Bachelor Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2019

GPS-Tracking Device with Long Range and Bluetooth Low Energy Communication

Rasmus Oliv

Bachelor Thesis in Electrical Engineering

GPS-Tracking Device with Long Range and Bluetooth Low Energy Communication

Rasmus Oliv

LiTH-ISY-EX-ET--19/0487—SE

Supervisor:
Yonatan Kifle
ISY, Linköping University

Examiner:
J Jacob Wikner
ISY, Linköping University

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

The thesis is about the construction of a GPS-tracker that can read NFC (Near Field Communication)-tags and communicate with LoRa (Long Range) and BLE (Bluetooth low energy) and investigate which of the components in the GPS-tracker that consumes most power. The usage area for the GPS-tracker is to make the work on disaster affected sites more efficient and secure by having an operation leader that can organizing the operation with help of the information provided by the GPS-trackers that are placed on the injured people and recuing personnel. The GPS-tracker is built around the sensor development kit Thingy:52 from Nordic Semiconductor. The Firmware (FW) for the Thingy:52 is developed by modifying the provided factory FW by Nordic Semiconductor. The GPS-module and the NFC-reader showed to be the most power consuming parts of the GPS-tracker. An energy optimization proposal for these parts are given in the report. A proposal to a circuit diagram for the GPS-tracker is also given in the report, that can be used for future miniaturization of the GPS-tracker.

Sammanfattning

Projektet har innefattat att ta fram en GPS-spårsändare som kan läsa NFC (Near Field Communication)-taggar, kommunicera med LoRa (Long Range) och BLE (Bluetooth low energy) samt undersöka vilka av GPS-spårsändarens olika delar som konsumerar mest energi. Användningsområdet för GPS-spårsändaren är att effektivisera räddningsinsatser på skadeplatser där det finns skadade människor exempelvis efter en översvämning eller terroristattack. Effektiviseringen är tänkt ska ske genom att en operationsledare styr räddningsinsatsen med hjälp av informationen som skickas från GPS-spårsändarna som kommer att bäras av skadade personer och räddningspersonalen på skadeplatsen. GPS-spårsändaren är utvecklad kring sensorutvecklings kittet Thingy:52 från Nordic Semiconductor och dess mjukvara har utvecklats genom att modifiera den mjukvara som Nordic Semiconductor har utvecklat för Thingy:52. De delar av GPS-spårsändaren som visade sig konsumera mest energi var GPS-modulen och NFC-läsaren. I rapporten finns energioptimerings förslag för dessa delar. Rapporten innehåller även ett förslag till ett kretsschema för GPS-spårsändaren som kan användas vid framtida miniatyrisering av GPS-spårsändaren.

Acknowledgments

I would like to thank my examiner J Jacob Wikner and supervisor Yonatan Kifle for helping me during the project by discussing problems and giving helpful advice.

I also want to thank Mangus Bång for making the thesis possible by ordering this project.

Definitions

Abbreviation	Meaning	Explanation
BLE	Bluetooth low energy	Low energy version of Bluetooth communication.
LPWAN	Low power wide area network	Type of wireless network that is built for long distances communication with low power consumption.
LoRa	Long Range	Communication technology that works in LPWAN.
LoRaWAN	Long Range Wide Area Network	The protocol LoRa-devices uses for communication.
CSS	Chirp Spread Spectrum	A type of radio modulation technique which is for example used by LoRa-devices.
ISM band	Industrial, Scientific and Medical band	Unlicensed frequency bands.
PHY	Physical layer	A layer that consists of electronic circuits.
OTAA	Over the air activation	Activation method for LoRa communication.
RETTS	Rapid Emergency Triage and Treatment System	A system that is used for doing assessment of how fast a person needs medical care.
NFC	Near Field Communication	Over the air communication technology with a range up to ten centimeters.
SoC	System-on-a-chip	A chip that consists of different functional blocks that together build up a system.
DK	Development Kit	A set of tools that can be used for facilitating the development process.
IDE	Integrated Development Environment	A program that normally consists of text editor, debugger and a compiler.
FW	Firmware	Software for some hardware.
SDK	Software development kit	A set of tools that can be used for facilitating the software development.
TTN	The Things Network	A LoRa community which is working with building a decentralized data network that can connect to internet by using LoRaWAN.
UUID	Universally Unique IDentifier	A number that is used for identifying information in a computer system.
LMIC	LoraMAC-in-C	A library built for handling LoRa communication.
GATT	Generic Attribute Profile	Defines how the data exchange between BLE devices will work.
NDEF	NFC Data Exchange Format	Format of the data which NFC-devices uses when exchanging information between devices.
MSB	Myndigheten för samhällsskydd och beredskap	Swedish authority which is working with development and handling of protection and preparedness for crises and accidents in the society.
AppSKey	Application Session Key	A key which is used for LoRa communication.
NwkSKey	Network Session Key	A key which is used for LoRa communication.

Content

Table of figures

1 Introduction

The thesis is about building a GPS-tracker that can communicate with internet through LoRa and Bluetooth low energy (BLE) and the usage area for the tracker is to improve the way of working on disaster sites.

The GPS-tracker is planned to be used by fireman, paramedics and injured people at disaster sites. The device will be equipped with a NFC-reader for assigning the need for care levels to the injured people.

For example, a disaster has just occurred and there are dead and injured people at the disaster site. When the rescuing personnel arrives to the disaster site, they can put on their own GPS-trackers and bring a bunch of other GPS-trackers to place on injured people on the site. When the rescuing personnel finds people at the site, they can do an assessment of the need of care of the injured by doing a NFC-scan on the injured persons GPS-tracker. The GPS-tracker will send information (ex: GPS-coordinates, condition status and sensor values) to the internet when the device is triggered by some sensor.

An operation leader can monitor the whole situation from a computer and see where the personnel and the injured people are and hopefully make efficient decisions on where the personnel should go. Doctors and nurses in hospitals can also do NFC-scans on injured peoples GPS-trackers when they arrive to the hospital. To inform the rescue personnel at the disaster site that the injured people have arrived at the hospital.

This project is funded by JPI Climate and co-funded by the European Union
Particular research funding agencies: FORMAS (Sweden), RCN
(Norway), NWO (The Netherlands), and FCT (Portugal)

2019 Team of the CitizenSensing Project

1.1 Motivation

The Swedish authority myndigheten för samhällsskydd och beredskap (MSB) is working on a project to develop new methods and technique to handle future disaster sites better. The motivation for this thesis is to contribute to MSB project by developing a GPS-tracker that can make the rescue operations on disaster sites more efficient and secure.

1.2 Purpose

The purpose of this thesis is to make the work at disaster sites more efficient and secure. If GPS-trackers are placed on fireman, paramedics and injured people on disaster sites will possibly make it easier for a rescue operation leader to get a good overview of the situation and hopefully make more efficient and better decisions with help of the information provided by the GPS-trackers.

All information provided by the GPS-trackers during a disaster scenario will be saved. The collected information can then be studied afterwards. By studying the information after the disaster scenario, makes it possible to correct the errors that occurred and strengthen the things that were handled well for the next disaster situation.

1.3 Delimitations

The GPS-tracker will be based on the sensor development kit Nordic Thingy:52 from Nordic semiconductor. The usage area for the GPS-tracker will be focused on disaster affected sites. The project will be limited to build a prototype of a GPS-tracker as a proof of concept for the usage of GPS-tracker on disaster sites.

1.4 Research question

How will the power consumption be distributed between the different components in a GPS-tracker with the capability of reading NFC-tags and communicate with LoRa and BLE?

What effects can the usage of GPS-trackers with the following features: LoRa communication, BLE communication and NFC-reading have on a rescue operation on a disaster site?

2 Theory

The theory chapter will contain the theoretic background to the different techniques used in the project and a description of the triage system that rescue personnel uses for doing assessments of injured people. At the end of the theory chapter is a description about the hardware and software that has been used during the project.

The theory chapter will also include information about BLE and the main difference between BLE and Bluetooth are shown in Figure 1.

Type	BLE	Bluetooth
Channels	40, 2 MHZ bandwidth per channel	79, 1 MHZ bandwidth per channel
Data rate	125 Kb/s – 2 Mb/s	1 Mb/s – 3 Mb/s
Power consumption	~0.01x – 0.5x of reference	1 (reference value)
Network topologies	Point to Point Mesh Broadcast	Point to Point

Figure 1 A table that is showing the main differences between Bluetooth low energy and Bluetooth [1].

2.1 Bluetooth low energy

BLE operating in the unlicensed 2.4 GHZ industrial, scientific and medical (ISM) band and it is communicating through 40 channels where three channels are advertising channels which are used for establishing connections between devices and for broadcasting purpose and 37 channels are used as data channels [1] [2]. The data rates be adjusted from 125 kb/s to 2 Mb/s and the power levels can be adjusted from 1 mW to 100 mW and this is possible because of BLE usage of multiple physical layer (PHY) [1]. BLE 5.0 has a maximum range of 400 m and 1000 m outdoor in an open field [3].

In the BLE topology exists four roles:

- Broadcaster: advertising packets (transmitting data), cannot connect to other BLE devices, the purpose is to announce it presence for other devices [2].
- Observer: receives advertised packets from other units, do not connect to other BLE units [2].
- Peripheral: connects to central devices in a slave role [2].
- Central: can have multiple connection and can act as a central and peripheral device at the same time [2].

The GPS-tracker developed in this project has the role of a peripheral device in the BLE implementation.

2.1.1 Bluetooth low energy mesh network

BLE mesh networks makes it possible to send messages between and via other BLE units and in that way increases the networks connectivity and the communication range [4]. By comparing the star topology in Figure 3 and the BLE mesh network topology in Figure 2 it is not hard to see that the BLE mesh network has much larger communication range than the star topology.

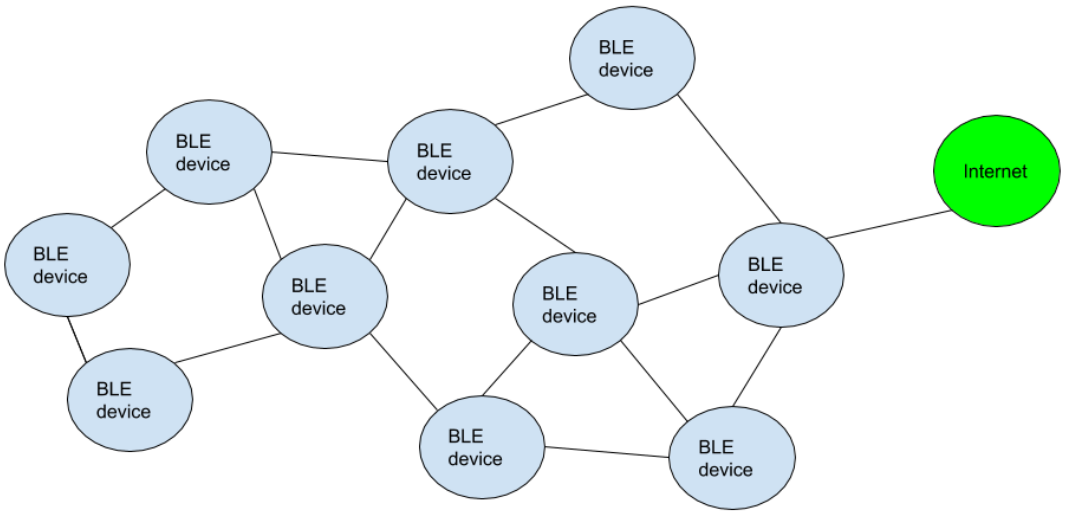


Figure 2 Topology illustration over a Bluetooth low energy mesh network where one of the nodes in the mesh has connection to internet.

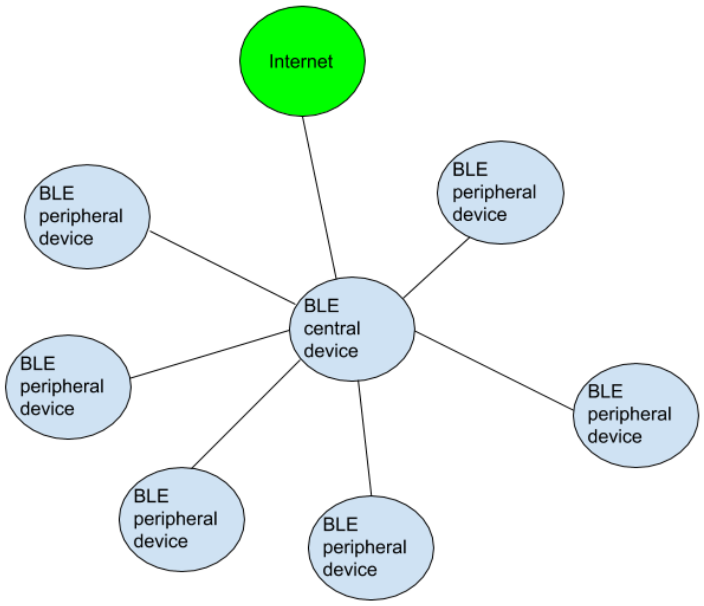


Figure 3 Star topology illustration of a Bluetooth low energy network with six peripheral devices connected to a central device with internet connection.

2.2 Long range

Long range (LoRa) is a low power wide area network (LPWAN) that uses the protocol long range wide area network (LoRaWAN) to communicate and the radio modulation technique chirp spread spectrum (CSS) [5]. Maximum data rate for LoRa is 50 kb/s downlink and 250 b/s uplink and the communications range is around five km in urban environment and 15 km in rural environment [5]. LoRa uses the unlicensed ISM bands on frequency 869 MHz in Europe and 915 MHz in North America [6]. Figure 4 down below illustrates the topology over a LoRa network, where it shown how the communication between LoRa-modules and a PC is built up.

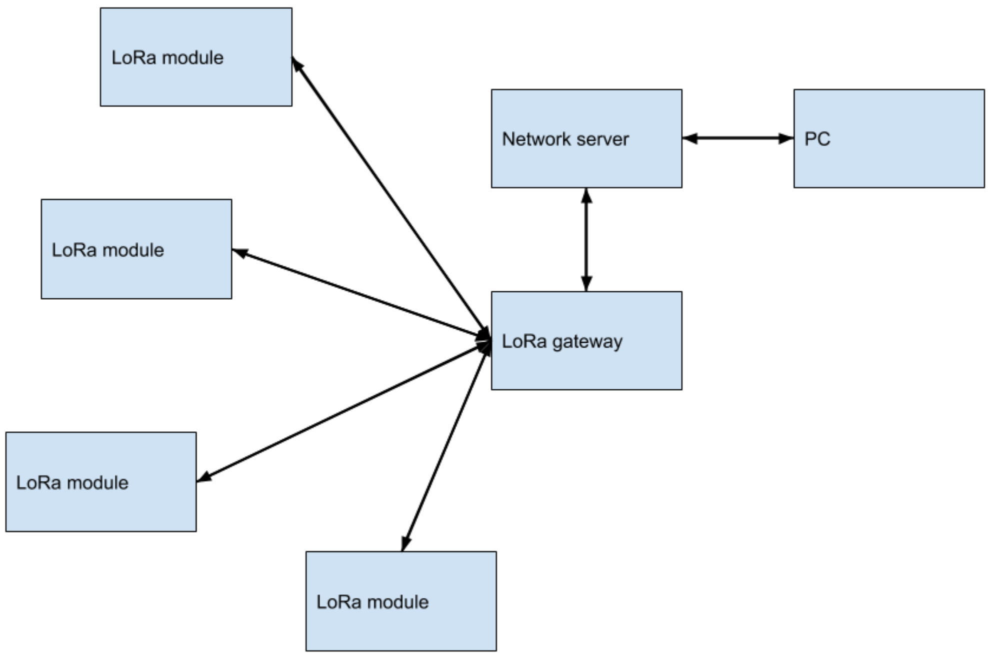


Figure 4 Topology illustration of a long range network consisting of four long range modules that are connected to a long range gateway which is connected to a networks sever which is communicating with a PC.

2.2.1 Over the air activation

To start sending information with the LoRaWAN protocol an activation of the LoRa device is required. The activation can be performed by activation by personalization (ABP) or over the air activation (OTAA) [7]. The way of doing the activation in this project has been OTAA and therefore is the theoretical background only given for OTAA and not the ABP. How the OTAA works is illustrated in Figure 5.

