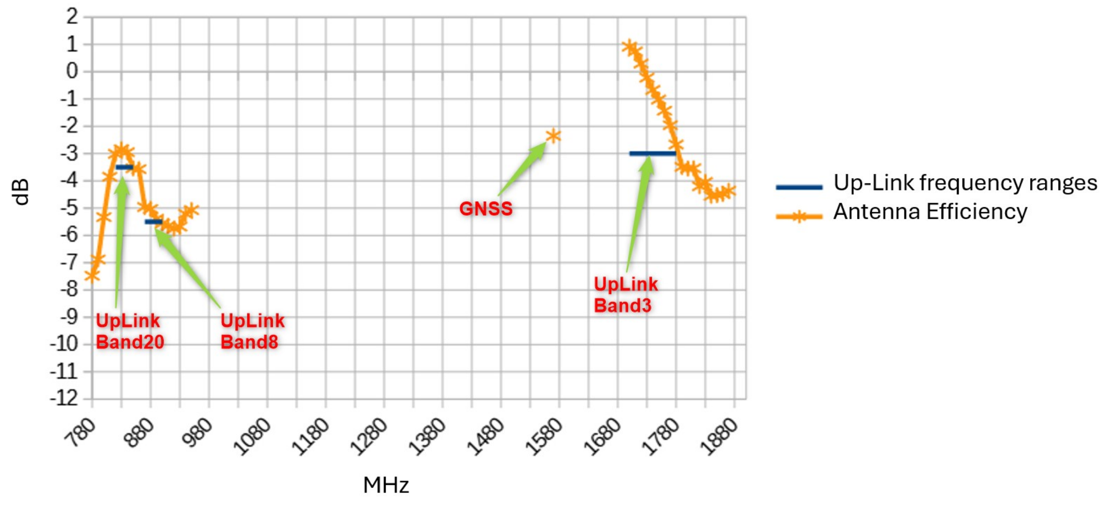


Figure 5.2: PCB NB-IoT antenna performance of the STEVAL-NBIOT for bands 3, 8, and 20

It was observed that for B8, the minimum efficiency was 30%. Next was B20, with a minimum efficiency of 45%. Finally, the best minimum efficiency was on B3, with 65%. Based on the graph for the B8 frequency range, the efficiency starts high and linearly drops to the minimum reported efficiency as the frequency increases. The same can be observed for B3. B20 differs because the overall efficiency reported inside the frequency range of B20 is almost constant, with drops near the edges of the band. Therefore, B20 is better than the other two if we consider the efficiency along the whole range of frequencies inside the band.

The efficiency of the antenna for GNSS use cases was determined to be 60%, making it more than ideal to be used for both NB-IoT and GNSS.

5.3 GNSS protocol evaluation

For the evaluation of the performance of the GNSS protocol, tests were conducted for both the evaluation kit board EVLST87M01 for the ST87M01 modem and the STEVAL-NBIOT board. Figure 5.3 showcases the setup used for the EVLST87M01 board with the external antenna. The setup for the STEVAL-NBIOT used just the board as-is since every necessary component is present on the board, including the PCB antenna. One of the goals was to compare the new platform against the already developed evaluation board for the ST87M01 module. The test results featured here were taken from [40].

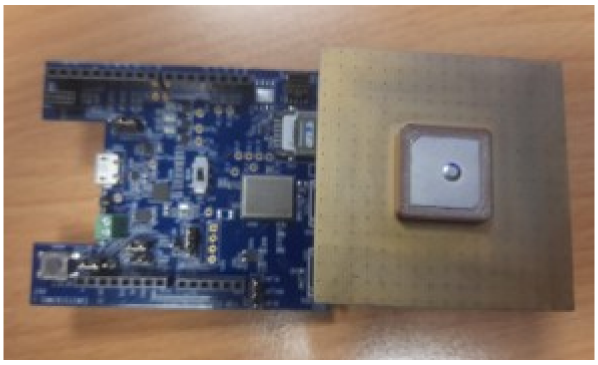


Figure 5.3: EVLST87M01 with an external antenna used for GNSS protocol performance testing [40]

The following parameters were taken into consideration for performance evaluation [40]:

- Number of tracked satellites
- Minimum, maximum, and average value for the C/N_0 (the ratio of carrier power to the noise power mixed with the signal)
- Accuracy of the determined position

The tests were done under the following conditions [40]:

- Outside field tests in a suburban area during a sunny day
- Different antenna slopes, with a test duration for each slope around 15 minutes

The overview of the testing area is given below in Figure 5.4.

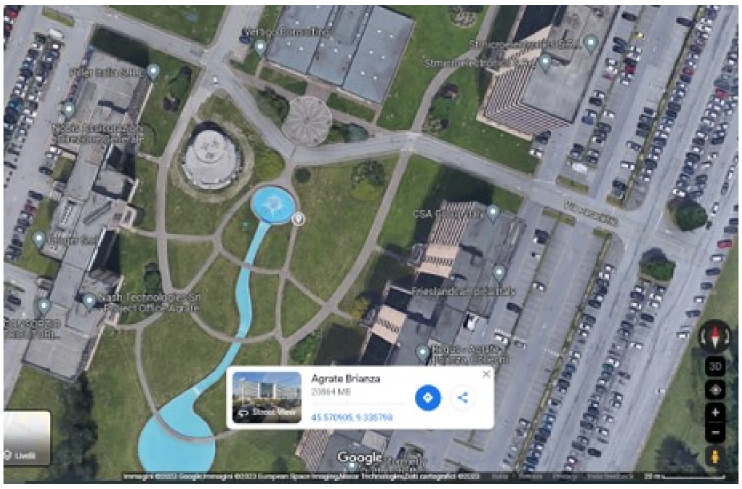


Figure 5.4: GNSS protocol testing area [40]

Comparative tests were repeated for different slopes around the X and Y axes at 0, ±45°, and ±90°, as shown in Figures 5.5 and 5.6, respectively. Note that the X, Y, and Z axes correspond to the standard coordinate system used by the GNSS. Here, the different orientations of the EVLST87M01 and the external antenna attached to it can be seen. The same orientations were used when testing the STEVAL-NBIOT.

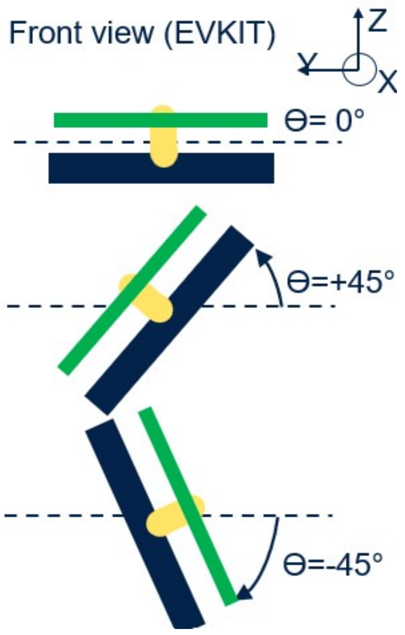


Figure 5.5: Board orientations around X-axis used for GNSS protocol testing [40]