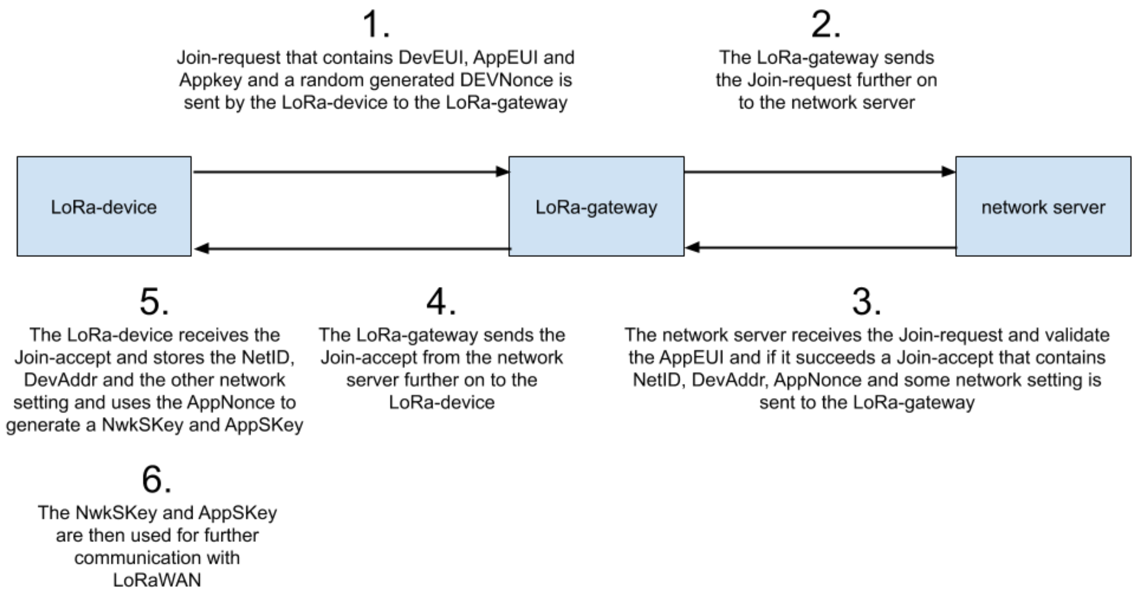

1.
Join-request that contains DevEUI, AppEUI and Appkey and a random generated DEVNonce is sent by the LoRa-device to the LoRa-gateway
2.
The LoRa-gateway sends the Join-request further on to the network server
LoRa-device
LoRa-gateway
network server
5.
The LoRa-device receives the Join-accept and stores the NetID, DevAddr and the other network setting and uses the AppNonce to generate a NwkSKey and AppSKey
4.
The LoRa-gateway sends the Join-accept from the network server further on to the LoRa-device
3.
The network server receives the Join-request and validate the AppEUI and if it succeeds a Join-accept that contains NetID, DevAddr, AppNonce and some network setting is sent to the LoRa-gateway
6.
The NwkSKey and AppSKey are then used for further communication with LoRaWAN

Figure 5 An illustration over the process of activating the long range communication for a long range device by using the activation method over the air activation [7].

The reason for choosing OTAA instead of ABP is because it is more secure since the network session key (NwkSKey) and the application session key (AppSKey) are generated during the network joining procedure and therefore the keys cannot get into the wrong hands before the activation is performed [7]. If the ABP is used, the NwkSKey and the AppSKey are statically stored in the LoRa-device and the risk of the keys getting into the wrong hands before activation exists [7].

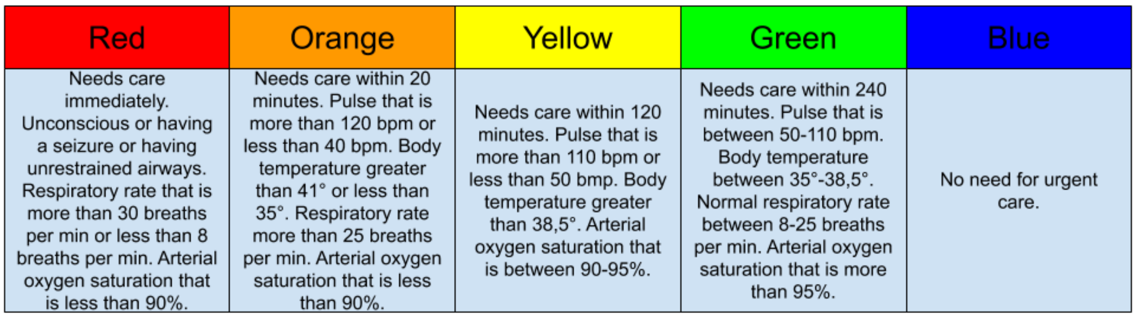
2.2.2 The things network

The things network (TTN) is a community which is working with building a decentralized data network that can connect to internet by using LoRaWAN [8]. On TTN website is it possible to register a LoRa-devices and receive the necessary keys to be able to communicate with a LoRa-device through TTN. On the TTN website is it also possible to open a console and monitor all received messages from the LoRa-device.

2.3 Triage

Triage systems are used to do assessment on patients and to sort them into different levels based on the need of care [9]. Triage is the first step in the process of taking care of injured people and the most common triage system in Swedish emergency departments is the rapid emergency triage and treatment system (RETTS) [9].

RETTS consists of a combinational assessment which consists of the reason for seeking care and the assessment of the critical physiological functions [9]. The reason for searching care can change the priority to a higher priority level but not to a lower level [9]. The combined assessment can be divided into five priority levels that defines the need for care [9]. Each one of the priority levels are represented with a unique color code [9]. The priority levels define how fast the patient needs care [9]. A description for each of the five color are shown in Figure 6.



Red	Orange	Yellow	Green	Blue
Needs care immediately. Unconscious or having a seizure or having unrestrained airways. Respiratory rate that is more than 30 breaths per min or less than 8 breaths per min. Arterial oxygen saturation that is less than 90%.	Needs care within 20 minutes. Pulse that is more than 120 bpm or less than 40 bpm. Body temperature greater than 41° or less than 35°. Respiratory rate more than 25 breaths per min. Arterial oxygen saturation that is less than 90%.	Needs care within 120 minutes. Pulse that is more than 110 bpm or less than 50 bmp. Body temperature greater than 38,5°. Arterial oxygen saturation that is between 90-95%.	Needs care within 240 minutes. Pulse that is between 50-110 bpm. Body temperature between 35°-38,5°. Normal respiratory rate between 8-25 breaths per min. Arterial oxygen saturation that is more than 95%.	No need for urgent care.

Figure 6 Explanation of what the different color codes means in the rapid emergency triage and treatment system [9].

2.4 Near field communication

The NFC technology communicates over the air and it is based on a radio frequency field that uses the base frequency 13.56 MHz [10]. The communication range for NFC is up to ten centimeters [10]. The NFC data exchange format (NDEF) is used when sending data between two NFC devices.

A NDEF-message consists of one or several records [10]. A record contains an identifier for the record, the length of the record, what kind of information the records payload contains e.g. URL and the payload itself [10]. See Figure 7 for an illustration of the structure of a NDEF-message with one record and how the record format for the NFC-tags used during the project looks.

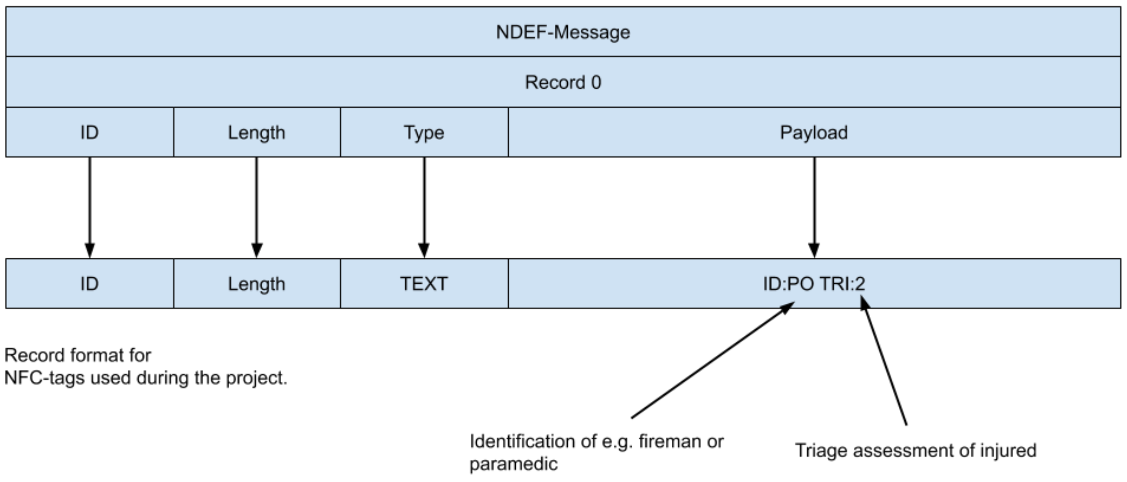

NDEF-Message
Record 0
ID
Length
Type
Payload
ID
Length
TEXT
ID:PO TRI:2
Record format for NFC-tags used during the project.
Identification of e.g. fireman or paramedic
Triage assessment of injured

Figure 7 An illustration over the NFC data exchange format-message and what information the NFC-tags used during the project contained.

It exists serval different NFC-tag types e.g. type 1, type 2, type 3 and type 4 [11]. The tag types distinguish from each other by having different formats and capacities [11], the differences are illustrated in Figure 8. The tag type 2 is used in this project which got a communication speed of 106 kb/s and a basic memory size of 48 B which can be expanded to 2 kB [11].

The reason for choosing tag type 2 was because it existed an example program for NFC-reading of tag type 2 provided by Nordic Semiconductor. Which was used to speed up the implementation of the NFC functionality in the project.

NFC-tag type	Type 1	Type 2	Type 3	Type 4
Communication speed	106 kb/s	106 kb/s	212 kb/s	106 kb/s – 424 kb/s
Memory size	96 B – 2 kB	48 B – 2 kB	2 kB	32 kB
Format	ISO14443A	ISO14443A	ISO18092	ISO14443A and ISO14443B

Figure 8 Table over the capacity and format differences between NFC-tag types [11].

2.5 Hardware

The hardware chapter will present a description about all the hardware that has been used during the project.

The whole project is built around the sensor development board Nordic Thingy:52 from Nordic semiconductor [12]. Thingy:52 is constructed around the nRF52832 Bluetooth 5 system-on-a-chip (SoC) and the SoC is connected to several different sensors e.g. accelerometer, temperature and humidity. Thingy:52 it also equipped with a microphone and a speaker [12].

The programming of the Thingy:52 was done through a nRF52 Development Kit (DK) board from Nordic semiconductor which was connected to a computer with USB cable. A 10-pin SWD cable was connected between the Thingy:52 and the nRF52 DK [13]. The programming setup is illustrated in Figure 9.

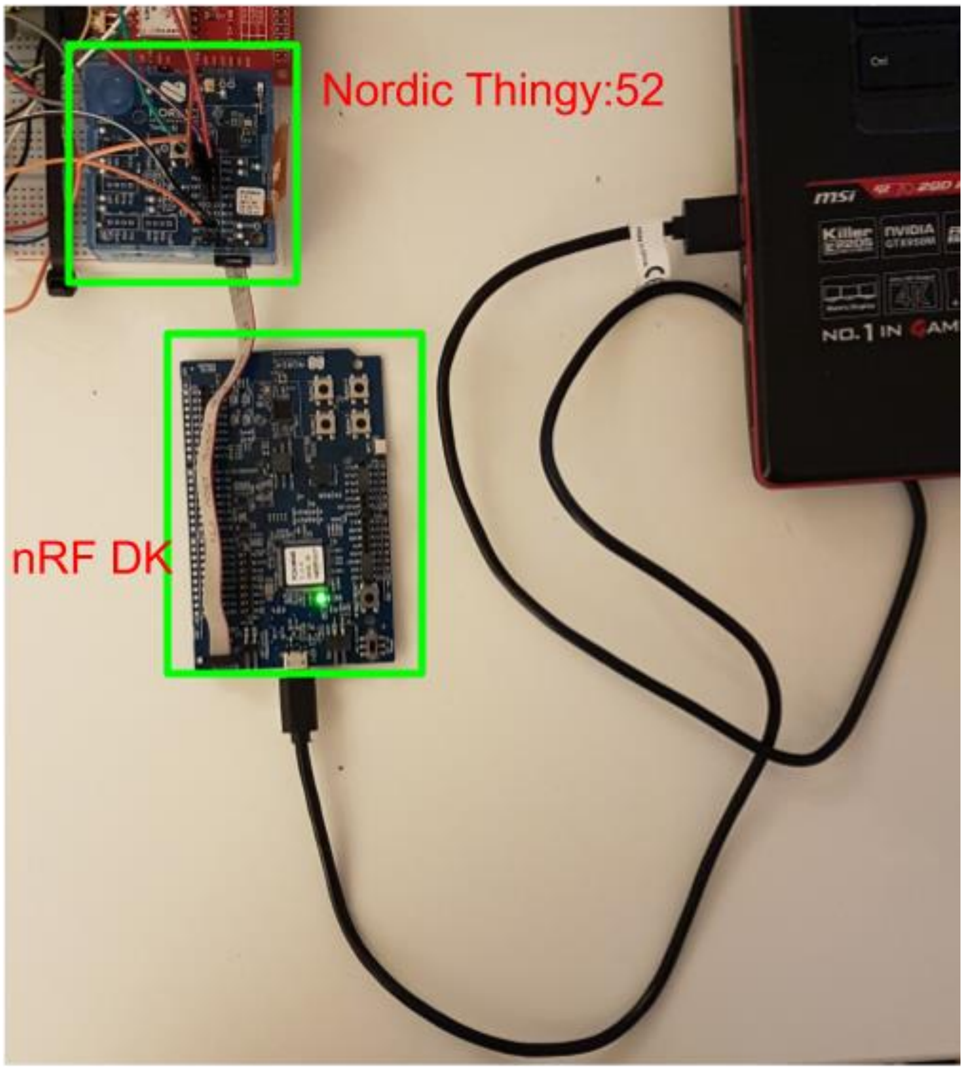


Figure 9 Illustration over the programming setup for Thingy:52 where the nRF DK and Thingy:52 is connected with a 10-pin SWD cable and the computer and Thingy:52 is connect with a USB cable.

The PN532 NFC-reader from Adafruit was used to handle reading of NFC-tag with information of rescue personnel id and triage assessments [14]. To receive GPS-coordinates and the current time was the L80 GPS component on the LoRa/GPS HAT from Dragino used [15].

For handling the LoRa communication was the LoRa Mini module from Dragino used, which is an Arduino compatible board that is based on the microcontroller ATmega328P [16]. For programming the LoRa-module a USB to UART cable called TTL-235R-3V3 was connected between the computer and the LoRa-module. The cable was also used for trying different settings on GPS-module from the computer.

An Arduino UNO was used for measuring the voltage on the GPS-tracker circuit and those voltage measurements were then used for calculating the power consumption for the different modules on the GPS-tracker.

Figure 10 illustrates all coherent hardware to the GPS-tracker hooked up on a breadboard.

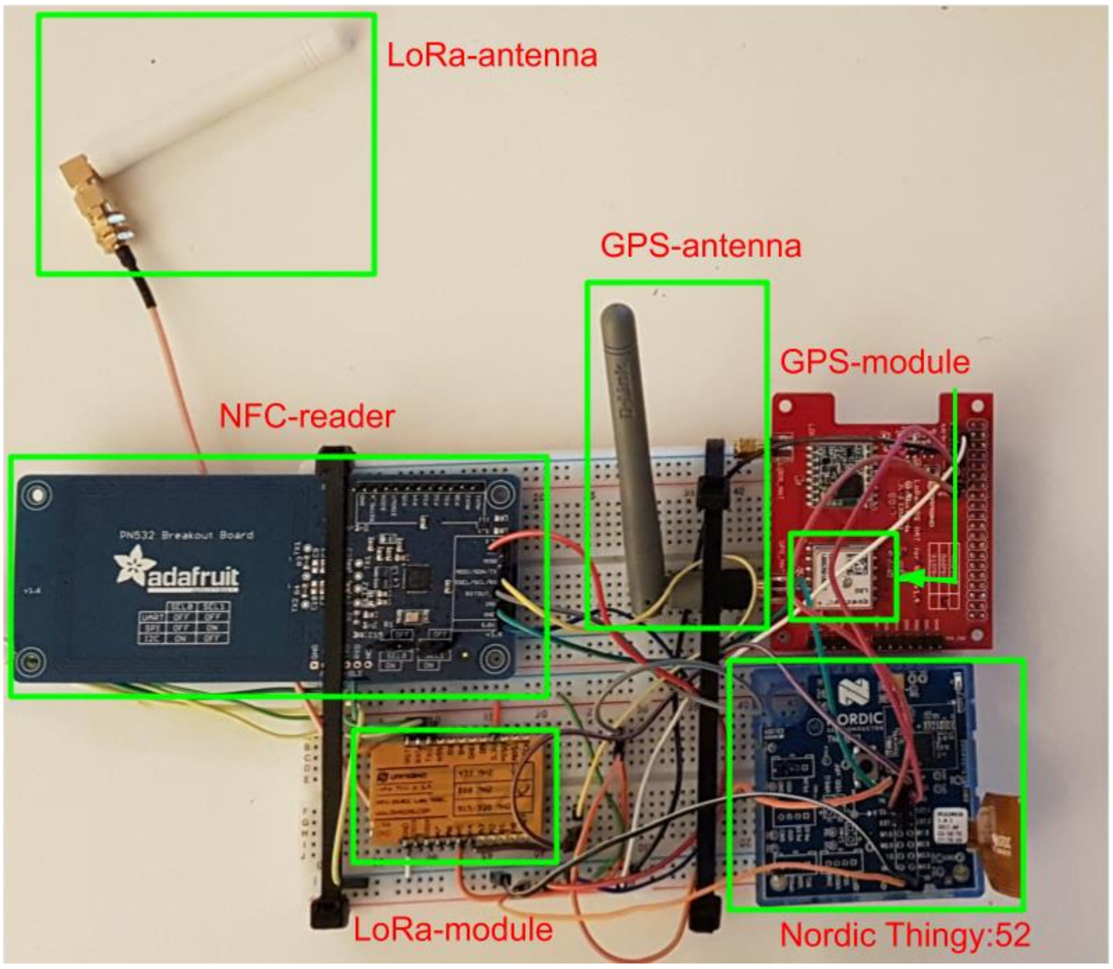


Figure 10 All hardware that the GPS-tracker is constructed of hooked up on a bread board.

2.6 Software

The software chapter contains descriptions of all the software that has been used during the work of the project.

The J-Link RTT Client V6.40 program was used to display the debug printouts from the Thingy:52 in a terminal window on a computer. The J-Link Commander v6.40 program was used to establish a connection between the computer and the nRF DK for making it possible for the nRF DK to send the received debug printouts from the Thingy:52 into the computer.

The Nordic Thingy app for android has been used for trying out the different features of the Thingy:52 such as monitoring sensor values, controlling the LED and playing sound from the speaker, etc.

The editor Atom was used for writing Firmware (FW) for the Thingy:52 and the GNU Tools ARM Embedded 4.9 2015q3 was used to compile FW for the Thingy:52. The code used for the Thingy:52 during the project is initially based on Thingy FW version 2.2.0 provided by Nordic semiconductor [17]. The nRF Connect v2.6.2 program was used on a computer for uploading code from the computer into the Thingy:52 via the nRF DK. The nRF Connect program was also used to verify the BLE communication of the GPS-tracker.

The Arduino integrated development environment (IDE) was used on the computer for writing and compiling code for the LoRa-module and for uploading code into the LoRa-module. The program KiCad was used for drawing the proposal circuit diagram for the GPS-tracker, the drawn circuit diagram can be seen in Figure 17. The NFC Tools app for android was used to writing information about triage assessments and rescue personnel id into NFC-tags.

The program ScriptCommunicator V5.09 was used on the computer to send UART commands to the GPS-module. The use of ScriptCommunicator made the processes of trying out different settings for the GPS-module faster than it would have been if the settings were tested by establishing the UART communication between the Thingy:52 and GPS-module right away.

Simulink was used for doing the power measurement on the different modules on the GPS-tracker, see Figure 14 for an illustration of the power measurement.

3 Method

The method chapter will include information of the literature studying of the different technologies and hardware. The chapter will also contain information about how the GPS-tracker FW was developed, how the different modules were implemented, how the power measurement was performed, how the functionality was verified, how the drawing of the proposal circuit diagram was performed and information about the possibility to adjust the system by using macros.

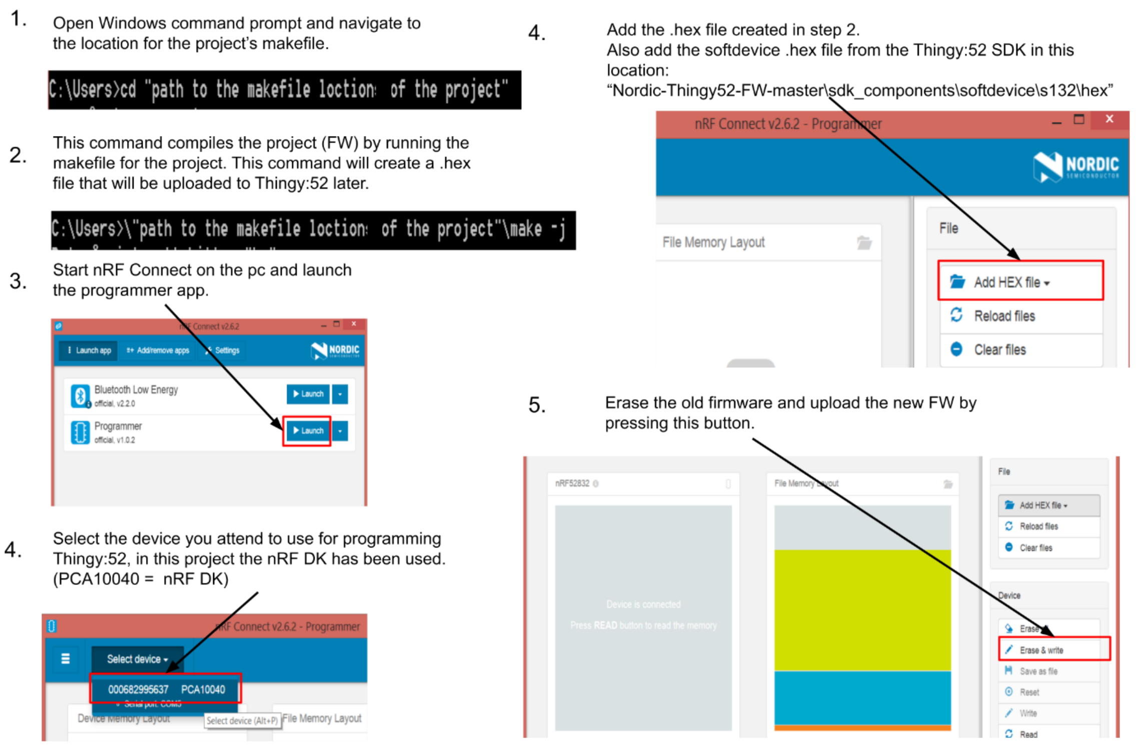
3.1 Preparation phase

The project started with studying the datasheets and other available documents for the Thingy:52 and a research about these different technologies: BLE, BLE mesh networks, NFC and LoRa.

The Nordic Thingy app for android from Nordic semiconductor was downloaded and the app was used for establishing a BLE connecting between the cellphone and the Thingy:52. After establishing a connection the app was used to get to explore the features of Thingy:52 by reading sensor values, controlling LED light, playing sound from the speaker through the app.

Necessary preparations for writing firmware for the Thingy:52 were performed:

- The software development kit (SDK) for Thingy:52 was downloaded, it is like a big library provided by Nordic Semiconductor which for instance consisted of an example program which has been used as the base for the Thingy:52 FW during this project.
- To make the compiling of FW possible by running a makefile, the Make program was installed on the computer, see an illustration of the compiling procedure in Figure 11.
- The compiler GNU Tools ARM Embedded 4.9 2015q3 was downloaded and installed on the computer for compiling FW for the Thingy:52.
- For loading compiled code from the computer into the Thingy:52 the nRF Connect v2.6.2 program was installed on a computer.


1. Open Windows command prompt and navigate to the location for the project's makefile.

```
C:\Users>cd "path to the makefile loction of the project"
```

2. This command compiles the project (FW) by running the makefile for the project. This command will create a .hex file that will be uploaded to Thingy:52 later.

```
C:\Users>\"path to the makefile loction of the project"\make -j
```

3. Start nRF Connect on the pc and launch the programmer app.

4. Select the device you attend to use for programming Thingy:52, in this project the nRF DK has been used. (PCA10040 = nRF DK)

4. Add the .hex file created in step 2. Also add the softdevice .hex file from the Thingy:52 SDK in this location: "Nordic-Thingy52-FW-master\sdk_components\softdevice\s132\hex"

5. Erase the old firmware and upload the new FW by pressing this button.

Figure 11 Illustration of all necessary steps for compiling and uploading firmware to Thingy:52.

It occurred several problems during the preparation phase that were very time consuming and more about those problems can be read under the discussion chapter.

3.2 Initially programming of Thingy:52

After the preparation phase some small adjustments were performed in the given example program and uploaded to Thingy:52 just to verify that the programming of Thingy:52 worked properly.

The following functionality in the FW for Thingy:52 were added: reaction after a certain number of steps taken, reaction by changes in orientations, reaction when a timer triggered. The step detections functionality was already implemented in the original FW but for having the Thingy to react on a certain number of steps a counter was added into the FW. The counter was increased each time a step was detected and when the counter reached a certain value the Thingy:52 could react on the detection. Step detection was not enabled automatically in the origin FW and therefore an automatically enabling of the step detection was also added to the Thingy:52 FW.

Change in orientation detection was also implemented in the original FW but the functionality for orientations detection was not enabled automatically and therefore an automatically enabling of the orientation detection was added into the initiation part of the Thingy:52 FW. For the timer triggering a timer was created to trigger periodically.

3.3 Long range and Bluetooth low energy implementation

A device was registered on TTN website and the necessary keys for OTAA for LoRa communication i.e. device EUI, application EUI and app key were received from TTN website.

The format of the data package that the LoRa communication and BLE communication uses was defined, the format of the data package is illustrated in Figure 15.

The necessary modification in the Thingy:52 FW for sending data packages over BLE when some sensor triggers the data transfer were made, see an illustration of the BLE communication implementation in Figure 12. The modifications were done by first creating copy of the already existing BLE service for orientations detection in the Thingy:52 FW. When the copying was performed some additional modification had to be done such as:

- Assigning a new universally unique identifier (UUID) to the new BLE service.
- Add read property to the service characteristic to make it possible to send more than 20 bytes data at a time since the orientation service only got the notification property which only allow a maximum of 20 bytes to be transmitted at a time and the data package that the GPS-tracker send exceeds 20 bytes.
- Changed the of the provided value in the service characteristic from orientation data to the define data package for the GPS-tracker.
- At first the Thingy:52 was not able to run the FW with the newly added BLE service but it was solved by changing the location where the generic attribute profile (GATT) attributes for the new BLE service was stored from BLE_GATTS_VLOC_STACK to BLE_GATTS_VLOC_USER. The BLE_GATTS_VLOC_STACK location ran probably out of memory when the new service was added.

The initial idea of the BLE implementation was to implement BLE mesh functionality but due to time limitation that idea was set as a future work instead and more information of what effects a BLE mesh can have is described under the future work chapter.

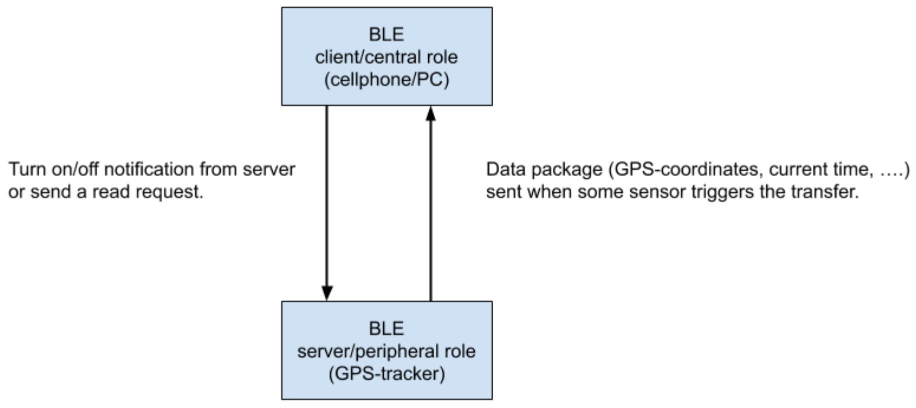

BLE
client/central role
(cellphone/PC)
Turn on/off notification from server
or send a read request.
Data package (GPS-coordinates, current time, ….)
sent when some sensor triggers the transfer.
BLE
server/peripheral role
(GPS-tracker)

Figure 12 Illustration of how the BLE implementation works for the GPS-tracker where a cellphone/PC has a central role and the GPS-tracker has a peripheral role.

3.4 GPS-module implementation

The GPS-module and a computer were connected with a USB to UART cable called TTL-235R-3V3 between each other. Since the GPS-module uses UART to communicate the ScriptCommunicator V5.09 program was used on the computer to send UART commands to the GPS-module for testing the different commands on the GPS-module before implementing the GPS-module together with Thingy:52.

After finishing the testing of the commands on the GPS-module, the GPS-module was implemented together with the Thingy:52. Based on the previous testing of commands some commands were chosen to be used on the GPS-module together with the Thingy:52. The PMTK314 command was used for excluding all information sent from the GPS-module except the information about longitude, latitude, and current time. The PMTK220 command was used to change the positioning fix interval from one second (default) to ten second for optimizing the power consumption.

3.5 Long range-module implementation

For the implementation of the LoRa-module, Arduino IDE was installed on the computer to be able to program the LoRa-module since it is an Arduino compatible board. For handling the LoRa communication the LoraMAC-in-C (LMIC) library was downloaded. In the LMIC library existed an example program for LoRa communication with OTAA and that program was used as the starting point for the LoRa-module program.

The modifications that needed to be made in the provided LMIC example was to add device EUI, application EUI and app key from the registered device on the TTN website and the setup of a I2C communication to be able to receive data packages from the Thingy:52. Each time a data package is received from the Thingy:52, the LoRa-module sends that data package further to a LoRa-gateway that can send the package up to the internet.

3.6 Near field communication-reader implementation

For the implementation of the NFC-reader it already existed an example program for the PN532 NFC-reader, but the program was written for the nRF DK. For making it possible to use the NFC-reader together with the Thingy:52 the pin mapping had to be changed and some files and paths had to be added in the makefile of the project. Some definitions were also added into the projects sdk_config.h file (this file consists of many different macros that can be used for enabling debug messages, UART, etc. for the Thingy:52).

After implementing the NFC-reader unused code was removed and the existing reading function was modified to just read the 16 bytes where the triage assessment and rescue personnel id were stored instead of reading all bytes of the NFC-tag to optimizing the power consumption. The original example for the NFC-reader was continuously checking the presence for NFC-tags, to save energy the code for the NFC-reader was modified to just check presence for NFC-tags every twice second. Some code for sending a data packed with BLE and LoRa when a NFC-reading occurs were also added.

The NFC-reader implementation in this project can only read NFC-tags of type 2. The NFC-reader functionally was tested by reading the payload from four NFC-tags of type 2 that contained records with different payloads of the type text that contained IDs (identifications for e.g. fireman or paramedic staff) and triage assessments. The record was written to the NFC-tags by using the NFC Tools app for android. To indicate that a NFC-tag has been successfully read the Thingy:52 plays a tone from the speaker and emits green light from the LED.

3.7 Sound detection

Thingy:52 has a built-in microphone and for the sound detection implementation a timer was set to trigger once in a minute and when it times out the microphone is turned on for two seconds. When the microphone is turned on the Thingy:52 is continuously checking if value of the sound level is over a specific threshold and if it exceeds the threshold the Thingy:52 can react and transfer a data package with BLE and LoRa. If the threshold is not exceeded the microphone is turned off and the microphone will be turned on again after one minute.

3.8 Power consumption measurement

The power measurement on the GPS-tracker was performed by using Simulink with an Arduino plugin and an Arduino UNO. The Arduino UNO measured the voltage over resistors that were connected in series with the different modules as shown in Figure 24.

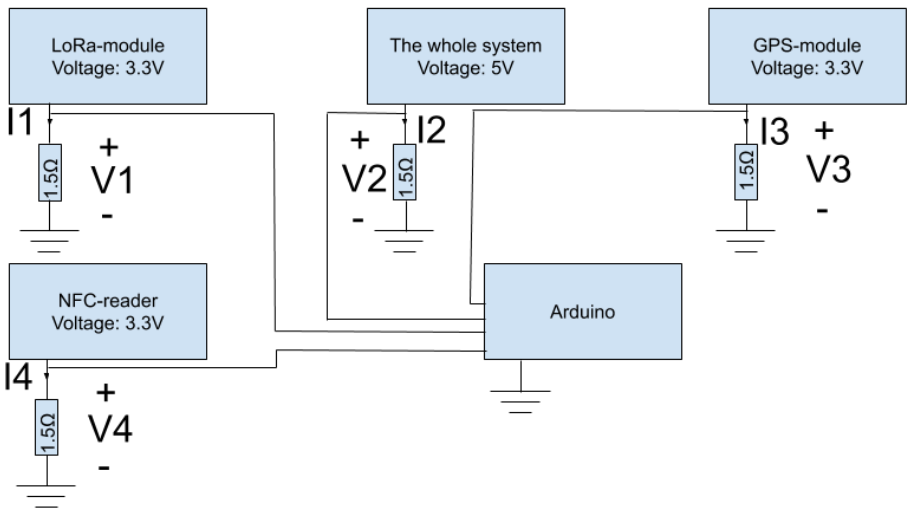


Figure 13 Illustration of how an Arduino performing voltage measurements over resistors that are connected in series with the different modules of the GPS-tracker.

In Simulink the power consumption was calculated by doing the following calculation for the different modules:

1. $\frac{\text{Voltage over resistor}}{1.5\Omega} = Current\ for\ the\ module$
2. $Current\ for\ the\ module * Voltage\ for\ the\ module = Power\ consumtion\ for\ the\ model$

How the program was setup in Simulink is illustrated in Figure 14.