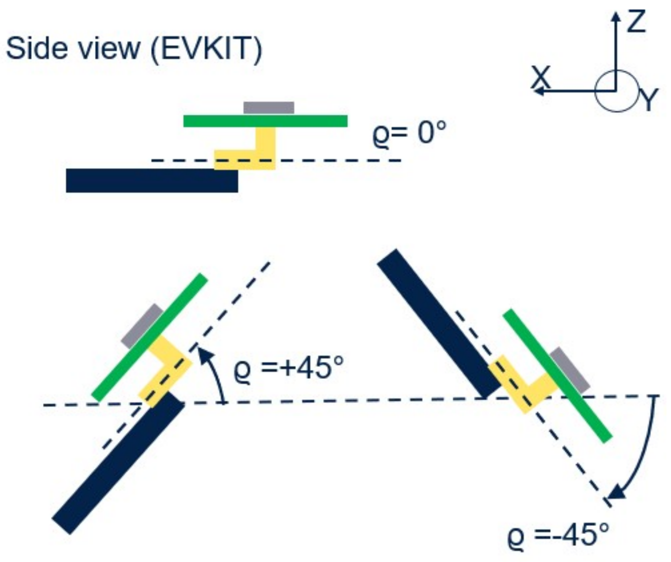
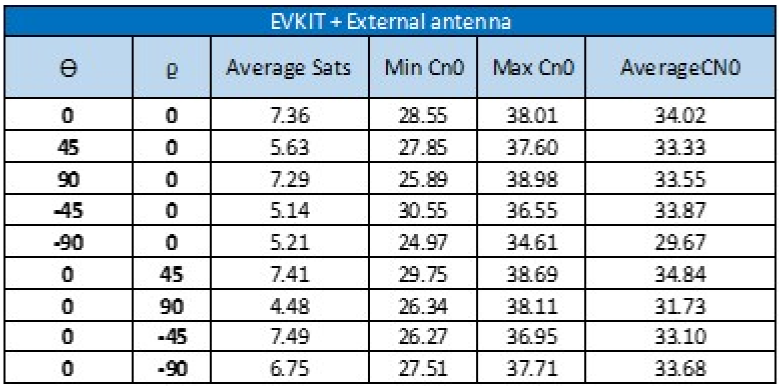


Figure 5.6: Board orientations around Y-axis used for GNSS protocol testing [40]

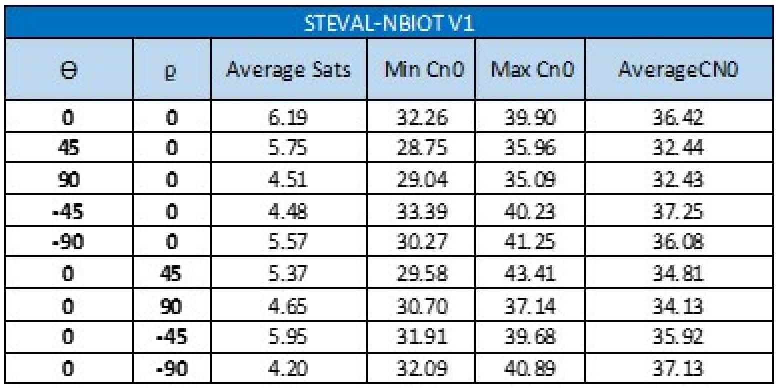
Regarding the average number of satellites and C/N_0 for different slopes the figure 5.7 gives an overview for the EVLST87M01. Figure 5.8 gives more details about the same measurements but for the STEVAL-NBIOT board. While 5.9 gives a comparison between the two.

Regarding the average number of satellites and C/N_0 for different slopes, Figure 5.7 provides an overview for the EVLST87M01. Figure 5.8 gives more details about the same measurements for the STEVAL-NBIOT board. Figure 5.9 offers a comparison between the two.



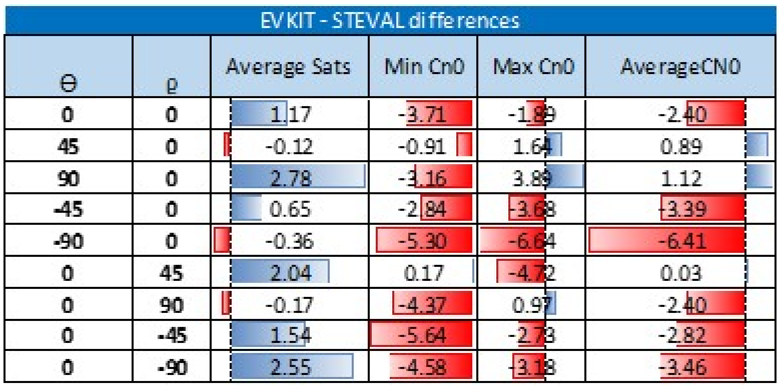
EVKIT + External antenna					
θ	ϱ	Average Sats	Min Cn0	Max Cn0	AverageCN0
0	0	7.36	28.55	38.01	34.02
45	0	5.63	27.85	37.60	33.33
90	0	7.29	25.89	38.98	33.55
-45	0	5.14	30.55	36.55	33.87
-90	0	5.21	24.97	34.61	29.67
0	45	7.41	29.75	38.69	34.84
0	90	4.48	26.34	38.11	31.73
0	-45	7.49	26.27	36.95	33.10
0	-90	6.75	27.51	37.71	33.68

Figure 5.7: EVLST87M01 average number of satellites and C/N_0 for different board orientations [40]



STEVAL-NBIOT V1					
ϴ	ρ	Average Sats	Min Cn0	Max Cn0	AverageCN0
0	0	6.19	32.26	39.90	36.42
45	0	5.75	28.75	35.96	32.44
90	0	4.51	29.04	35.09	32.43
-45	0	4.48	33.39	40.23	37.25
-90	0	5.57	30.27	41.25	36.08
0	45	5.37	29.58	43.41	34.81
0	90	4.65	30.70	37.14	34.13
0	-45	5.95	31.91	39.68	35.92
0	-90	4.20	32.09	40.89	37.13

Figure 5.8: STEVAL-NBIOT average number of satellites and C/N_0 for different board orientations [40]

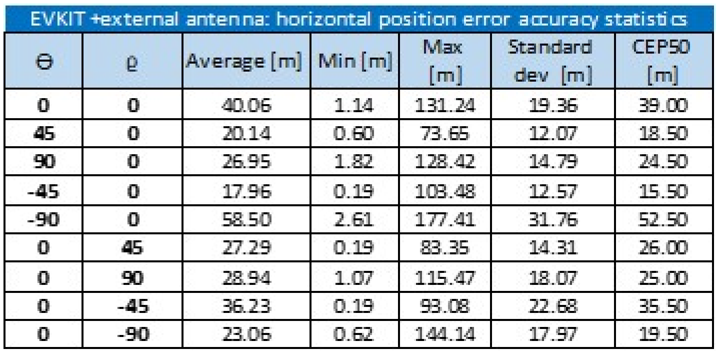


EVKIT - STEVAL differences					
ϴ	ρ	Average Sats	Min Cn0	Max Cn0	AverageCN0
0	0	1.17	-3.71	-1.89	-2.40
45	0	-0.12	-0.91	1.64	0.89
90	0	2.78	-3.16	3.89	1.12
-45	0	0.65	-2.84	-3.68	-3.39
-90	0	-0.36	-5.30	-6.64	-6.41
0	45	2.04	0.17	-4.72	0.03
0	90	-0.17	-4.37	0.97	-2.40
0	-45	1.54	-5.64	-2.73	-2.82
0	-90	2.55	-4.58	-3.18	-3.46

Figure 5.9: Comparison between the EVLST87M01 and STEVAL-NBIOT for the average number of satellites and C/N_0 test [40]

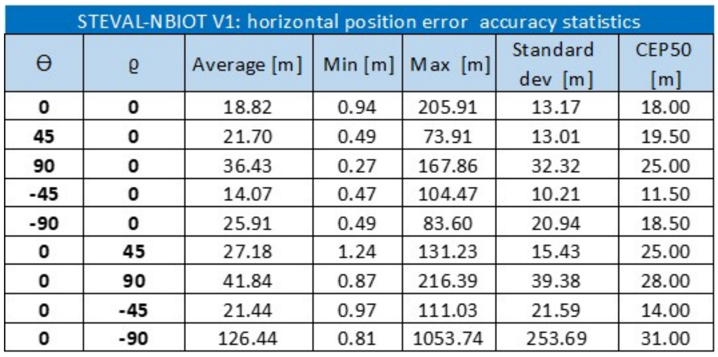
The results from the figures above can be considered good in general for both boards. The STEVAL-NBIOT board showed better performance in terms of C/N_0, while the EVLST87M01 showed a slightly higher number of tracked satellites. The STEVAL-NBIOT board also showed better results compared to the EVLST87M01, especially when the slope angles were higher.