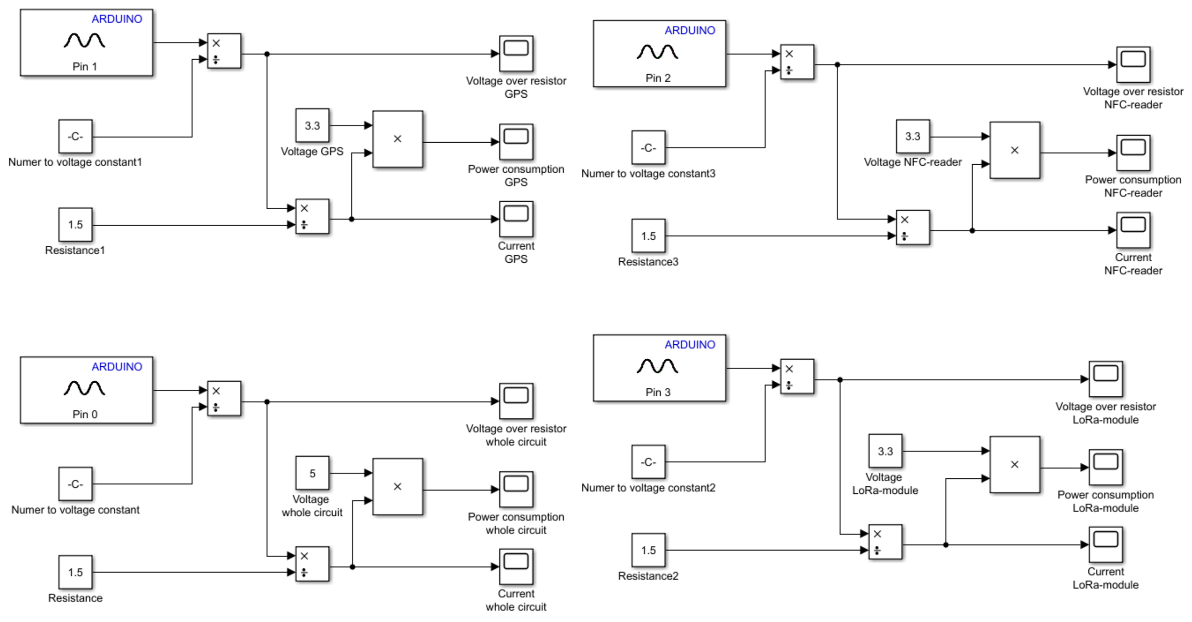

ARDUINO
Pin 1
Numer to voltage constant1
-C-
1.5
Resistance1
3.3
Voltage GPS
Voltage over resistor GPS
Power consumption GPS
Current GPS
ARDUINO
Pin 2
Numer to voltage constant3
Resistance3
Voltage NFC-reader
Voltage over resistor NFC-reader
Power consumption NFC-reader
Current NFC-reader
ARDUINO
Pin 0
Numer to voltage constant
Resistance
5
Voltage whole circuit
Voltage over resistor whole circuit
Power consumption whole circuit
Current whole circuit
ARDUINO
Pin 3
Numer to voltage constant2
Resistance2
Voltage LoRa-module
Voltage over resistor LoRa-module
Power consumption LoRa-module
Current LoRa-module

Figure 14 Illustration of how Simulink calculate the power consumption for the different modules of the GPS-tracker.

3.9 Functionality test

To verify the functionality of the whole system, the GPS-tracker was carried around in the area of Linköping university and different sensors were triggered and it was controlled if data packages with correct data was sent up to the TTN. Some of the received data packages from the test run is illustrated in Figure 18.

3.10 Adjustment macros

When the GPS-trackers functionalities worked as it should some macros for adjustments purpose were added into the Thingy:52 FW.

Adjustments macros located in the main.c file in Thingy:52 FW and can be seen in Appendix 1:

TIMER_INTERVAL_VALUE_MS: defines how often the GPS-tracker should send data packages via LoRa and BLE depending on a timer interval set by this macro.

TIME_OUT_NFC_SCAN_MS: defines how long time after a NFC-scan the NFC-reader should be blocked for new NFC-scans.

NFC_SCAN_INTERVAL_MS: defines how often the NFC-reader should check for the present of NFC-tags.

SOUND_STARTING_INTERVAL_MS: defines how often the GPS-tracker should turn on the microphone and listen if a specific threshold is exceeded.

SOUND_LISTNING_TIME_MS: defines how long time the microphone should be on when it is checking if the sound level threshold is exceeded.

Adjustments macros located in the sdk_config.c file in Thingy:52 FW and can be seen in Appendix 2:

SOUND_THRESHOLD: defines the threshold for sound detection.

NUMBER_OF_STEP_DETECTION: defines how many steps it needs to be taken before the GPS-tracker sends a data package through LoRa and BLE.

UART_ON_OFF: This macro can be used for disabling/enabling the UART communication with the GPS-module.

TAG_ON_OFF: This macro can be used for disabling/enabling the I2C communication with the NFC-reader.

LORA_I2C_ON_OFF: This macro can be used for disabling/enabling the I2C communication with the LoRa-module.

3.11 Circuit Diagram for GPS-tracker

For the drawing of a proposal circuit diagram for the GPS-tracker the program KiCad was used. The manufactures of the coherent hardware to the GPS-module, LoRa-module and the NFC-reader had available circuit diagrams that were drawn in the circuit design program Eagle. A new circuit diagram was created in KiCad and the circuit diagrams for the GPS-module, the LoRa-module and the NFC-reader were imported into the created circuit diagram in KiCad. The pins strip on the Thingy:52 where all the wires from the other modules are connected to was also added into the circuit diagram. After all components were placed in the circuit diagram all wires between the components were drawn out. The proposal for the circuit diagram can be seen in Figure 17.

4 Results

The result chapter will present the different features of the constructed GPS-tracker, format of the used data package, energy consumption and a proposal circuit diagram for the GPS-tracker.

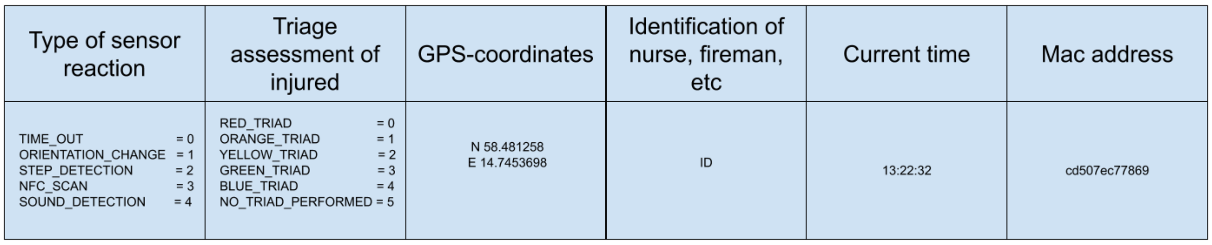
4.1 GPS-tracker construction and data format

The project has resulted in a GPS-tracker device with the following features:

- LoRa communication that can transfer data packages up to the internet see Figure 18 for an illustration of how the data from the LoRa-module looks like when it is received by TTN.
- Point to point BLE communication which can transfer data packages to Nordic Semiconductors BLE apps, see Figure 19 for an illustration of how a received data package looks in the BLE service on Nordic Semiconductors nRF connect BLE app.
- Reading triage assessments and ID-numbers from NFC-tags of type 2.
- Data package transfer via LoRa and BLE is triggered when the follow things are detected by the GPS-tracker: x number of steps have been taken, the device has been rotated 90°, a certain sound level is exceeded, timer triggering in a defined time interval and when a NFC-reading occurs.

The format of the data package being sent through LoRa and BLE is illustrated in Figure 15 and a block diagram over the GPS-tracker can be seen in Figure 16.

A proposal circuit diagram for the GPS-tracker can be seen in Figure 17



Type of sensor reaction	Triage assessment of injured	GPS-coordinates	Identification of nurse, fireman, etc	Current time	Mac address
TIME_OUT = 0 ORIENTATION_CHANGE = 1 STEP_DETECTION = 2 NFC_SCAN = 3 SOUND_DETECTION = 4	RED_TRIAD = 0 ORANGE_TRIAD = 1 YELLOW_TRIAD = 2 GREEN_TRIAD = 3 BLUE_TRIAD = 4 NO_TRIAD_PERFORMED = 5	N 58.481258 E 14.7453698	ID	13:22:32	cd507ec77869

Figure 15 Format of the data package being sent over long range and Bluetooth low energy when a sensor triggers the data transfer.

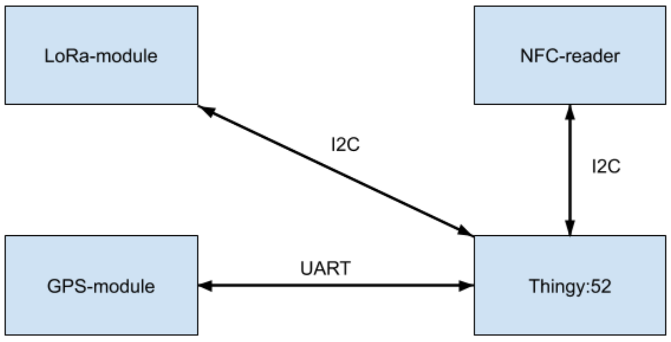


Figure 16 Block diagram of how the GPS-tracker is connected to the long range-module and the NFC-reader with an I2C connection and with the GPS-module with a UART connection.

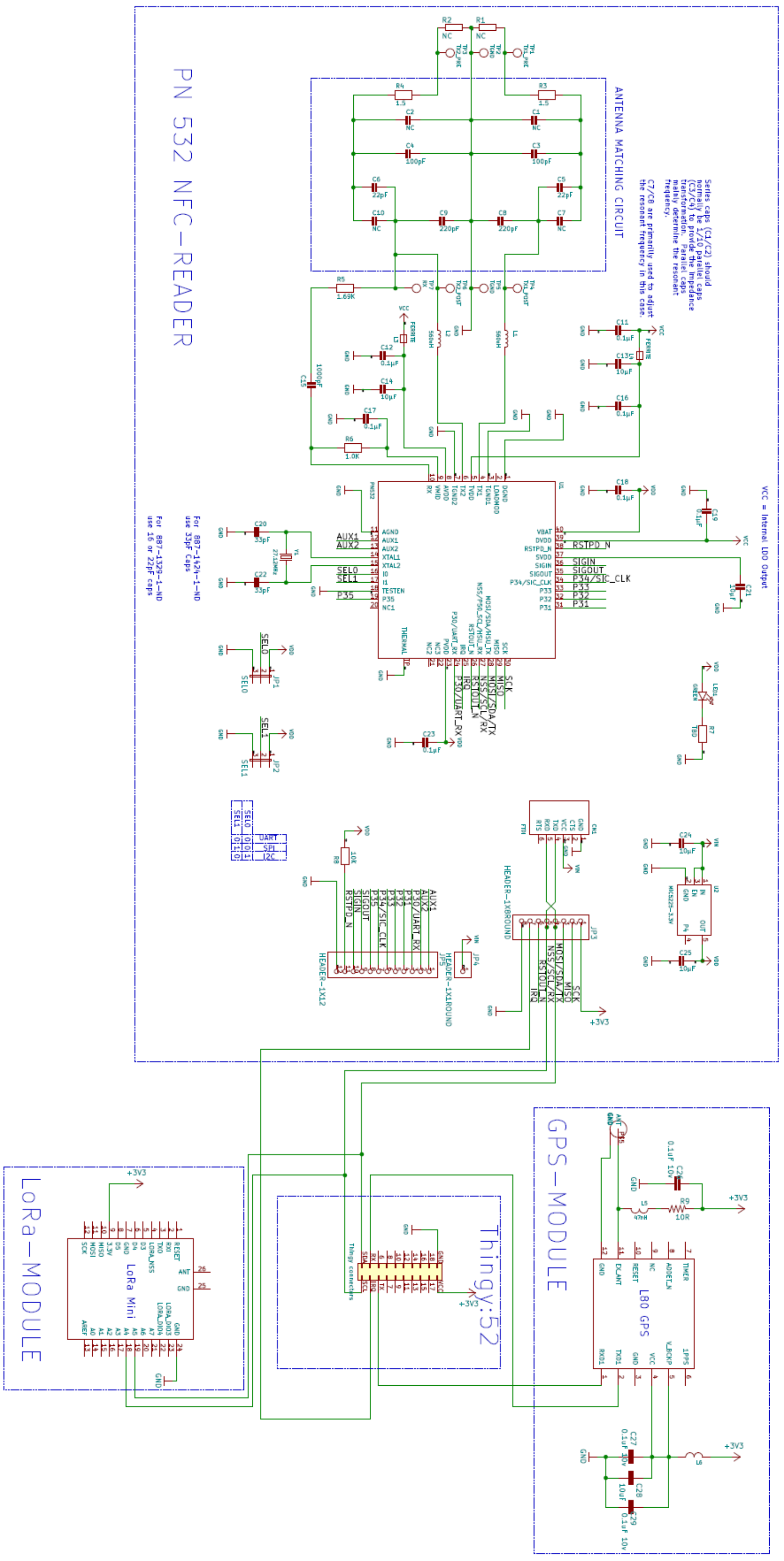
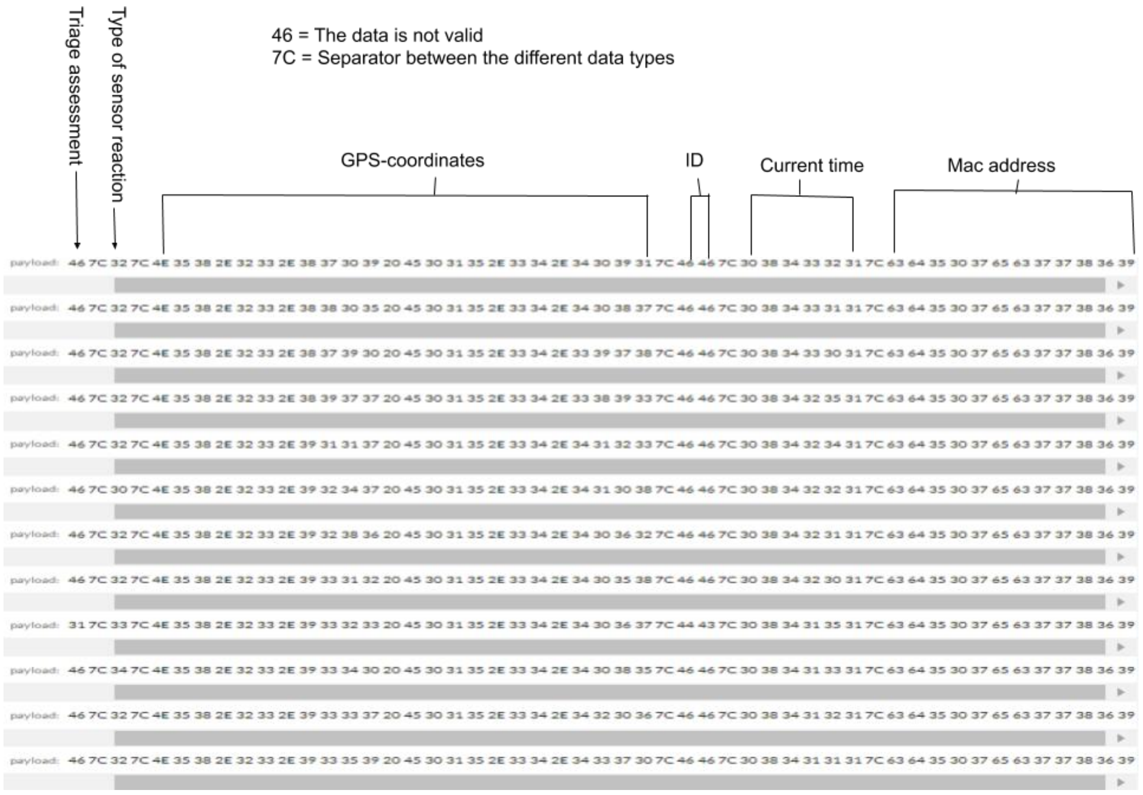
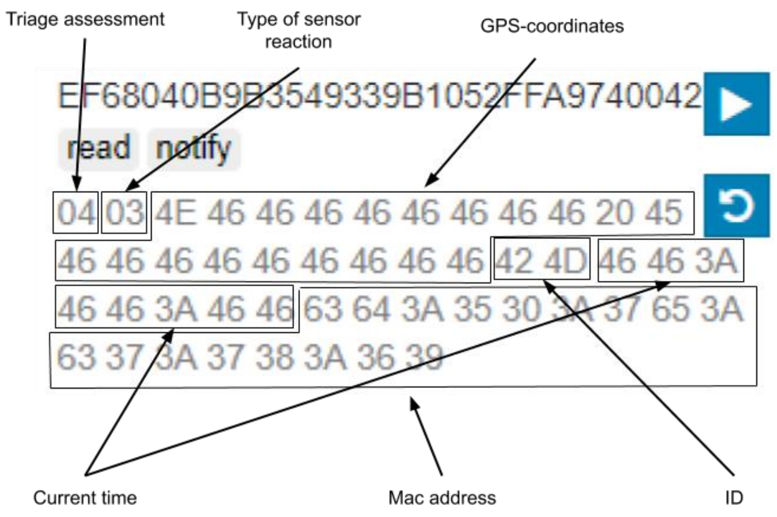


Figure 17 Circuit diagram proposal for the GPS-tracker.



```
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 38 37 30 39 20 45 30 31 35 2E 33 34 2E 34 30 39 31 7C 46 46 7C 30 38 34 33 32 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 38 38 30 35 20 45 30 31 35 2E 33 34 2E 34 30 38 37 7C 46 46 7C 30 38 34 33 31 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 38 37 39 30 20 45 30 31 35 2E 33 34 2E 33 39 37 38 7C 46 46 7C 30 38 34 33 30 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 38 39 37 37 20 45 30 31 35 2E 33 34 2E 33 38 39 33 7C 46 46 7C 30 38 34 32 35 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 39 31 31 37 20 45 30 31 35 2E 33 34 2E 34 31 32 33 7C 46 46 7C 30 38 34 32 34 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 30 7C 4E 35 38 2E 32 33 2E 39 32 34 37 20 45 30 31 35 2E 33 34 2E 34 31 30 38 7C 46 46 7C 30 38 34 32 32 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 39 32 38 36 20 45 30 31 35 2E 33 34 2E 34 30 36 32 7C 46 46 7C 30 38 34 32 31 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 39 33 31 32 20 45 30 31 35 2E 33 34 2E 34 30 35 38 7C 46 46 7C 30 38 34 32 30 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 31 7C 33 7C 4E 35 38 2E 32 33 2E 39 33 32 33 20 45 30 31 35 2E 33 34 2E 34 30 36 37 7C 44 43 7C 30 38 34 31 35 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 34 7C 4E 35 38 2E 32 33 2E 39 33 34 30 20 45 30 31 35 2E 33 34 2E 34 30 38 35 7C 46 46 7C 30 38 34 31 33 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 39 33 33 37 20 45 30 31 35 2E 33 34 2E 34 32 30 36 7C 46 46 7C 30 38 34 31 32 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
payload: 46 7C 32 7C 4E 35 38 2E 32 33 2E 39 33 35 39 20 45 30 31 35 2E 33 34 2E 34 33 37 30 7C 46 46 7C 30 38 34 31 31 31 7C 63 64 35 30 37 65 63 37 37 38 36 39
```

Figure 18 Illustration of how messages looks like when it is received by The Things Network from the long range-module.



```
EF68040B9B3549339B1052FFA9740042
04 03 4E 46 46 46 46 46 46 46 46 20 45
46 46 46 46 46 46 46 46 46 42 4D 46 46 3A
46 46 3A 46 46 63 64 3A 35 30 3A 37 65 3A
63 37 3A 37 38 3A 36 39
```

Figure 19 Illustration of how the data package received in the Bluetooth low energy service looks like.

4.2 Power consumption

The power consumption chapter will present the power consumption of the different modules and the distribution of the power consumption between the modules.

The power consumption for the different modules and the whole GPS-tracker is shown in Figure 20, Figure 21, Figure 22 and Figure 23.

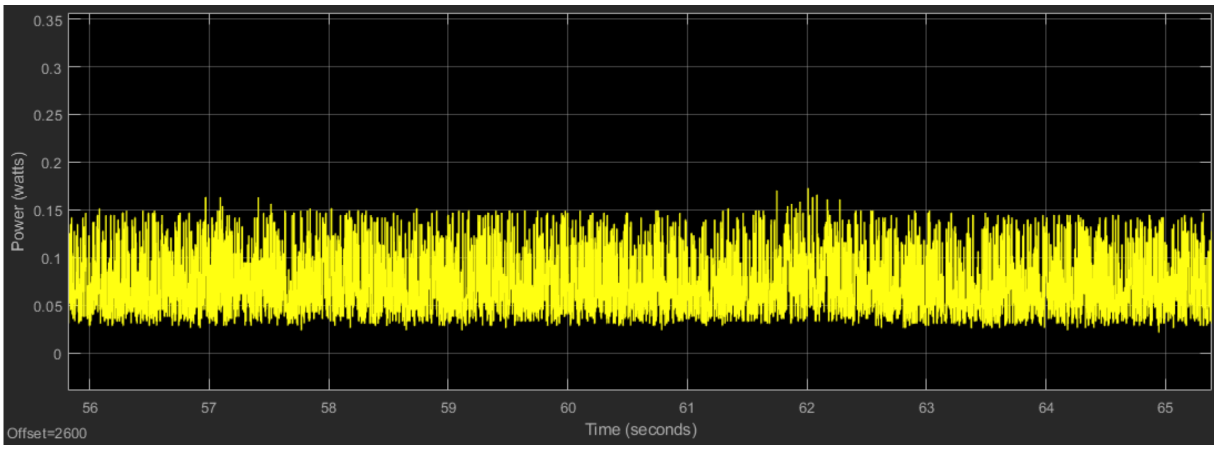


Figure 20 Graph over the power consumption for the long range-module.

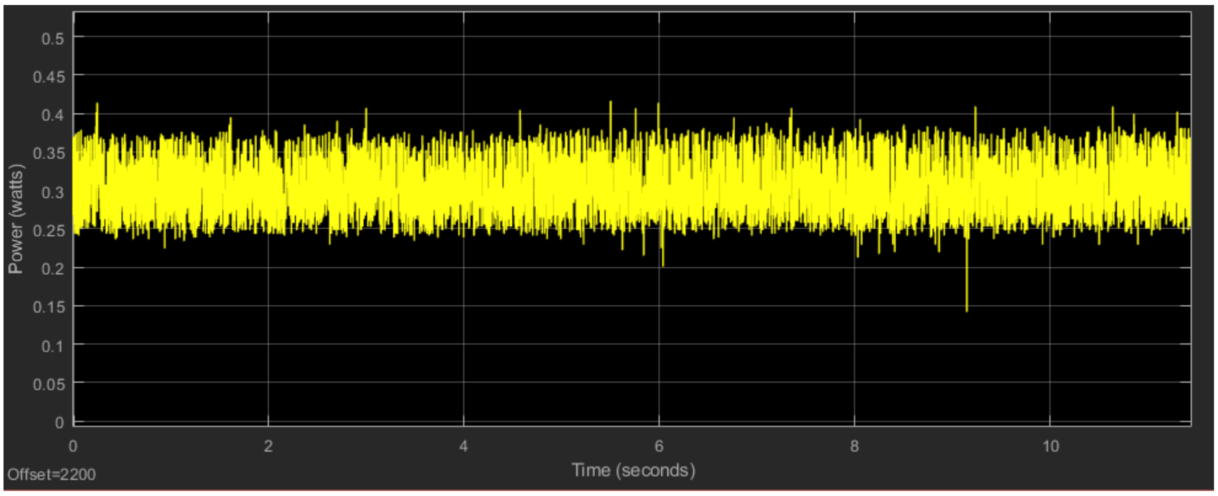


Figure 21 Graph over the power consumption for the NFC-reader.

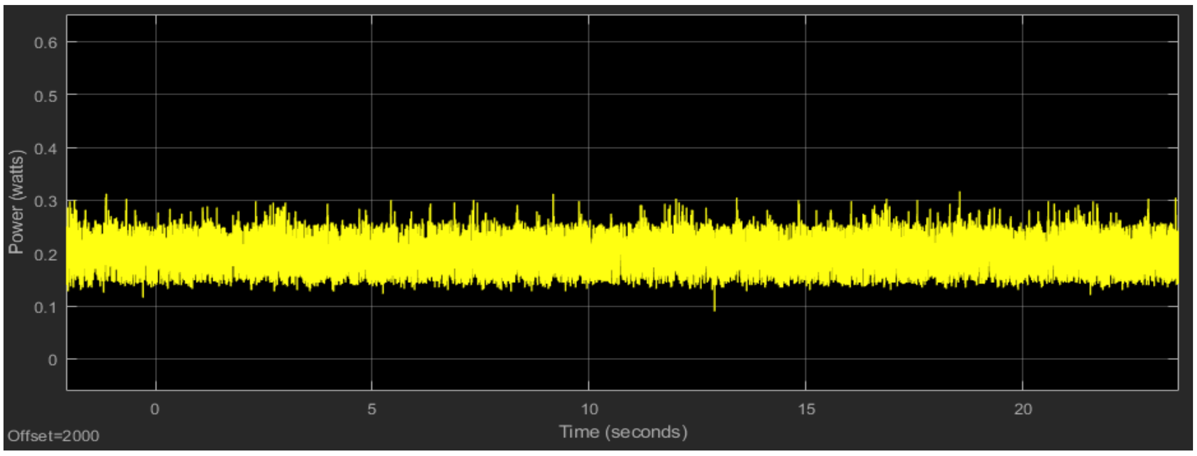


Figure 22 Graph over the power consumption for the GPS-module.

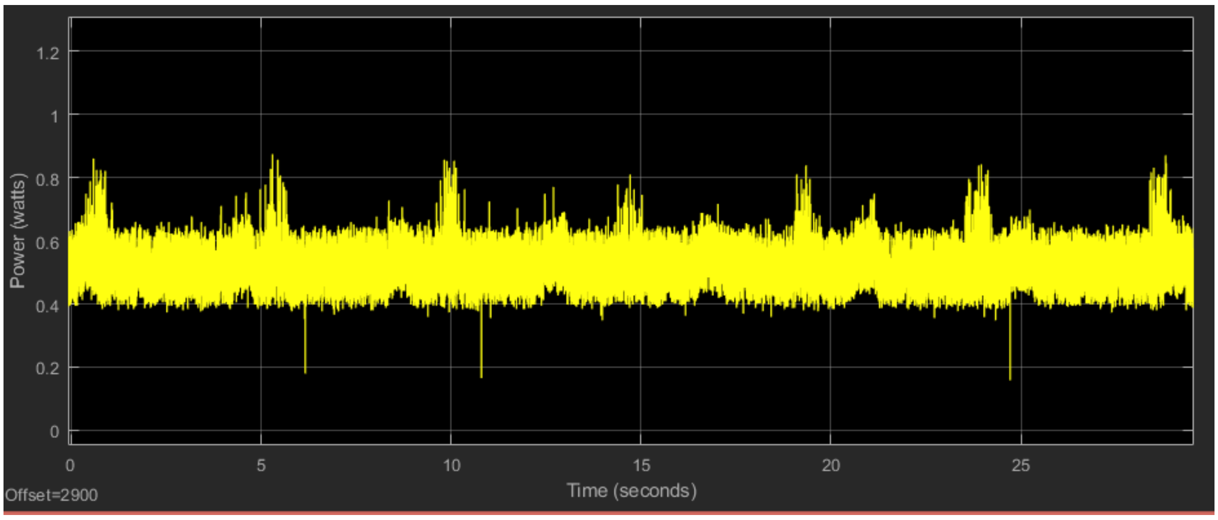


Figure 23 Graph over the power consumption for the whole GPS-tracker device.

The power consumption of the Thingy:52 could not be measured directly since the other modules gets their power supply from the Thingy:52. To calculate the power consumption for the Thingy:52 the following calculation was performed:

$$Whole\ GPS\ device\ power - LoRa\ module\ power - GPS\ module\ power - NFC\ reader\ power = Thingy{:}\,52\ power$$

An illustration of the power consumption distribution between the modules can be seen in Figure 24.

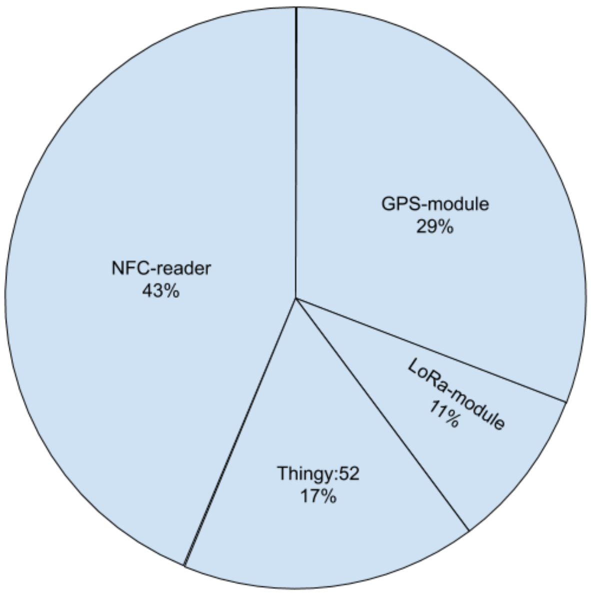


Figure 24 Illustration of how the power consumption is distribution between the different modules of the GPS-tracker.

5 Discussion

The discussion chapter includes discussions about the results of the project, future work for the GPS-tracker, methods used during the project, ethical aspects and a discussion about how a rescue operation in a future disaster can look like if GPS-trackers like the one constructed in this project are used on disaster sites.

5.1 Results

A GPS-tracker with LoRa, BLE capability has successfully been built. As the Figure 24 illustrate is the biggest part of the power consumed by the NFC-reader and the GPS-module. Therefore, it is probably a good idea to start looking for power optimizations for the NFC-reader and the GPS-module if attempting to optimize the power consumption of the GPS-tracker in the future.

A proposal for power optimization of the GPS-module and NFC-reader is described in the chapter future work.

The initial idea for the BLE communication was to add BLE mesh functionality to the GPS-tracker also but due to the time limitations it was decided to aim for creating a point to point BLE communication instead which has been accomplished.

5.2 Method

The method chapter will discuss issues and solutions for issues during the work of the project and source criticism will also be discussed.

5.2.1 Preparation phase and initially programming of Thingy:52

During the preparation of the project it was quite a struggle to get FW to compile and upload to the Thingy:52. The struggling with compiling FW to Thingy:52 depended on that it was three ways to compile the code and only one worked. The three ways for compiling FW for Thingy:52 are:

1. Compiling FW with the Kiel program did not work because there was a limitation of how big the FW could be in the free version of Kiel.
2. Compiling FW with SEGGER studio, the FW made for SEGGER studio that was provided by Nordic Semiconductor did not work and it seems like they by mistake released a broken FW for SEGGER studio, at least according to information from Nordic Semiconductor forum Devzone [18].
3. Compiling FW with a GNU compiler and makefile was finally the way for compiling that worked but initially that did not work either. The reason was because git bash terminal did not work to compiling the project with, but the project complied successfully when the standard Windows terminal was used instead.

Problems with uploading code into Thingy:52 dependent on a cable with the wrong pinout was used. The cables were very similar and therefore it took a while to realize that it was the cable that caused the problem, see Figure 25 for a comparison between the cables.

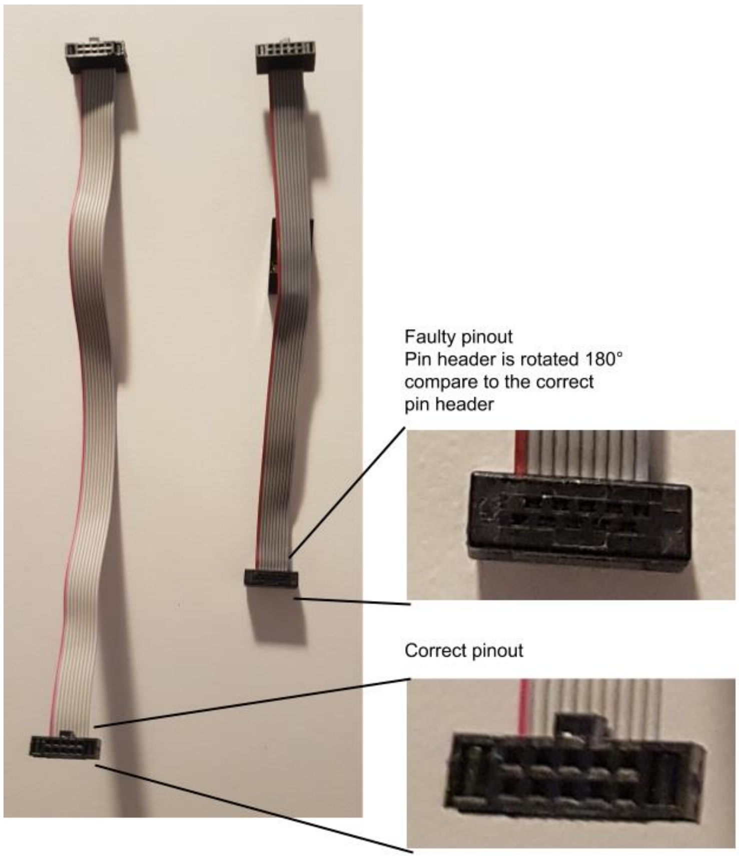


Figure 25 Comparison image between a correct and faulty programming cable for Thingy:52.

5.2.2 Near field communication-reader implementation

Initially it was assumed that Thingy:52 had capability to read passive NFC-tags but after asking Nordic Semiconductor forum Devzone an engineer from Nordic Semiconductor replied and informed that Thingy:52 cannot read other passive NFC-tags. This new insight lead to an external NFC-reader had to be ordered.

5.2.3 Time optimization

In the middle of the project it was realized to try some new changes in the code on the Thingy:52 was quite time consuming since the process for compiling and uploading code to Thingy:52 looks like this: do the five compiling steps illustrated in Figure 11 -> start the programs J-Link client and J-Link commander to be able to see the debug printouts. Often it was some functionality that did not need Thingy:52 hardware to be tested. A faster way to try it was to write the functionality in a separate program and just compile it with a GNU compiler and run the program in the windows terminal.

5.2.4 Long range

The reason for choosing LoRa as communication for this project was because the LoRa technique can transfer data long distance with a low energy consumption. It is important to have a low energy consumption since it will extend the battery time and minimizing the risk of having the GPS-trackers discharged during a rescue operation.

On the TTN website there is a map over LoRa-gateways in the world and initially it was assumed if the LoRa-module is nearby a gateway on that map the LoRa-module will be able to send data to it. That was not true all the times and the reason for that could possibly be that all the gateways on that map might not be turned on around the clock.

During the project a colleague at Linköping university helped to set up a LoRa-gateway at Linköping university and after that the LoRa-module always had a gateway to test the functionality against.

5.2.5 Power consumption measurement

The power measurement was performed by having an Arduino measuring the voltage over 1.5 Ω resistors that were connected in series with the different modules as illustrated in Figure 13. The voltage measurements from the Arduino were then used by Simulink and calculate the power consumption for the different modules.