The following figures give an estimated horizontal position (latitude and longitude) error compared to the true position of the device for both the EVLST87M01 in Figure 5.10 and the STEVAL-NBIOT in Figure 5.11. It shows the distance in meters between the estimated and true positions. The comparison between the two is given in Figure 5.12.



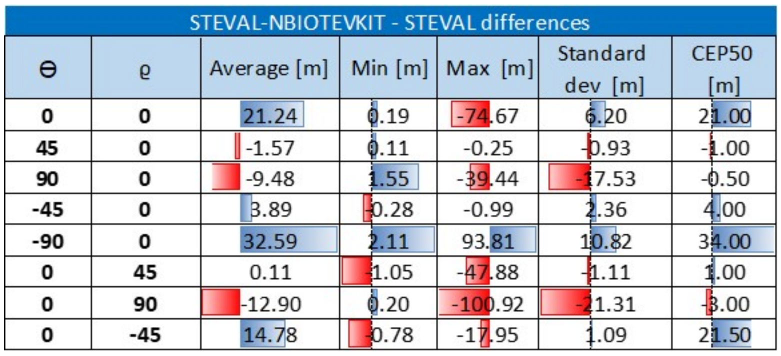
EVKIT +external antenna: horizontal position error accuracy statistics						
ϴ	ϱ	Average [m]	Min [m]	Max [m]	Standard dev [m]	CEP50 [m]
0	0	40.06	1.14	131.24	19.36	39.00
45	0	20.14	0.60	73.65	12.07	18.50
90	0	26.95	1.82	128.42	14.79	24.50
-45	0	17.96	0.19	103.48	12.57	15.50
-90	0	58.50	2.61	177.41	31.76	52.50
0	45	27.29	0.19	83.35	14.31	26.00
0	90	28.94	1.07	115.47	18.07	25.00
0	-45	36.23	0.19	93.08	22.68	35.50
0	-90	23.06	0.62	144.14	17.97	19.50

Figure 5.10: EVLST87M01 horizontal (latitude and longitude) position error for different board orientations [40]



STEVAL-NBIOT V1: horizontal position error accuracy statistics						
ϴ	ϱ	Average [m]	Min [m]	Max [m]	Standard dev [m]	CEP50 [m]
0	0	18.82	0.94	205.91	13.17	18.00
45	0	21.70	0.49	73.91	13.01	19.50
90	0	36.43	0.27	167.86	32.32	25.00
-45	0	14.07	0.47	104.47	10.21	11.50
-90	0	25.91	0.49	83.60	20.94	18.50
0	45	27.18	1.24	131.23	15.43	25.00
0	90	41.84	0.87	216.39	39.38	28.00
0	-45	21.44	0.97	111.03	21.59	14.00
0	-90	126.44	0.81	1053.74	253.69	31.00

Figure 5.11: STEVAL-NBIOT horizontal (latitude and longitude) position error for different board orientations [40]



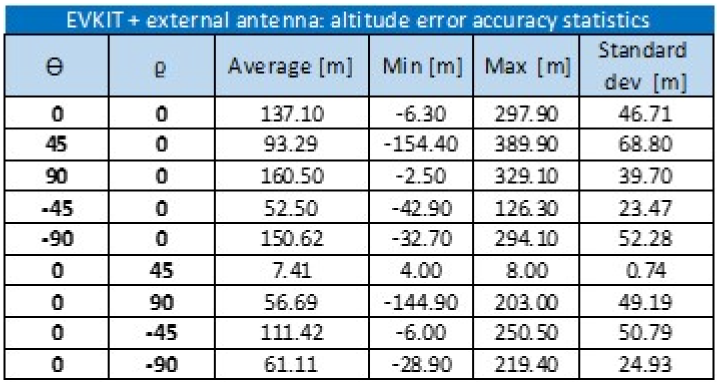
STEVAL-NBIOTEVKIT - STEVAL differences						
ϴ	ϱ	Average [m]	Min [m]	Max [m]	Standard dev [m]	CEP50 [m]
0	0	21.24	0.19	-74.67	6.20	21.00
45	0	-1.57	0.11	-0.25	-0.93	-1.00
90	0	-9.48	1.55	-39.44	-17.53	-0.50
-45	0	3.89	-0.28	-0.99	2.36	4.00
-90	0	32.59	2.11	93.81	10.82	34.00
0	45	0.11	-1.05	-47.88	-1.11	1.00
0	90	-12.90	0.20	-100.92	-21.31	-3.00
0	-45	14.78	-0.78	-17.95	1.09	21.50

Figure 5.12: Comparison between the EVLST87M01 and STEVAL-NBIOT for the horizontal (latitude and longitude) position error test [40]

The STEVAL-NBIOT board generally showed better horizontal position accuracy compared to the EVLST87M01 board with an external antenna.

In the same manner as the previously presented results for the horizontal position estimation error, the figures below show the comparison for the vertical position error estimation.

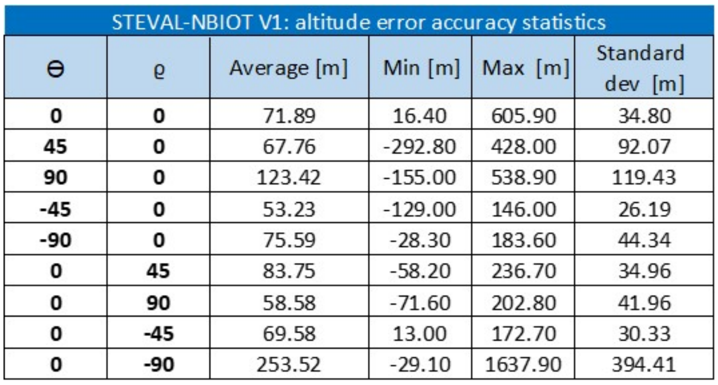
The error represents the offset in meters between the true vertical position of the device and the estimated one. For the EVLST87M01, the results are given in Figure 5.13, while for the STEVAL-NBIOT, the results are given in Figure 5.14. Finally, the comparison between the two is given in Figure 5.15.



EVKIT + external antenna: altitude error accuracy statistics

ϴ	ϱ	Average [m]	Min [m]	Max [m]	Standard dev [m]
0	0	137.10	-6.30	297.90	46.71
45	0	93.29	-154.40	389.90	68.80
90	0	160.50	-2.50	329.10	39.70
-45	0	52.50	-42.90	126.30	23.47
-90	0	150.62	-32.70	294.10	52.28
0	45	7.41	4.00	8.00	0.74
0	90	56.69	-144.90	203.00	49.19
0	-45	111.42	-6.00	250.50	50.79
0	-90	61.11	-28.90	219.40	24.93

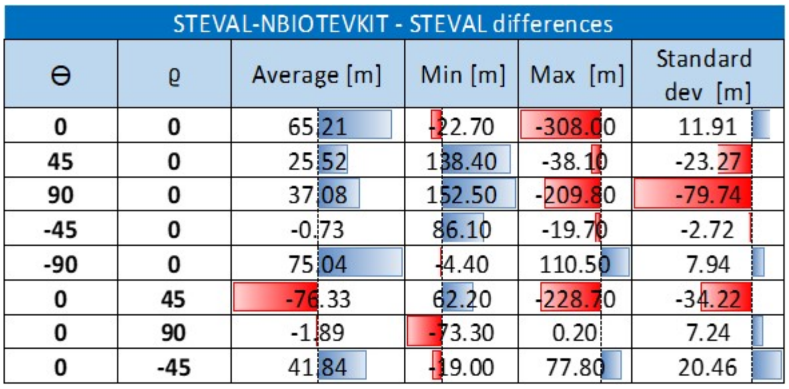
Figure 5.13: EVLST87M01 vertical (elevation) position error for different board orientations [40]



STEVAL-NBIOT V1: altitude error accuracy statistics

ϴ	ϱ	Average [m]	Min [m]	Max [m]	Standard dev [m]
0	0	71.89	16.40	605.90	34.80
45	0	67.76	-292.80	428.00	92.07
90	0	123.42	-155.00	538.90	119.43
-45	0	53.23	-129.00	146.00	26.19
-90	0	75.59	-28.30	183.60	44.34
0	45	83.75	-58.20	236.70	34.96
0	90	58.58	-71.60	202.80	41.96
0	-45	69.58	13.00	172.70	30.33
0	-90	253.52	-29.10	1637.90	394.41

Figure 5.14: STEVAL-NBIOT vertical (elevation) position error for different board orientations [40]



STEVAL-NBIOTEVKIT - STEVAL differences					
ϴ	ϱ	Average [m]	Min [m]	Max [m]	Standard dev [m]
0	**0**	65.21	-22.70	-308.00	11.91
45	**0**	25.52	138.40	-38.10	-23.27
90	**0**	37.08	152.50	-209.80	-79.74
-45	**0**	-0.73	86.10	-19.70	-2.72
-90	**0**	75.04	-4.40	110.50	7.94
0	**45**	-76.33	62.20	-228.70	-34.22
0	**90**	-1.89	-73.30	0.20	7.24
0	**-45**	41.84	-19.00	77.80	20.46

Figure 5.15: Comparison between the EVLST87M01 and STEVAL-NBIOT for the vertical (elevation) position error test [40]

Again, similar to the horizontal position estimation comparison, the STEVAL-NBIOT board showed better vertical position accuracy compared to the EVLST87M01 board with an external antenna.

5.4 Evaluating power consumption

Power consumption was evaluated using the Power Profiler Kit II from Nordic. Both the Ampere Meter, and the Source Mode configurations were used. Three different power consumption measurements were performed. First one was focused on only determining the power consumption of the ST87M01, second on STM32U575, and final measurement was focused on determining the power consumption of the whole system. The Ampere Meter mode was used for measuring consumption of ST87M01 and STM32U575. The Source Mode was used to determine the power consumption of the whole STEVAL-NBIOT board. For all of the three aforementioned measurements, the same firmware was used. This firmware example was designed to periodically (every 10 seconds) acquire the data from the temperature and humidity sensor, send it over the NB-IoT network using UDP protocol, and wait for the response (echo from the server). During the idle periods the STM32U575 system was put into STOP2 low-power mode. For the ST87M01, the sleep mode was enabled, and the modem would go to the low-power mode on its own, autonomously, following the PSM and eDRX operating modes defined by the NB-IoT protocol.

5.4.1 Evaluating power consumption of the ST87M01

Waiting to attach to the NB-IoT base station and register for a valid connection has the biggest impact on the power consumption of the ST87M01. The duration of the time period from the power-up phase until the valid connection has been established can vary a lot. In certain situations it could take tens of minutes, especially if the device is performing this for the very first time, without ever been connected to the network previously. This process can be greatly sped up by manually setting certain parameters to force the selection of the specific mobile network operator. In best cases, the process takes only a few seconds. When in sleep mode, the ST87M01 consumes on average around 90 μA.

At the moment of performing the measurements, it took 33 seconds for the device to establish a valid connection with the NB-IoT base station averaging 34 mA of current with peak registered current going up to 180 mA.

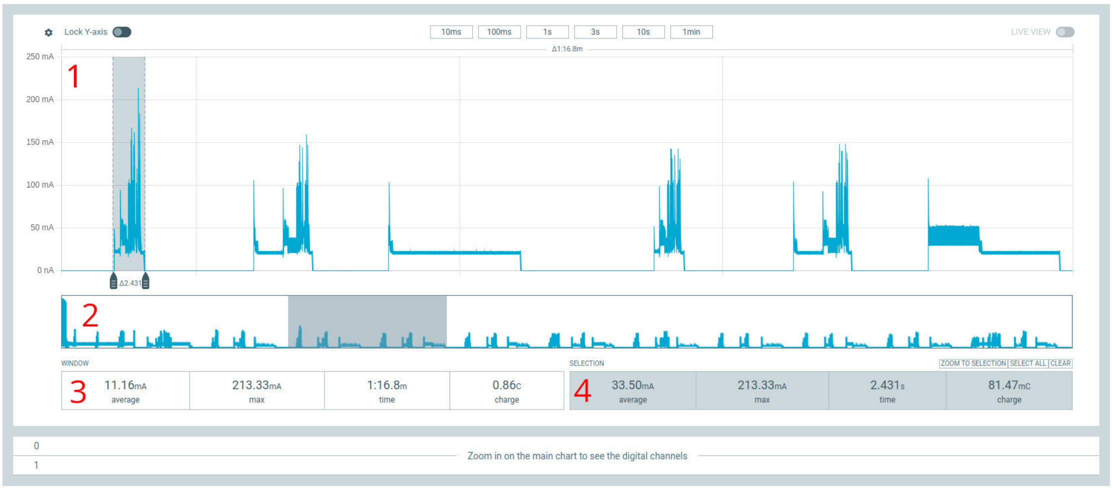
The rest of the data that was gathered is summed up in the table 5.1. Since the data was sent and gathered periodically, data for several periods can be seen in the table 5.1. The number of the period (starting from 1), the duration of the whole period from waking up, performing necessary actions, to going again back to sleep, and the average and maximum current consumption for this period can be seen in the table.

Period	Duration (seconds)	Average current [mA]	Max current [mA]
1	10.6	32.09 mA	170.92
2	3.24	30.21 mA	176.31
3	5.4	31.78 mA	170.92
4	2.4	33.89 mA	213.33
5	4.7	26.55 mA	159.4
6	2.47	32.2 mA	142.54
7	4.64	26.5 mA	148.66
8	2.42	33.87 mA	147.9
9	4.57	26.74 mA	144.07
10	4.82	34.66 mA	142.54

Table 5.1: Power consumption of ST87M01 for UDP send/receive

What can be seen from the data is that the ST87M01 required a couple of seconds for sending the data and for receiving the response. The current consumption was on average around 30 mA. The power consumption graph from the PPKII for the ST87M01 can be seen in Figure 5.16.

The area that sits in the top half in the Figure 5.16 marked with "1" showcases just a portion of the whole data. It gives the ability to zoom in and take a better look at the data points for the selected part of the whole graph. The area just under marked with "2" gives an overview of the whole graph but with lower resolution. Grayed out part of the area marked with "2" tells which part of the whole dataset is visualized in the upper bigger area marked with "1". The measurement in the bottom are split into two parts marked with "3" and "4". The area "3" on the left showcases measures like average current in mA, max current spike registered in mA, time duration and estimated charge in C for the time period that can be seen above in area "1". The right side area "4", slightly grayed out, showcases the same measurements but for the selected time period that is grayed out on the part of the graph from area "1".



3 11.16mA average	213.33mA max	1:16.8m time	0.86C charge
4 33.50mA average	213.33mA max	2.431s time	81.47mC charge

Figure 5.16: Power consumption graph for ST87M01 obtained using PPKII

5.4.2 Evaluating power consumption of the STM32U575

It should be noted that when data was gathered for STM32U575 the measurements were performed from the power-off state, a new measurement cycle was started, focusing now on the consumption of the STM32U575. The power consumption of STM32U575 for the same scenario was on average 21 mA to 22 mA without major fluctuations, for the active periods when the device was running. The power consumption for this periods was almost constant. The data from the table, similar to the one for ST87M01 for active periods, would end up not providing much of an insight. For this reasons the tabular data form was omitted. Only the average current consumption for active periods is mentioned here and also for the periods when the device is sleeping. When the device is sleeping, when it is in the STOP2 mode, the average power consumption no matter the time period is on average around 450 μA. One interesting fact is that there are almost periodical current spikes during this mode. This is likely because the LPTIM1 is active and ticking during STOP2 mode and because LPBAM mode was enabled as well. The current spikes during STOP2 mode can reach the values of up to 3 mA. The overall graph obtained from the PPKII can be seen in Figure 5.17.