The initial idea was to use a very small resistor around 0.02 Ω to make sure the resistor would affect the power consumption measurement as little as possible but the voltage over the resistor would have been in size order that the Arduino would only had been able to show a few different voltage levels then. During the voltage measurement the Arduinos internal voltage level on 1.1 V was used as voltage reference and the resolution for voltage measurements in the Arduino is ten-bits (1024 different levels can be represented). The step size between the voltage level in the Arduino measurement is $\frac{1.1}{1024} \approx 0.001\ V$. The GPS-tracker was power by an external power source that showed that the power source consumed around 0.1 A which would have given a voltage over the resistor around $R * I = U = 0.02 * 0.1\ V = 0.002\ V$and since the step size is 0.001 V for the Arduino, the voltage measurement would not have given enough voltage levels to get a reliable calculation for the power consumption. Therefor a resistor of 1.5 Ω was chosen instead which gave a voltage drop around $1.5 * 0.1\ V = 0.15\ V$.

In Figure 20, Figure 23, Figure 21 and Figure 22 it is possible to see that the power consumption graphs are not smooth clean graphs and an average function could have made the graph looked more smooth and clean. But the purpose with the measurement was to give an approximate picture over how the power consumption is distributed between the different modules and for that purpose the measurement is good enough.

5.2.6 Source criticism

The sources used in the project have mostly been from published articles, published conference proceedings and websites from technology companies.

Sources taken from websites of technology companies were used with caution since there is a risk that the companies exaggerating the performance of their own technology. The sources taken from companies have therefore tried to be kept as sources of information for how the technology works and not for how well technology performs.

5.3 Ethical aspects

A GPS-tracker that sends information about condition and position of an injured person to the internet as the GPS-tracker in this report can be useful on disaster sites for making it easier and more efficient to organize the paramedics and fireman at the site. An operation leader could monitor the status from all GPS-trackers on the site and with the information the operation leader can organize the rescue operation based on the injured persons conditions and which of the rescue personnel that are closest to the injured people.

By utilizing the provided information about positions and conditions from a GPS-tracker on a disaster site could contribute to that the most injured people get help faster and the average time to get help could potentially be reduced. All data sent from the GPS-tracker to the internet will be saved and the data can be studied afterwards. By studying the data from a disaster scenario can possibly contribute to that injured people gets helps faster since it makes it possible to investigate what went wrong to avoid it in the future and see what went well and keep doing it in the future.

It is also important that information from the GPS-trackers do not gets in the wrong hands since false GPS coordinates can be sent by unauthorized data which exposes the system for potential danger.

A GPS-tracker that sends information via LoRa will probably be more troublesome to use for people that are planning to use the GPS-tracker for illegal surveillance compared with GPS-trackers that is sending information via the GSM net as many GPS-trackers on the market do currently. This is because the GPS-tracker with LoRa communication requires to be within a radius of 5 – 15 km from a LoRa-gateway to be able to send information to the internet [6].

5.4 Technology at a future disaster site

If paramedics, fireman and injured people carry GPS-trackers like the one built in this project a possible scenario when a disaster has occurred e.g. earthquake or avalanche that has buried people in snow is illustrated in Figure 26. As illustrated in Figure 26 a firetruck has a LoRa-gateway placed on it for giving the disaster site LoRa coverage to make it possible for the GPS-trackers to send information via LoRa. In Figure 26 there is also an operational leader that is monitoring the whole situation from a computer where the operation leader can see all GPS-position of people on site that carries a GPS-tracker and the injured peoples triage assessment and can based on that information give order to the rescue personnel on where they should go next.

Figure 26 illustrates that the paramedics have got orders from the operational leader to take care of the people that have the most urgent need for help. Some of the firemen have got orders to help the green marked injured people which do not have such urgent need for help as the orange and red marked injured people. The fireman F3 has got order to search for other injured people on the site.

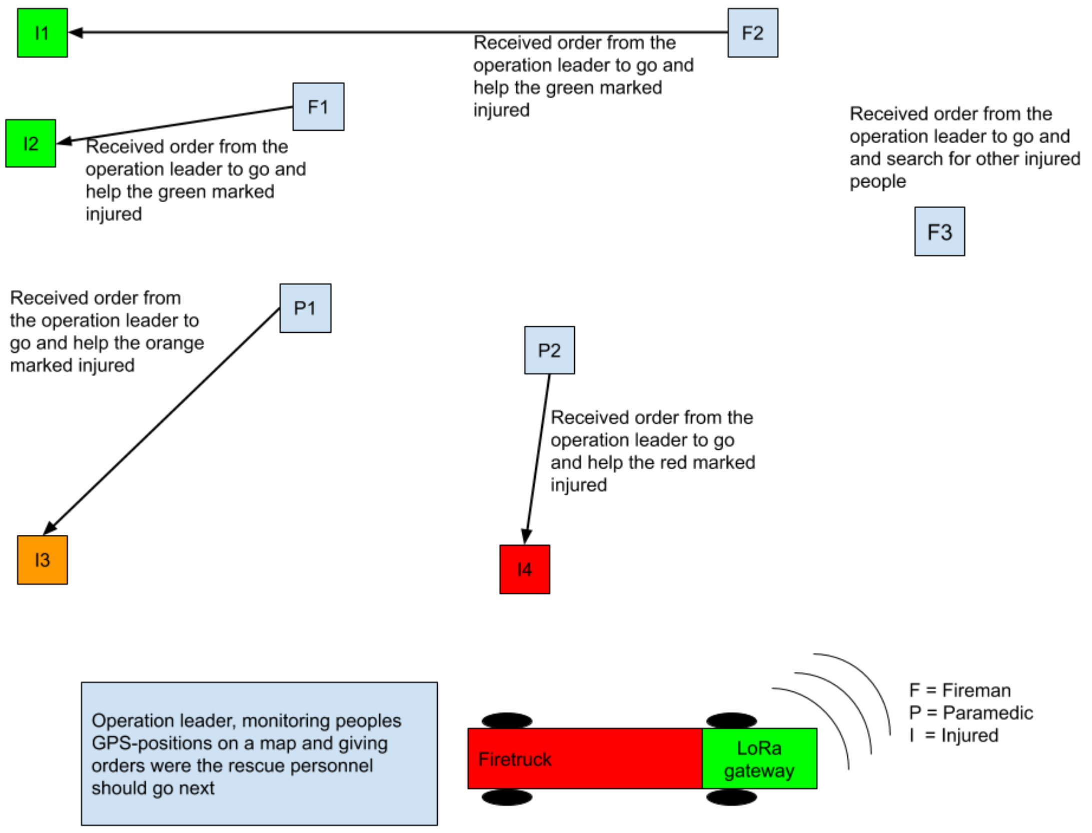

I1
F2
Received order from the operation leader to go and help the green marked injured
F1
I2
Received order from the operation leader to go and help the green marked injured
Received order from the operation leader to go and and search for other injured people
F3
Received order from the operation leader to go and help the orange marked injured
P1
P2
Received order from the operation leader to go and help the red marked injured
I3
I4
Operation leader, monitoring peoples GPS-positions on a map and giving orders were the rescue personnel should go next
Firetruck
LoRa gateway
F = Fireman
P = Paramedic
I = Injured

Figure 26 Illustration over a possible future situation on a disaster site with injured people and an operation leader giving orders to the rescue personnel and a firetruck with a long range-gateway placed on it.

5.5 Future work

The future work chapter will discuss future work for the GPS-tracker.

5.5.1 Energy optimization

Figure 24 illustrate that the NFC-reader stands for 43% of the GPS-trackers power consumption. The current implementation of the NFC-reader is checking for nearing NFC-tags every twice second when the Thingy:52 tells the NFC-reader to do that. Since the GPS-tracker would not be receiving NFC-scans several times in a minute, a more realist NFC-scan frequency is probably around a couple of scans per hour. A way of reducing the energy consumption of the NFC-reader could be to have the power supply to the NFC-reader to be normally switch off and adding a resilient switch to the GPS-tracker. By pressing a NFC-tag against the button it could generates an interrupt signal to the Thingy:52 and in the interrupt handler could then turn on the power supply for the NFC-reader and perform a reading of the NFC-tag and then turn off the power supply until the bottom is pressed again. This solution will also reduce the power consumption for the Thingy:52 since it would not be needed to command the NFC-reader to check after nearby tags every twice second.

The GPS-module stand for 29% of the total power consumption which is a big part of the total consumption. A way to reduce the power consumption of the GPS-module could be to add a dead reckoning device (uses accelerometer and gyroscope to navigate instead of using satellites as a GPS) to complement the GPS-module with. An alternative to a dead reckoning device could be the BHI160BP from Bosch which according to Bosch them self says that it could reduce the power consumption up to 80% [19]. If a dead reckoning device is added to the system it does not only gives the opportunity to minimize power consumption, it will also make positioning possible when there is no GPS-coverage since it only uses accelerometer and gyroscope for navigation.

5.5.2 Bluetooth low energy mesh

An implementation of a BLE mesh network to the system will increase the connectivity between device which could be good in a situation like this: a paramedic has not received any orders from the operation leader and cannot find any injured people, a nearby injured GPS-tracker broadcast the position of itself and the paramedic staff then receives the position and can go and help the injured.

Additional hardware will not be necessary if BLE mesh functionality is implemented to the system since the nRF52832 chip on the Thnigy:52 already has the capability to use BLE mesh.

A BLE mesh could also help a LoRa message to be transmitted to a LoRa-gateway from a GPS-tracker that do not has LoRa coverage by passing the message through other nodes in the mesh. See Figure 27 for an illustration of this situation.

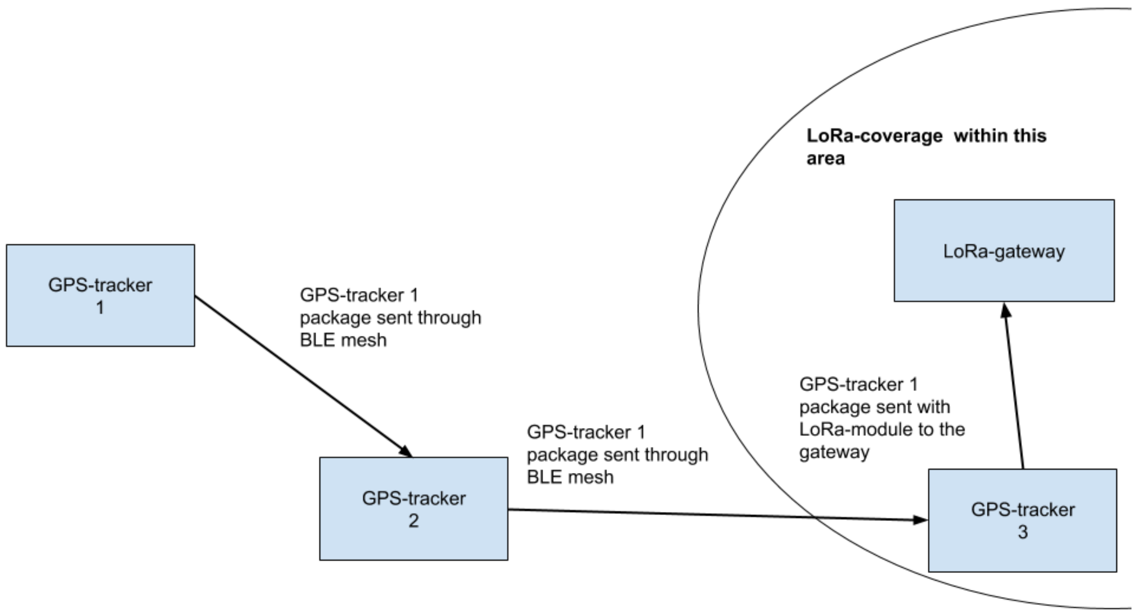

LoRa-coverage within this area
LoRa-gateway
GPS-tracker 1
GPS-tracker 1 package sent through BLE mesh
GPS-tracker 1 package sent with LoRa-module to the gateway
GPS-tracker 1 package sent through BLE mesh
GPS-tracker 2
GPS-tracker 3

Figure 27 Illustration of how the long range communication range is extended by using Bluetooth low energy mesh to transfer message through other devices in the network.

5.5.3 Miniaturizing

A future work on the GPS-tracker is to miniaturize the current hardware layout. Something that will save time during the miniaturizing work is that the footprints for the coherent hardware to the LoRa-module, NFC-reader and the GPS-module is provided by the manufactures in the circuit design program Eagle.

6 Conclusions

A GPS-tracker with BLE communication, LoRa communications, and NFC-reading capability has been built during this project. This device needs to be miniaturized before a realistic test on a disaster site is performed for avoiding that the size of the device being an obstacle when moving around with it attached to the body.

How much of a help the GPS-tracker will have on a disaster site can only be speculated in right now. The GPS-tracker will have to be put to a realistic test in some kind of disaster situation before it is possible to say what effects the GPS-tracker has on a rescue operation. In the purpose chapter it is described what the project wants to contribute with on a rescue operation on a disaster site.

The largest power consumption on the GPS-tracker comes from the GPS-module and the NFC-read. Power optimization proposals are presented in the future work chapter.

In the chapter technology at a future disaster site is it presented how the work at a future disaster site can look like if a GPS-tracker like the one built in this project is used in the rescuing operation at a future disaster site.

7 References

[1] Bluetooth SIG, Bluetooth technology radio versions, [Online]. Available: https://www.bluetooth.com/bluetooth-technology/radio-versions. [Accessed 04 04 2019].

[2] M. Marawaha, P. Jha, R. Razdan, J. Dukes, S. Sheehan, E. O. Nuallain. (2018), Performance Evaluation of Bluetooth Low Energy Communication, in *Journal of Information Sciences and Computing Technologies [Online]. 7(2)*, pp. 718-725, Available: http://www.scitecresearch.com/journals/index.php/jisct/article/view/1416.

[3] Nordic Semiconductor, Things you should know about bluetooth range, [Online]. Available: https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range. [Accessed 04 04 2019].

[4] Bluetooth SIG, Mesh Profile Specification 1.0.1, [Online]. Available: https://www.bluetooth.com/specifications/mesh-specifications. [Accessed 04 04 2019].

[5] J. de Carvalho Silva, J.J.P.C. Rodrigues, A.M. Alberti, P. Solic, A.L.L. Aquino, "LoRaWAN - A Low Power WAN Protocol for Internet of Things: a Review and Opportunities", presented at the 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech), Split, Croatia.

[6] I. Kuzminykh, A. Carlsson, R Franksson, A. Liljegren, "Measuring a LoRa Network: Performance, Possibilities and Limitations," in *Proceedings of the 18th International Conference, NEW2AN 2018, and 11th Conference (ruSMART'18)*, St. Petersburg, Russia, pp. 116-128.

[7] Ambiductor, OTAA, [Online]. Available: https://www.ambiductor.se/blog/lora-skola-hur-aktivera-otaa-respektive-abp. [Accessed 12 05 2019].

[8] TTN, [Online]. Available: https://www.thethingsnetwork.org/. [Accessed 13 05 2019].

[9] L. Sandman, N. Ekerstad, K. Lindroth, "Triage som prioriteringsinstrument på akutmottagning," Linköping University Electronic Press, Linköping, 2012.

[10] D. Nordström, D. Nyqvist, "Near Field Communication: En studie av säkerhetsaspekternas påverkan för mobila betalningar," Uppsala universitet, Uppsala, 2012.

[11] Electronics-notes, NFC-tag types, [Online]. Available: https://www.electronics-notes.com/articles/connectivity/nfc-near-field-communication/tags-types.php. [Accessed 12 05 2019].

[12] Nordic Semiconductor, Thingy:52, [Online]. Available: https://www.nordicsemi.com/?sc_itemid=%7B3C201A33-5CA5-457B-87E4-A7B04C19EE71%7D. [Accessed 07 05 2019].

[13] Nordic Semiconductor, nRF DK, [Online]. Available: https://www.nordicsemi.com/?sc_itemid=%7BF2C2DBF4-4D5C-4EAD-9F3D-CFD0276B300B%7D. [Accessed 08 05 2019].

[14] Elfa, PN532 NFC/RFID controller, Adafruit, [Online]. Available: https://www.elfa.se/sv/pn532-nfc-rfid-controller-adafruit-364/p/30129234?pos=1&origPos=1&origPageSize=10&prodprice=401.625&q=PN532+&p=cat-L1D_1859641~cat-DNAV_PL_1965016&isProductFamily=false&campaign=&track=true. [Accessed 07 05 2019].

[15] Dragino, LoRa/GPS-HAT, [Online]. Available: http://wiki.dragino.com/index.php?title=Lora/GPS_HAT. [Accessed 07 05 2019].

[16] Dragino, LoRa Mini, [Online]. Available: http://www.dragino.com/products/lora/item/125-lora-mini.html). [Accessed 07 05 2019].

[17] GitHub, Nordic-Thingy52-FW, [Online]. Available: https://github.com/NordicSemiconductor/Nordic-Thingy52-FW. [Accessed 08 05 2019].

[18] Devzone, segger studio, [Online]. Available: https://devzone.nordicsemi.com/f/nordic-q-a/27311/thingy-sdk-compile-error-using-segger-embedded-studio-within-windows-10. [Accessed 13 05 2019].

[19] Bosch BHI160BP, [Online]. Available: https://www.bosch-presse.de/pressportal/de/en/bosch-announces-industrys-first-position-tracking-smart-sensor-bhi160bp-for-wearables-175104.html. [Accessed 13 05 2019].