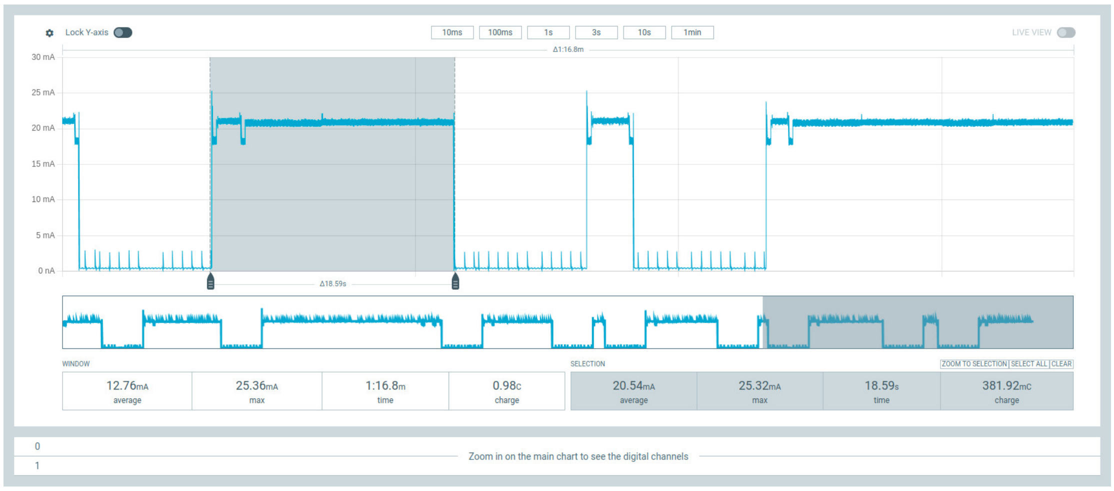


Figure 5.17: Power consumption graph for STM32U575 obtained using PPKII

5.4.3 Evaluating power consumption of STEVAL-NBIOT

Once again, the same analysis and scenario, but this time with the focus on the power consumption of the whole STEVAL-NBIOT device. It was observed that the current consumption values were mostly impacted by the consumption requirements of the ST87M01 module. The registered peak current values that were occurring were present because of the ST87M01 module. No other component on the board was producing peaks that would go as high as 200 mA. Also, the current consumption of the whole device during the active periods consists mainly of the consumption of the STM32U575 and ST87M01. The contribution of the STM32U575 component to the average power consumption was almost constant and identical no matter the operating conditions. The current consumption on average when considering only the active periods was around 50 mA to 55 mA with ST87M01 contributing more to these numbers. This makes sense because of the antenna power requirements when transmitting data. The graph showcasing power consumption of the STEVAL-NBIOT device is presented in Figure 5.18.

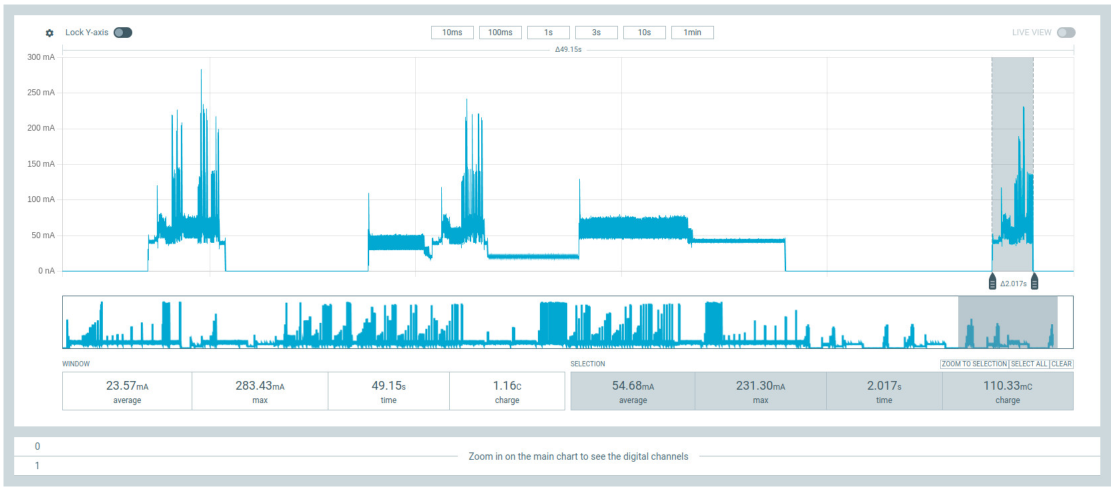


Figure 5.18: Power consumption graph for the STEVAL-NBIOT obtained using PPKII

5.5 Evaluating firmware support for ST's data logging and dashboard infrastructures

The results of the work on firmware examples developed to showcase the features of the STEVAL-NBIOT, as well as to enable out-of-the-box support for easy gathering of data from the sensors, are presented below.

After the board bring-up tests and work on BSP, ensuring that every component was working as it should, the firmware support for ST's data logging and dashboard infrastructure was developed.

5.5.1 ST's data logging infrastructure support

A developed firmware example for this particular case enabled the use of ST's data logging desktop GUI application with the STEVAL-NBIOT. This, in turn, made it possible for the users of the board to perform data acquisitions from the onboard sensors and store the data locally on the PC. This data could be later used for any purpose (e.g., machine learning).

Figures 5.20 and 5.19 show the GUI after connecting the board and starting the acquisition of data for all of the sensors. Figure 5.20 shows the part of the GUI for the accelerometer and barometer. Figure 5.19 shows the part of the GUI for the humidity and temperature sensor. Note that this is just one sensor capable of outputting two different readings, but the data representation is split in the GUI.

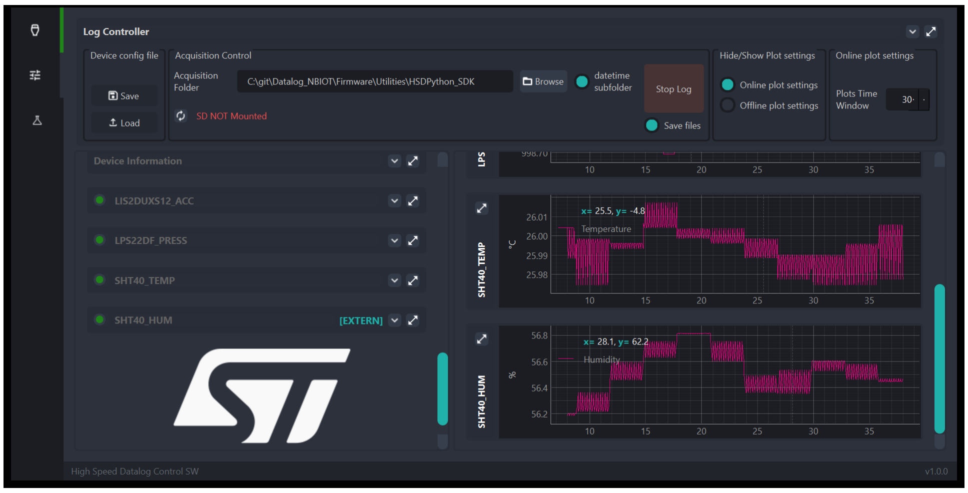


Figure 5.19: ST's data loggin GUI showing humidity and temperature sensor data acquisition