8 Appendix 1

Code from the main.c file in the Thingy:52 FW.

```c
/*
  Copyright (c) 2010 - 2017, Nordic Semiconductor ASA
  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification,
  are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this
     list of conditions and the following disclaimer.

  2. Redistributions in binary form, except as embedded into a Nordic
     Semiconductor ASA integrated circuit in a product or a software update for
     such product, must reproduce the above copyright notice, this list of
     conditions and the following disclaimer in the documentation and/or other
     materials provided with the distribution.

  3. Neither the name of Nordic Semiconductor ASA nor the names of its
     contributors may be used to endorse or promote products derived from this
     software without specific prior written permission.

  4. This software, with or without modification, must only be used with a
     Nordic Semiconductor ASA integrated circuit.

  5. Any software provided in binary form under this license must not be reverse
     engineered, decompiled, modified and/or disassembled.

  THIS SOFTWARE IS PROVIDED BY NORDIC SEMICONDUCTOR ASA "AS IS" AND ANY EXPRESS
  OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
  OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE ARE
  DISCLAIMED. IN NO EVENT SHALL NORDIC SEMICONDUCTOR ASA OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
  GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
  OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/** @file
 *
 * @brief    Thingy application main file.
 *
 * This file contains the source code for the Thingy application that uses the Weather Station service.
 */

#include <stdint.h>
#include <float.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "ble_hci.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "softdevice_handler.h"
#include "app_scheduler.h"
#include "app_button.h"
#include "app_util_platform.h"
#include "m_ble.h"
#include "m_environment.h"
```

```
#include "m_sound.h"
#include "m_motion.h"
#include "m_ui.h"
#include "m_batt_meas.h"
#include "drv_ext_light.h"
#include "drv_ext_gpio.h"
#include "nrf_delay.h"
#include "twi_manager.h"
#include "support_func.h"
#include "pca20020.h"
#include "app_error.h"

#define  NRF_LOG_MODULE_NAME "main          "
#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "drv_motion.h"
#include "drv_nfc.h"
#include "app_uart.h"
#include <stdio.h>
#include <stdbool.h>
#include "hardfault.h"
#include "sdk_macros.h"
#include "sdk_config.h"
#include "adafruit_pn532.h"
#include "nfc_ndef_msg_parser.h"
#include "nfc_t2t_parser.h"
#include "nrf_drv_twi.h"
#include "macros_common.h"
#include "drv_mic.h"
#include "drv_speaker.h"

#define TIMER_INTERVAL_VALUE_MS 120000
#define TIME_OUT_NFC_SCAN_MS 15000
#define NFC_SCAN_INTERVAL_MS 2000
#define SOUND_LISTNING_TIME_MS 2000
#define SOUND_STARTING_INTERVAL_MS 60000

bool sound_dectected = false;
uint8_t sound_state = 0;

       APP_TIMER_DEF(mic_time_out);

       APP_TIMER_DEF(reading_nfc_timer);

       //GPS formate N 58° 24.7035', E 15° 38.2040'
       ble_gps_data  gps_datax = { .triad_colour = 'F',
                                           .sensor_type = 'F',
                                           .gps_coordinatN = {'N','F','F','F','F','F','F','F','F',' '},
                                           .gps_coordinatE = {'E','F','F','F','F','F','F','F','F','F'},
                                           .nfc_scan_id = {'F','F'},
                                           .current_time = {'F','F',':','F','F',':','F','F'},
                                           .device_id = {'F','F',':','F','F',':','F','F',':','F','F',':','F','F',':','F','F'}
       };

       #if LORA_I2C_ON_OFF

       static const nrf_drv_twi_config_t twi_config_lora =
       {
       .scl                = TWI_SCL_EXT,
       .sda                = TWI_SDA_EXT,
       .frequency          = NRF_TWI_FREQ_400K,
       .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
       .clear_bus_init     = false
       };

       /* Common addresses definition for LoRa unit */
       #define LORA_ADDR          0x08U

       static const nrf_drv_twi_t m_twi_lora = NRF_DRV_TWI_INSTANCE(0);

       uint32_t write_data_to_lora(void);

       #endif

       #if  TAG_ON_OFF

       #define START_BYTE_ID_TRIAGE 7

       #define SEL_RES_CASCADE_BIT_NUM            3               /// Number of Cascade bit within SEL_RES byte.
       #define SEL_RES_TAG_PLATFORM_MASK          0x60            /// Mask of Tag Platform bit group within SEL_RES byte.
       #define SEL_RES_TAG_PLATFORM_BIT_OFFSET    5               /// Offset of the Tag Platform bit group within SEL_RES byte.

       #define TAG_TYPE_2_UID_LENGTH              7               /// Length of the Tag's UID.
       #define TAG_TYPE_2_DATA_AREA_SIZE_OFFSET   (T2T_CC_BLOCK_OFFSET + 2)        /// Offset of the byte with Tag's Data size.
       #define TAG_TYPE_2_DATA_AREA_MULTIPLICATOR 8               /// Multiplicator for a value stored in the Tag's Data size byte.
       #define TAG_TYPE_2_FIRST_DATA_BLOCK_NUM    (T2T_FIRST_DATA_BLOCK_OFFSET / T2T_BLOCK_SIZE) /// First block number with Tag's Data.
       #define TAG_TYPE_2_BLOCKS_PER_EXCHANGE     (T2T_MAX_DATA_EXCHANGE / T2T_BLOCK_SIZE)       /// Number of blocks fetched in single Tag's Read command.

       static uint8_t tag_id_triage[16];
       static bool confirmed_nfc_scan = false;

       /**
        * @brief Possible Tag Types.
        */
       typedef enum
       {
           NFC_T2T = 0x00,      ///< Type 2 Tag Platform.
           NFC_TT_NOT_SUPPORTED ///< Tag Type not supported.
       } nfc_tag_type_t;

       #endif

       #if  UART_ON_OFF

       #define UART_TX_BUF_SIZE 256                         /**< UART TX buffer size. */
       #define UART_RX_BUF_SIZE 256                         /**< UART RX buffer size. */

       static uint8_t rx_buffer[UART_RX_BUF_SIZE];

       static int uart_counter = 0;

       #endif

       APP_TIMER_DEF(sending_interval_timer);

       #define DEAD_BEEF   0xDEADBEEF          /**< Value used as error code on stack dump, can be used to identify stack location on stack unwind. */
```

```
#define SCHED_MAX_EVENT_DATA_SIZE   MAX(APP_TIMER_SCHED_EVENT_DATA_SIZE, BLE_STACK_HANDLER_SCHED_EVT_SIZE) /**< Maximum size of scheduler events. */
#define SCHED_QUEUE_SIZE            60  /**< Maximum number of events in the scheduler queue. */

static const nrf_drv_twi_t     m_twi_sensors = NRF_DRV_TWI_INSTANCE(TWI_SENSOR_INSTANCE);
static m_ble_service_handle_t  m_ble_service_handles[THINGY_SERVICES_MAX];


void app_error_fault_handler(uint32_t id, uint32_t pc, uint32_t info)
{
    #if NRF_LOG_ENABLED
        error_info_t * err_info = (error_info_t*)info;
        NRF_LOG_ERROR(" id = %d, pc = %d, file = %s, line number: %d, error code = %d = %s \r\n", \
        id, pc, nrf_log_push((char*)err_info->p_file_name), err_info->line_num, err_info->err_code, nrf_log_push((char*)nrf_strerror_find(err_info->err_code)));
    #endif

    (void)m_ui_led_set_event(M_UI_ERROR);
    NRF_LOG_FINAL_FLUSH();
    nrf_delay_ms(5);

    // On assert, the system can only recover with a reset.
    #ifndef DEBUG
        NVIC_SystemReset();
    #endif

    app_error_save_and_stop(id, pc, info);
}


/**@brief Function for assert macro callback.
 *
 * @details This function will be called in case of an assert in the SoftDevice.
 *
 * @warning On assert from the SoftDevice, the system can only recover on reset.
 *
 * @param[in] line_num    Line number of the failing ASSERT call.
 * @param[in] p_file_name File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}

/**@brief Function for putting Thingy into sleep mode.
 *
 * @note This function will not return.
 */
 //Turned off since the GPS-tracker needs to always be active as the implementation looks now
/*static void sleep_mode_enter(void)
{
    uint32_t err_code;

    NRF_LOG_INFO("Entering sleep mode \r\n");
    err_code = m_motion_sleep_prepare(true);
    APP_ERROR_CHECK(err_code);

    err_code = support_func_configure_io_shutdown();
    APP_ERROR_CHECK(err_code);
```

```

        // Enable wake on button press.
        nrf_gpio_cfg_sense_input(BUTTON, NRF_GPIO_PIN_PULLUP, NRF_GPIO_PIN_SENSE_LOW);
        // Enable wake on low power accelerometer.
        nrf_gpio_cfg_sense_input(LIS_INT1, NRF_GPIO_PIN_NOPULL, NRF_GPIO_PIN_SENSE_HIGH);

        NRF_LOG_FLUSH();
        nrf_delay_ms(5);
        // Go to system-off (sd_power_system_off() will not return; wakeup will cause a reset). When debugging, this function may return and code execution will continue.
        err_code = sd_power_system_off();
        NRF_LOG_WARNING("sd_power_system_off() returned. -Probably due to debugger being used. Instructions will still run. \r\n");
        NRF_LOG_FLUSH();

        #ifdef DEBUG
            if(!support_func_sys_halt_debug_enabled())
            {
                APP_ERROR_CHECK(err_code); // If not in debug mode, return the error and the system will reboot.
            }
            else
            {
                NRF_LOG_WARNING("Exec stopped, busy wait \r\n");
                NRF_LOG_FLUSH();

                while(true) // Only reachable when entering emulated system off.
                {
                    // Infinte loop to ensure that code stops in debug mode.
                }
            }
        #else
            APP_ERROR_CHECK(err_code);
        #endif
    }
    */

    /**@brief Function for placing the application in low power state while waiting for events.
     */

    #define FPU_EXCEPTION_MASK 0x0000009F

    static void power_manage(void)
    {
        __set_FPSCR(__get_FPSCR()  & ~(FPU_EXCEPTION_MASK));
        (void) __get_FPSCR();
        NVIC_ClearPendingIRQ(FPU_IRQn);

        uint32_t err_code = sd_app_evt_wait();
        APP_ERROR_CHECK(err_code);
    }


    /**@brief Battery module data handler.
     */
    static void m_batt_meas_handler(m_batt_meas_event_t const * p_batt_meas_event)
    {
        NRF_LOG_INFO("Voltage: %d V, Charge: %d %%, Event type: %d \r\n",
                    p_batt_meas_event->voltage_mv, p_batt_meas_event->level_percent, p_batt_meas_event->type);

        if (p_batt_meas_event != NULL)
        {
            if( p_batt_meas_event->type == M_BATT_MEAS_EVENT_LOW)
            {
                uint32_t err_code;

                err_code = support_func_configure_io_shutdown();
                APP_ERROR_CHECK(err_code);

                // Enable wake on USB detect only.
                nrf_gpio_cfg_sense_input(USB_DETECT, NRF_GPIO_PIN_NOPULL, NRF_GPIO_PIN_SENSE_HIGH);

                NRF_LOG_WARNING("Battery voltage low, shutting down Thingy. Connect USB to charge \r\n");
                NRF_LOG_FINAL_FLUSH();
                // Go to system-off mode (This function will not return; wakeup will cause a reset).
                err_code = sd_power_system_off();

                #ifdef DEBUG
                    if(!support_func_sys_halt_debug_enabled())
                    {
                        APP_ERROR_CHECK(err_code); // If not in debug mode, return the error and the system will reboot.
                    }
                    else
                    {
                        NRF_LOG_WARNING("Exec stopped, busy wait \r\n");
                        NRF_LOG_FLUSH();
                        while(true) // Only reachable when entering emulated system off.
                        {
                            // Infinte loop to ensure that code stops in debug mode.
                        }
                    }
                #else
                    APP_ERROR_CHECK(err_code);
                #endif
            }
        }
    }


    /**@brief Function for handling BLE events.
     */
    static void thingy_ble_evt_handler(m_ble_evt_t * p_evt)
    {
        switch (p_evt->evt_type)
        {
            case thingy_ble_evt_connected:
                NRF_LOG_INFO(NRF_LOG_COLOR_CODE_GREEN "Thingy_ble_evt_connected \r\n");
                break;

            case thingy_ble_evt_disconnected:
                NRF_LOG_INFO(NRF_LOG_COLOR_CODE_YELLOW "Thingy_ble_evt_disconnected \r\n");
                NRF_LOG_FINAL_FLUSH();
                nrf_delay_ms(5);
                //NVIC_SystemReset(); //Turned off since the GPS-tracker needs to always be active as the implementation looks now
                break;

            case thingy_ble_evt_timeout:
```

```
            NRF_LOG_INFO(NRF_LOG_COLOR_CODE_YELLOW "Thingy_ble_evt_timeout \r\n");
            //sleep_mode_enter();//Turned off since the GPS-tracker needs to always be active as the implementation looks now
            //NVIC_SystemReset();//Turned off since the GPS-tracker needs to always be active as the implementation looks now
            break;
    }
}


/**@brief Function for initializing the Thingy.
 */
static void thingy_init(void)
{
    uint32_t                 err_code;
    m_ui_init_t              ui_params;
    m_environment_init_t     env_params;
    m_motion_init_t          motion_params;
    m_ble_init_t             ble_params;
    batt_meas_init_t         batt_meas_init = BATT_MEAS_PARAM_CFG;

    /**@brief Initialize the TWI manager. */
    err_code = twi_manager_init(APP_IRQ_PRIORITY_THREAD);
    APP_ERROR_CHECK(err_code);

    /**@brief Initialize LED and button UI module. */
    ui_params.p_twi_instance = &m_twi_sensors;
    err_code = m_ui_init(&m_ble_service_handles[THINGY_SERVICE_UI],
                         &ui_params);
    APP_ERROR_CHECK(err_code);

    /**@brief Initialize environment module. */
    env_params.p_twi_instance = &m_twi_sensors;
    err_code = m_environment_init(&m_ble_service_handles[THINGY_SERVICE_ENVIRONMENT],
                                  &env_params);
    APP_ERROR_CHECK(err_code);

    /**@brief Initialize motion module. */
    motion_params.p_twi_instance = &m_twi_sensors;

    err_code = m_motion_init(&m_ble_service_handles[THINGY_SERVICE_MOTION],
                             &motion_params);
    APP_ERROR_CHECK(err_code);

    err_code = m_sound_init(&m_ble_service_handles[THINGY_SERVICE_SOUND]);
    APP_ERROR_CHECK(err_code);

    /**@brief Initialize the battery measurement. */
    batt_meas_init.evt_handler = m_batt_meas_handler;
    err_code = m_batt_meas_init(&m_ble_service_handles[THINGY_SERVICE_BATTERY], &batt_meas_init);
    APP_ERROR_CHECK(err_code);

    err_code = m_batt_meas_enable(BATT_MEAS_INTERVAL_MS);
    APP_ERROR_CHECK(err_code);

    /**@brief Initialize BLE handling module. */
    ble_params.evt_handler       = thingy_ble_evt_handler;
    ble_params.p_service_handles = m_ble_service_handles;
    ble_params.service_num       = THINGY_SERVICES_MAX;

    err_code = m_ble_init(&ble_params);
    APP_ERROR_CHECK(err_code);

    err_code = m_ui_led_set_event(M_UI_BLE_DISCONNECTED);
    APP_ERROR_CHECK(err_code);

}


static void board_init(void)
{
    uint32_t            err_code;
    drv_ext_gpio_init_t ext_gpio_init;

    #if defined(THINGY_HW_v0_7_0)
        #error   "HW version v0.7.0 not supported."
    #elif defined(THINGY_HW_v0_8_0)
        NRF_LOG_WARNING("FW compiled for depricated Thingy HW v0.8.0 \r\n");

    #elif defined(THINGY_HW_v0_9_0)
        NRF_LOG_WARNING("FW compiled for depricated Thingy HW v0.9.0 \r\n");

    #endif

    static const nrf_drv_twi_config_t twi_config =
    {
        .scl                = TWI_SCL,
        .sda                = TWI_SDA,
        .frequency          = NRF_TWI_FREQ_400K,
        .interrupt_priority = APP_IRQ_PRIORITY_LOW
    };

    static const drv_sx1509_cfg_t sx1509_cfg =
    {
        .twi_addr       = SX1509_ADDR,
        .p_twi_instance = &m_twi_sensors,
        .p_twi_cfg      = &twi_config
    };

    ext_gpio_init.p_cfg = &sx1509_cfg;

    err_code = support_func_configure_io_startup(&ext_gpio_init);
    APP_ERROR_CHECK(err_code);

    nrf_delay_ms(100);
}

#if LORA_I2C_ON_OFF

/**@brief Open the TWI (I2C) bus for communication for LoRa.
 */
static __inline ret_code_t twi_open(void)
{
    ret_code_t err_code;
    err_code = twi_manager_request(&m_twi_lora,
                                   &twi_config_lora,
                                   NULL,
                                   NULL);
    RETURN_IF_ERROR(err_code);
    nrf_drv_twi_enable(&m_twi_lora);
    return NRF_SUCCESS;
}


/**@brief Close the TWI (I2C) bus after LoRa communications are finished.
 */
static __inline ret_code_t twi_close(void)
{
    nrf_drv_twi_disable(&m_twi_lora);
```

```
        nrf_drv_twi_uninit(&m_twi_lora);
        return NRF_SUCCESS;
    }

    uint32_t write_data_to_lora(void)
    {
        ret_code_t err_code;

        err_code = twi_open();
        APP_ERROR_CHECK(err_code);

        //sending two packed of data to lora module,32 bytes = max tranfer per package
        err_code = nrf_drv_twi_tx(&m_twi_lora, LORA_ADDR, (uint8_t *)&gps_datax, sizeof(ble_gps_data) - sizeof(gps_datax.device_id), false);
        err_code = nrf_drv_twi_tx(&m_twi_lora, LORA_ADDR, (uint8_t *)&gps_datax.device_id, sizeof(gps_datax.device_id), false);

        twi_close();
        RETURN_IF_ERROR(err_code);

        NRF_LOG_DEBUG("%s\r\n\r\n",(uint32_t)((uint8_t*)&gps_datax));
        return NRF_SUCCESS;
    }

    #endif

    #if  UART_ON_OFF

    void set_gps()
    {
      gps_datax.gps_coordinatN[0] = 'N';
      for(int i = 7; i <= 15; i++)
      {
        if(i > 10)
        {
          gps_datax.gps_coordinatN[i-6] = rx_buffer[i+1];
        }
        else
        {
          gps_datax.gps_coordinatN[i-6] = rx_buffer[i];
        }
      }
      gps_datax.gps_coordinatN[9] = ' ';

      gps_datax.gps_coordinatE[0] = 'E';

      for(int i = 19; i <= 28; i++)
      {
        if(i > 23)
        {
          gps_datax.gps_coordinatE[i-18] = rx_buffer[i+1];
        }
        else
        {
          gps_datax.gps_coordinatE[i-18] = rx_buffer[i];
        }
      }
    }

    void set_time()
    {
      gps_datax.current_time[0] = rx_buffer[7];
      gps_datax.current_time[1] = rx_buffer[8];
      gps_datax.current_time[2] = ':';
      gps_datax.current_time[3] = rx_buffer[9];
      gps_datax.current_time[4] = rx_buffer[10];
      gps_datax.current_time[5] = ':';
      gps_datax.current_time[6] = rx_buffer[11];
      gps_datax.current_time[7] = rx_buffer[12];
    }

    uint8_t calc_check_byte(uint8_t * check_byte)
    {
      uint8_t res = 0;
      for(int i = 0; check_byte[i] != '*' ;++i)
      {
        res ^= check_byte[i];
      }
      return res;
    }

    static void uart_send(uint8_t * data_to_send)
    {
      app_uart_put('$');
      for(int i = 0; data_to_send[i] != '*' ;++i)
      {
        app_uart_put(data_to_send[i]);
      }
      app_uart_put('*');

      //makes a ascii number from a uint_8
      if(((calc_check_byte(data_to_send) >> 4) & 0x0F) < 10)
      {
        app_uart_put('0' + ((calc_check_byte(data_to_send) >> 4) & 0x0F));
      }
      else
      {
        app_uart_put('A' - 10 + ((calc_check_byte(data_to_send) >> 4) & 0x0F));
      }

      if(((calc_check_byte(data_to_send) ) & 0x0F) < 10)
      {
        app_uart_put('0' + ((calc_check_byte(data_to_send) ) & 0x0F));
      }
      else
      {
        app_uart_put('A' - 10 + ((calc_check_byte(data_to_send) ) & 0x0F));
      }

      app_uart_put('\r');
      app_uart_put('\n');
    }

    void uart_event_handler(app_uart_evt_t * p_event)
    {
      if (p_event->evt_type == APP_UART_COMMUNICATION_ERROR)
      {
        APP_ERROR_HANDLER(p_event->data.error_communication);
      }
      else if (p_event->evt_type == APP_UART_FIFO_ERROR)
      {
        APP_ERROR_HANDLER(p_event->data.error_code);
      }
      else if (p_event->evt_type == APP_UART_TX_EMPTY) //TX has completed its sending
      {

      }
      else if (p_event->evt_type == APP_UART_DATA_READY) // Data is availible and can be fetch by app_uart_get
```

```
        {
          uint8_t   rx_data;
          app_uart_get(&rx_data);
          rx_buffer[uart_counter] = rx_data;
          uart_counter++;

          if(rx_data == '\n') //a whole message is received
          {
            uart_counter = 0;

            if (rx_buffer[4] == 'L' && rx_buffer[43] == 'A')
            {
              set_gps();
            }
            else if(rx_buffer[3] == 'Z')
            {
              set_time();
            }
            else if(rx_buffer[4] == 'X')
            {
              //do nothing
            }
            else //not valid gps data write 'F' to coordinates and time
            {
              char gps_coordinatN_overwrite[10] = {'N','F','F','F','F','F','F','F','F',' '};
              char gps_coordinatE_overwrite[10] = {'E','F','F','F','F','F','F','F','F','F'};
              char current_time_overwrite[8] = {'F','F',':','F','F',':','F','F'};
              memcpy(gps_datax.gps_coordinatN, gps_coordinatN_overwrite, sizeof(gps_datax.gps_coordinatN));
              memcpy(gps_datax.gps_coordinatE, gps_coordinatE_overwrite, sizeof(gps_datax.gps_coordinatE));
              memcpy(gps_datax.current_time, current_time_overwrite, sizeof(gps_datax.current_time));
            }
            NRF_LOG_DEBUG("%s\r\n",(uint32_t)((uint8_t*)rx_buffer)); //print recived UART data
            NRF_LOG_FLUSH();

            memset(rx_buffer, 0, sizeof(rx_buffer)); //resets the array only wors on (char, short, int, long, long long)
          }
        }
      }

      #endif

      static void time_out_reaction(void * p_context)
      {
        ret_code_t err_code;

        #if LORA_I2C_ON_OFF

        gps_datax.sensor_type = TIME_OUT;

        write_data_to_lora();

        #endif

        send_gps_data_ble();

        err_code = app_timer_start(sending_interval_timer, APP_TIMER_TICKS(TIMER_INTERVAL_VALUE_MS), NULL);
        APP_ERROR_CHECK(err_code);
      }
```

```

#if  TAG_ON_OFF

/**
 * @brief Function for reading data  from a Type 2 Tag Platform.
 */
ret_code_t t2t_data_read(nfc_a_tag_info * p_tag_info, uint8_t * buffer, uint32_t buffer_size)
{
    ret_code_t err_code;
    uint8_t    block_num = 0;

    // Not enough size in the buffer to read a tag header.
    if (buffer_size < T2T_FIRST_DATA_BLOCK_OFFSET)
    {
        return NRF_ERROR_NO_MEM;
    }

    if (p_tag_info->nfc_id_len != TAG_TYPE_2_UID_LENGTH)
    {
        return NRF_ERROR_NOT_SUPPORTED;
    }

    // Read blocks 0 - 3 to get the header information.
    err_code = adafruit_pn532_tag2_read(block_num, buffer);
    if (err_code)
    {
        NRF_LOG_INFO("Failed to read blocks: %d-%d\r\n", block_num,
                     block_num + T2T_END_PAGE_OFFSET);
        return NRF_ERROR_INTERNAL;
    }

    uint16_t data_bytes_in_tag = TAG_TYPE_2_DATA_AREA_MULTIPLICATOR *
                                 buffer[TAG_TYPE_2_DATA_AREA_SIZE_OFFSET];

    if (data_bytes_in_tag + T2T_FIRST_DATA_BLOCK_OFFSET > buffer_size)
    {
        return NRF_ERROR_NO_MEM;
    }

    err_code = adafruit_pn532_tag2_read(START_BYTE_ID_TRIAGE, tag_id_triage);
    if (err_code)
    {
        NRF_LOG_INFO("Failed to read blocks: %d-%d\r\n",
                     START_BYTE_ID_TRIAGE,
                     START_BYTE_ID_TRIAGE + 4);
        return NRF_ERROR_INTERNAL;
    }

    return NRF_SUCCESS;
}


/**
 * @brief Function for reading and analyzing data from a Type 2 Tag Platform.
 *
 * This function reads content of a Type 2 Tag Platform, parses it and prints it out.
 */
ret_code_t t2t_data_read_and_analyze(nfc_a_tag_info * p_tag_info)
{
    ret_code_t     err_code;
    static uint8_t t2t_data[TAG_TYPE_2_DATA_BUFFER_SIZE]; // Buffer for tag data.
```

```

        err_code = t2t_data_read(p_tag_info, t2t_data, TAG_TYPE_2_DATA_BUFFER_SIZE);
        VERIFY_SUCCESS(err_code);

        //t2t_data_analyze(t2t_data);

        return NRF_SUCCESS;
    }


    /**
     * @brief Function for identifying Tag Platform Type.
     */
    nfc_tag_type_t tag_type_identify(uint8_t sel_res)
    {
        uint8_t platform_config;

        // Check if Cascade bit in SEL_RES response is cleared. Cleared bit indicates that NFCID1 complete.
        if (!IS_SET(sel_res, SEL_RES_CASCADE_BIT_NUM))
        {
            // Extract platform configuration from SEL_RES response.
            platform_config = (sel_res & SEL_RES_TAG_PLATFORM_MASK) >> SEL_RES_TAG_PLATFORM_BIT_OFFSET;
            if (platform_config < NFC_TT_NOT_SUPPORTED)
            {
                return (nfc_tag_type_t) platform_config;
            }
        }

        return NFC_TT_NOT_SUPPORTED;
    }

    /**
     * @brief Function for detecting a Tag, identifying its Type and reading data from it.
     *
     * This function waits for a Tag to appear in the field. When a Tag is detected, Tag Platform
     * Type (2/4) is identified and appropriate read procedure is run.
     */
    ret_code_t tag_detect_and_read(void)
    {
        ret_code_t     err_code;
        nfc_a_tag_info tag_info;

        // Detect a NFC-A Tag in the field and initiate a communication. This function activates
        // the NFC RF field. If a Tag is present, basic information about detected Tag is returned
        // in tag info structure.
        err_code = adafruit_pn532_nfc_a_target_init(&tag_info, TAG_DETECT_TIMEOUT);

        if (err_code != NRF_SUCCESS)
        {
            return NRF_ERROR_NOT_FOUND;
        }
        //adafruit_pn532_tag_info_printout(&tag_info);

        nfc_tag_type_t tag_type = tag_type_identify(tag_info.sel_res);
        switch (tag_type)
        {
            case NFC_T2T:
                NRF_LOG_INFO("Type 2 Tag Platform detected. \r\n");
```

```
                    return t2t_data_read_and_analyze(&tag_info);

                default:
                    return NRF_ERROR_NOT_SUPPORTED;
            }
        }


        /**
         * @brief Function for waiting specified time after a Tag read operation.
         */
        void after_read_delay(void)
        {
            ret_code_t err_code;

            // Turn off the RF field.
            err_code = adafruit_pn532_field_off();
            APP_ERROR_CHECK(err_code);
            nrf_delay_ms(TAG_AFTER_READ_DELAY);
        }


        static void time_out_nfc_reader(void * p_context)
        {
          ret_code_t err_code;
          drv_ext_light_rgb_intensity_t rgb_color;

          err_code = twi_manager_request(ada_twi_instance(),
                                                ada_twi_config(),
                                                NULL,
                                                NULL);

          if(err_code == NRF_SUCCESS)
          {
            //open twi communication
           nrf_drv_twi_enable(ada_twi_instance());

            err_code = tag_detect_and_read();
            switch (err_code)
            {
                case NRF_SUCCESS:
                    after_read_delay();
                    break;

                case NRF_ERROR_NO_MEM:
                    NRF_LOG_INFO("Declared buffer for T2T is to small to store tag da
ta.\r\n");
                    after_read_delay();
                    break;

                case NRF_ERROR_NOT_FOUND:
                    NRF_LOG_INFO("No Tag found.\r\n");
                    // No delay here as we want to search for another tag immediately
.
                    break;

                case NRF_ERROR_NOT_SUPPORTED:
                    NRF_LOG_INFO("Tag not supported.\r\n");
                    after_read_delay();
                    break;

                default:
                    NRF_LOG_INFO("Error during tag read.\r\n");
                    err_code = adafruit_pn532_field_off();
                    break;
            }
```

```
      NRF_LOG_FLUSH();
       //cloes twi communication so other moduls can use it
      nrf_drv_twi_disable(ada_twi_instance());
      nrf_drv_twi_uninit(ada_twi_instance());

     }

     if(err_code == NRF_SUCCESS)//Add a extra long delay until next nfc-read if nfc read success
     {
      gps_datax.nfc_scan_id[0] = tag_id_triage[5];
      gps_datax.nfc_scan_id[1] = tag_id_triage[6];
      gps_datax.triad_colour   = tag_id_triage[12] - '0'; // convert from to int value from char value by subractiong '0'
      gps_datax.sensor_type = NFC_SCAN;

      rgb_color.r = 0;
      rgb_color.g = 50;
      rgb_color.b = 0;

      set_led_for_event(rgb_color,BLE_UIS_LED_MODE_CONST);
      confirmed_nfc_scan = true;

      err_code = drv_speaker_tone_start(1000,2000,50);
      APP_ERROR_CHECK(err_code);

      send_gps_data_ble();

      #if LORA_I2C_ON_OFF

      write_data_to_lora();

      #endif

      err_code = app_timer_start(reading_nfc_timer, APP_TIMER_TICKS(TIME_OUT_NFC_SCAN_MS), NULL);
      APP_ERROR_CHECK(err_code);
     }
     else
     {
      err_code = app_timer_start(reading_nfc_timer, APP_TIMER_TICKS(NFC_SCAN_INTERVAL_MS), NULL);
      APP_ERROR_CHECK(err_code);

      if(confirmed_nfc_scan)
      {
       rgb_color.r = 0;
       rgb_color.g = 0;
       rgb_color.b = 0;

       set_led_for_event(rgb_color,BLE_UIS_LED_MODE_CONST);
       confirmed_nfc_scan = false;
      }
     }

    }

    #endif

    static void mic_in_handler(void * p_context)
    {
      int32_t err_code;

      switch (sound_state)
      {
        case 0:
```

```
                   err_code = drv_mic_start();
                   APP_ERROR_CHECK(err_code);

                   err_code = app_timer_start(mic_time_out, APP_TIMER_TICKS(SOUND_LISTNING_TIME_MS), NULL);
                   APP_ERROR_CHECK(err_code);

                   sound_state++;

                   break;

               case 1:
                   if (sound_dectected)
                   {
                     sound_dectected = false;
                     gps_datax.sensor_type = SOUND_DETECTION;

                     #if LORA_I2C_ON_OFF

                     write_data_to_lora();

                     #endif

                     send_gps_data_ble();
                   }
                   else
                   {
                     //do nothing sound not high enough for detection
                   }

                   sound_state = 0;

                   err_code = drv_mic_stop();
                   APP_ERROR_CHECK(err_code);

                   err_code = app_timer_start(mic_time_out, APP_TIMER_TICKS(SOUND_STARTING_INTERVAL_MS), NULL);
                   APP_ERROR_CHECK(err_code);

                   break;

           }

       }

       static void gps_tracker_init(void)
       {
         uint32_t err_code;

         NRF_LOG_DEBUG("enable origentation detection\r\n\r\n");
         err_code = drv_motion_enable(DRV_MOTION_FEATURE_MASK_ORIENTATION); //enable origentation detection
         APP_ERROR_CHECK(err_code);                                         //enable origentation detection

         NRF_LOG_DEBUG("enable step detection\r\n\r\n");
         err_code = drv_motion_enable(DRV_MOTION_FEATURE_MASK_PEDOMETER);  //enable step detection
         APP_ERROR_CHECK(err_code);                                        //enable step detection

         NRF_LOG_DEBUG("creating timer\r\n\r\n");
         err_code = app_timer_create(&sending_interval_timer, APP_TIMER_MODE_SINGLE_SHOT, time_out_reaction);
         APP_ERROR_CHECK(err_code);