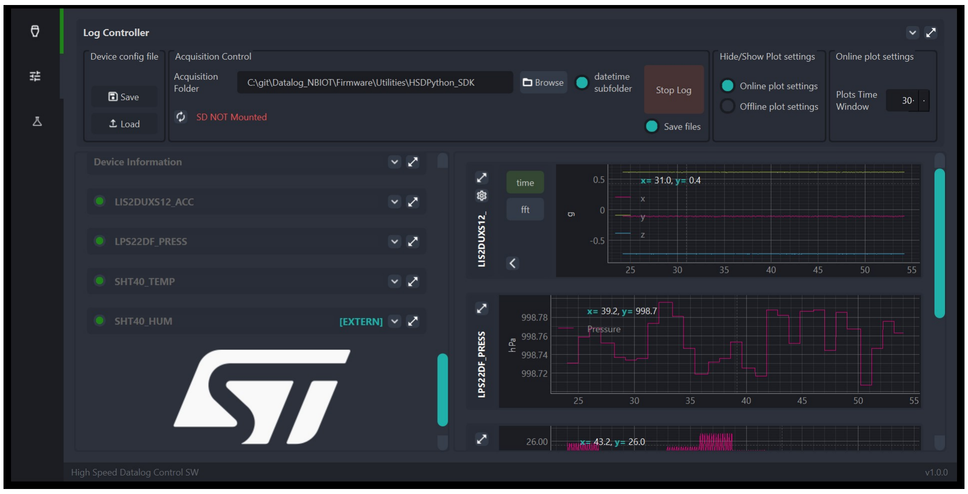


Figure 5.20: ST's data loggin GUI showing accelerometer and barometer data acquisition

5.5.2 ST's dashboard infrastructure support

Since the data logging is done locally via USB connection and the network is never used, an additional example that provides similar functionality but sends the data over the network to the cloud was developed. This example gathers the data from all of the sensors and sends it via the NB-IoT network using the HTTPS protocol to the cloud dashboard. The data report also includes the position data from the

GNSS.

Figure 5.21 gives an overview of the data coming to the dashboard from the barometer sensor.

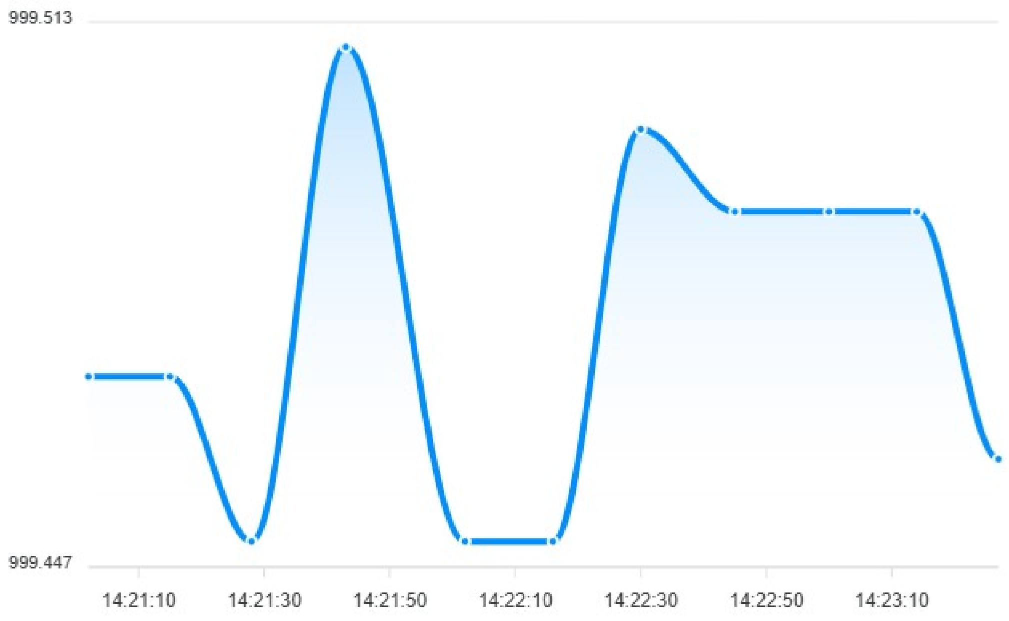


Figure 5.21: ST's dashboard website GUI showing barometer data acquisition

Figure 5.22 gives an overview of the data coming to the dashboard from the temperature and humidity sensor. In this case, the temperature is shown separately.

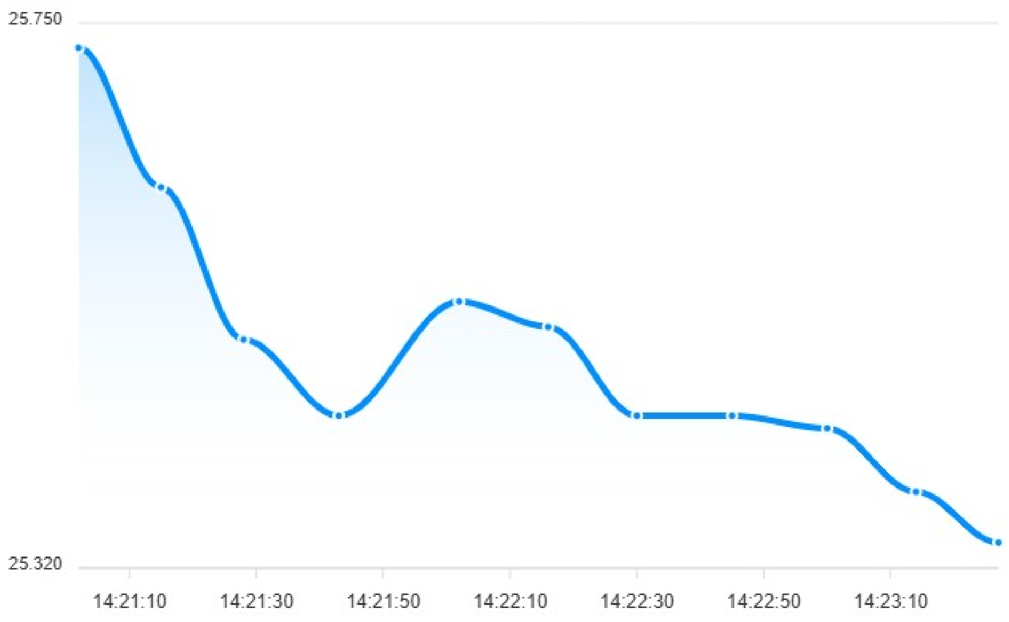


Figure 5.22: ST's dashboard website GUI showing temperature data acquisition

Figure 5.23 gives an overview of the data coming to the dashboard from the temperature and humidity sensor. In this case, the humidity is shown separately.

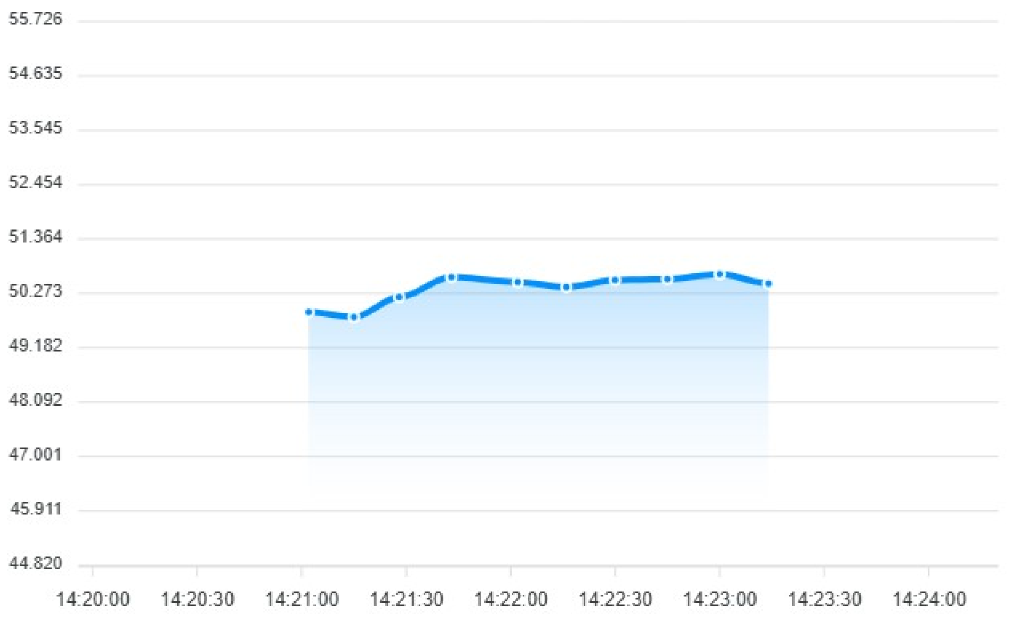


Figure 5.23: ST's dashboard website GUI showing humidity data acquisition

All three of the accelerometer axes from the dashboard overview are included in Figure 5.24.

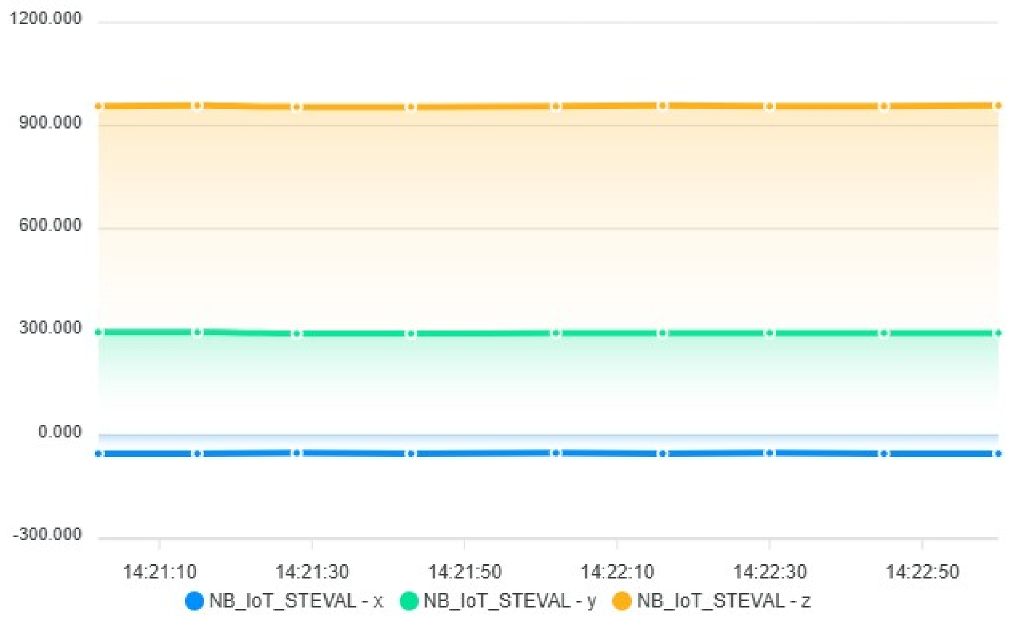
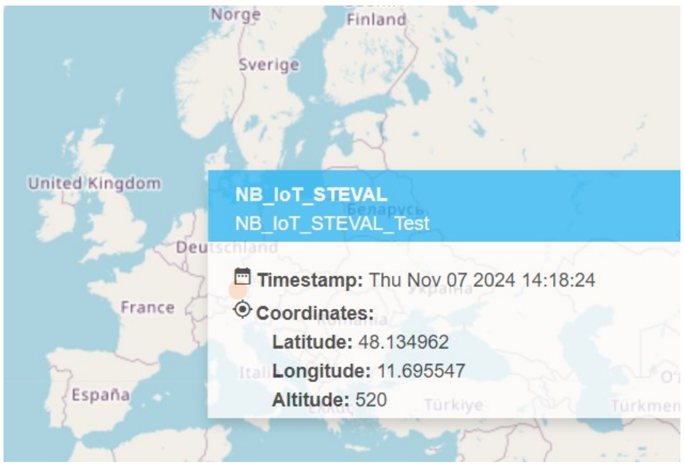


Figure 5.24: ST's dashboard website GUI showing accelerometer data acquisition

Finally, in Figure 5.25, the position data from the GNSS, including longitude, latitude, and elevation, can be seen.



NB_IoT_STEVAL
NB_IoT_STEVAL_Test

Timestamp: Thu Nov 07 2024 14:18:24
Coordinates:
Latitude: 48.134962
Longitude: 11.695547
Altitude: 520

Figure 5.25: ST's dashboard website GUI showing GNSS data

Chapter 6

Conclusion

Although there are a couple of already available development platforms for NB-IoT, given that the protocol is relatively new, the number of available options is still low. With the low number of available options, the choice and flexibility are also limited. Some of the platforms are limited in the hardware components they include out-of-the-box, which generally means that they provide fewer functionalities. The majority of development platforms available combine hardware parts from several different vendors. The higher the number of different vendors, the higher the possibility that if one of them obsoletes some hardware component that makes up the platform, the whole product line for that development platform becomes obsolete. This can be problematic because the developed solutions are expected to operate on average around ten years.

The new STEVAL-NBIOT brings a new development platform for the NB-IoT as well as the GNSS protocol to the market. It provides several power supply options, giving users the flexibility to develop solutions that can be powered by USB, battery, and even by solar energy. Onboard sensors make it possible to develop, with ease, applications that are required to sense and gather environmental data. Provided firmware examples integrate with ST's data logging and dashboard solutions. This gives users out-of-the-box support to try and test the device without even having to write any software. It also gives them a starting point for easier firmware development in case they choose to develop a custom firmware solution. The fact that it uses hardware components made by ST and is also manufactured by the same company provides security to the users. It enables easier and guaranteed support because the end-users depend on only one company.

The newly built platform has overall good performance in regards to antenna efficiency, power consumption, and firmware support. However, this can definitely be improved and more optimized in the future. Firmware power optimizations could be performed for scenarios that involve gathering data from the sensors. Also, firmware examples that showcase the usage of other relevant IoT protocols such as LWM2M or MQTT could be implemented to provide even more flexibility and ease of development to the users.

In general, with more flexibility and better support, users should have their development experience made easier and better when working on new NB-IoT solutions, all thanks to the newly developed NB-IoT low-power sensing platform.

Bibliography

[1] Velos. *A Brief Guide to Low Power Wide Area Network (LPWAN) LPWAN Guide White Paper Cover.* 2024. URL: `https://info.velosiot.com/a-brief-guide-to-low-power-wide-area-network` (visited on 11/20/2024) (cit. on p. 3).

[2] Velos. *Different LPWAN Technologies Explained.* 2024. URL: `https://blog.velosiot.com/different-lpwan-technologies-explained` (visited on 11/20/2024) (cit. on pp. 4, 5).

[3] Min Chen, Yiming Miao, Yixue Hao, and Kai Hwang. «Narrow Band Internet of Things». In: *IEEE Access* 5 (2017), pp. 20557–20577. DOI: `10.1109/ACCESS.2017.2751586` (cit. on pp. 6, 8–11).

[4] Sakshi Popli, Rakesh Jha, and Sanjeev Jain. «A Survey on Energy Efficient Narrowband Internet of Things (NBIoT): Architecture, Application and Challenges». In: *IEEE Access* PP (Nov. 2018), pp. 1–1. DOI: `10.1109/ACCESS.2018.2881533` (cit. on pp. 7, 9, 10).

[5] Alain L. Kornhauser. *Global Navigation Satellite System (GNSS).* 2006. URL: `https://www.princeton.edu/~alaink/Orf467F07/GNSS.pdf` (visited on 10/01/2024) (cit. on pp. 11, 13, 14).

[6] New Space Economy. *Global Navigation Satellite Systems (GNSS): A Comparative Overview of Satellite Constellations.* 2024. URL: `https://newspaceeconomy.ca/2024/09/09/global-navigation-satellite-systems-gnss-a-comparative-overview-of-satellite-constellations/` (visited on 10/11/2024) (cit. on p. 12).

[7] Dr. Robert Thirsk. *An Introduction to GNSS - A primer in using Global Navigation Satellite Systems for positioning and autonomy.* 2024. URL: `https://novatel.com/an-introduction-to-gnss` (visited on 10/11/2024) (cit. on pp. 12, 13).

[8] Brian Amos. *Hands-On RTOS with Microcontrollers: Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools.* Birmingham, UK: Packt Publishing, 2020 (cit. on pp. 14, 19).

[9] Richard Barry and The FreeRTOS Team. *Mastering the FreeRTOS™ Real Time Kernel - A Hands-On Tutorial Guide*. 2024. URL: `https://github.com/FreeRTOS/FreeRTOS-Kernel-Book/releases/download/V1.1.0/Mastering-the-FreeRTOS-Real-Time-Kernel.v1.1.0.pdf` (visited on 10/02/2024) (cit. on pp. 15–18).

[10] STMicroelectronics. *User manual - Developing applications on STM32Cube with RTOS*. 2019. URL: `https://www.st.com/resource/en/user_manual/um1722-developing-applications-on-stm32cube-with-rtos-stmicroelectronics.pdf` (visited on 10/01/2024) (cit. on pp. 18, 19).

[11] STMicroelectronics. *STM32U5 Power Management*. 2021. URL: `https://www.st.com/content/ccc/resource/training/technical/product_training/group1/95/38/81/9b/cb/0d/43/89/STM32U5-System-Power-management_PWRMNGMNT/files/STM32U5-System-Power-management_PWRMNGMNT.pdf/_jcr_content/translations/en.STM32U5-System-Power-management_PWRMNGMNT.pdf` (visited on 10/16/2024) (cit. on pp. 20, 40, 41).

[12] Microsoft. *Digital Twins Definition Language (DTDL)*. 2023. URL: `https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/DTDL.v2.md` (visited on 10/04/2024) (cit. on pp. 20, 21).

[13] Simbase. *Narrowband*. 2024. URL: `https://www.narrowband.com/` (visited on 10/11/2024) (cit. on p. 23).

[14] Quectel. *LPWA BC65 NB-IoT*. 2024. URL: `https://www.quectel.com/product/lpwa-bc65-nb-iot/` (visited on 10/11/2024) (cit. on p. 24).

[15] Quectel. *BC65-TE-B User Guide*. 2020. URL: `https://source.z2data.com/2021/3/24/8/57/54/371/Filed18233.pdf` (visited on 10/11/2024) (cit. on pp. 24, 25).

[16] Telit. *ME910C1 Series*. 2024. URL: `https://www.telit.com/devices/me910c1-series/` (visited on 10/31/2024) (cit. on p. 25).

[17] Telit. *Bravo: An Evaluation Kit for Cellular LPWA*. 2024. URL: `https://www.telit.com/support-tools/development-evaluation-kits/telit-bravo-evaluation-kit/` (visited on 10/31/2024) (cit. on p. 26).

[18] Nordic Semiconductor. *nRF9160*. 2024. URL: `https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF9160-SiP-PB-v2.1.pdf?sc_trk=%7BDownload%20product%20brief%7D` (visited on 10/31/2024) (cit. on p. 27).

[19] Nordic Semiconductor. *nRF9160 DK*. 2024. URL: `https://docs-be.nordicsemi.com/bundle/ug_nrf9160_dk/attach/nRF9160_DK_HW_User_Guide_v1.1.0.pdf?_LANG=enus` (visited on 10/31/2024) (cit. on pp. 27, 28).

[20] Nordic Semiconductor. *Nordic Thingy:91*. 2024. URL: `https://docs-be.nordicsemi.com/bundle/ug_thingy91/attach/Thingy91_UG.pdf?_LANG=enus` (visited on 10/31/2024) (cit. on p. 29).

[21] STMicroelectronics. *Data brief - ST87M01*. 2024. URL: `https://www.st.com/resource/en/data_brief/st87m01.pdf` (visited on 10/29/2024) (cit. on pp. 31, 32, 50).

[22] Telit. *AT Commands Reference Guide*. 2024. URL: `https://www.sparkfun.com/datasheets/Cellular%20Modules/AT_Commands_Reference_Guide_r0.pdf` (visited on 10/30/2024) (cit. on p. 32).

[23] STMicroelectronics. *EVKITST87M01 user manual*. 2024. URL: `https://www.st.com/resource/en/user_manual/um3382-evkitst87m011-stmicroelectronics.pdf` (visited on 10/29/2024) (cit. on p. 34).

[24] STMicroelectronics. «ST87Mxx GUI User Manual». Unpublished (cit. on pp. 35, 36).

[25] STMicroelectronics. «Easy Connect Library Application Note». Unpublished (cit. on p. 36).

[26] STMicroelectronics. *Data brief - NUCLEO-STM32U5*. 2024. URL: `https://www.st.com/resource/en/data_brief/nucleo-l496zg.pdf` (visited on 10/01/2024) (cit. on pp. 38, 50).

[27] STMicroelectronics. *STM32U575xx datasheet*. 2024. URL: `https://www.st.com/resource/en/datasheet/stm32u575ag.pdf` (visited on 10/03/2024) (cit. on pp. 39, 40).

[28] STMicroelectronics. «Vespucci WP2 Architecture Document». Unpublished (cit. on pp. 41–44).

[29] Microsoft. *Azure IoT documentation*. 2023. URL: `https://learn.microsoft.com/en-us/azure/iot/` (visited on 10/23/2024) (cit. on p. 43).

[30] STMicroelectronics. *Getting started with the STM32Cube function pack for high speed datalogging and ultrasound processing*. 2024. URL: `https://www.st.com/resource/en/user_manual/um3106-getting-started-with-the-stm32cube-function-pack-for-high-speed-datalogging-and-ultrasound-processing-stmicroelectronics.pdf` (visited on 10/30/2024) (cit. on pp. 45, 46).

[31] STMicroelectronics. *FP-AI-MONITOR1 an introduction to the technology behind*. 2024. URL: `https://wiki.st.com/stm32mcu/wiki/AI:FP-AI-MONITOR1_an_introduction_to_the_technology_behind` (visited on 10/31/2024) (cit. on p. 45).

[32] STMicroelectronics. *DSH-ASSETRACKING - Cloud Amazon-based web application for asset tracking*. 2021. URL: `https://www.st.com/resource/en/data_brief/dsh-assetracking.pdf` (visited on 11/20/2024) (cit. on pp. 46, 47).

[33] STMicroelectronics. *LIS2DUXS12 Datasheet*. 2024. URL: `https://www.st.com/resource/en/datasheet/lis2duxs12.pdf` (visited on 10/24/2024) (cit. on p. 48).

[34] STMicroelectronics. *LPS22DF Datasheet*. 2024. URL: `https://www.st.com/resource/en/datasheet/lps22df.pdf` (visited on 10/24/2024) (cit. on p. 49).

[35] Sensirion. *SHT40 Datasheet*. 2024. URL: `https://sensirion.com/resource/datasheet/sht4x` (visited on 10/24/2024) (cit. on p. 49).

[36] Nordic Semiconductor. *Power Profiler Kit II*. 2024. URL: `https://docs.nordicsemi.com/bundle/ug_ppk2/page/UG/ppk/PPK_user_guide_Intro.html` (visited on 10/16/2024) (cit. on p. 49).

[37] STMicroelectronics. *Data brief - X-NUCLEO-PGEEZ1*. 2024. URL: `https://www.st.com/resource/en/data_brief/x-nucleo-pgeez1.pdf` (visited on 09/13/2024) (cit. on p. 50).

[38] STMicroelectronics. *Data brief - X-NUCLEO-IKS4A1*. 2024. URL: `https://www.st.com/resource/en/data_brief/x-nucleo-iks4a1.pdf` (visited on 09/13/2024) (cit. on p. 50).

[39] STMicroelectronics. «NB-IoT Ultra-low power sensing platform (presentation slides)». Unpublished (cit. on pp. 64–68).

[40] Claudio Pessina (STMicroelectronics). «Comparison of GNSS performances between EVKIT and STEVAL-NBIOTV1 boards». Unpublished (cit. on pp. 78–85).