```

```
        NRF_LOG_DEBUG("starting timer\r\n\r\n");
        err_code = app_timer_start(sending_interval_timer, APP_TIMER_TICKS(TIMER_INTERVAL_VALUE_MS), NULL);
        APP_ERROR_CHECK(err_code);

        #if  UART_ON_OFF

        NRF_LOG_DEBUG("setting UART params\r\n\r\n");
        const app_uart_comm_params_t comm_params =
        {
          ANA_DIG1,
          ANA_DIG2,
          MOS_1,
          MOS_2,
          APP_UART_FLOW_CONTROL_DISABLED,
          false,
          UART_BAUDRATE_BAUDRATE_Baud9600
        };

        NRF_LOG_DEBUG("INIT UART communication\r\n\r\n");
        APP_UART_FIFO_INIT(&comm_params,
          UART_RX_BUF_SIZE,
          UART_TX_BUF_SIZE,
          uart_event_handler,
          APP_IRQ_PRIORITY_LOWEST,
          err_code);

        NRF_LOG_DEBUG("Sending commands to GPS-module\r\n\r\n");
        // Sending UART settings twice to be sure the command arrives
        // setting the gps to only send information every Xms
        uint8_t * send_via_uart_data = (uint8_t *)("PMTK220,10000*");
        uart_send(send_via_uart_data);
        nrf_delay_ms(75);
        uart_send(send_via_uart_data);

        // setting the gps to only send time and N and E coordinates
        send_via_uart_data = (uint8_t *)("PMTK314,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0*");
        uart_send(send_via_uart_data);
        nrf_delay_ms(75);
        uart_send(send_via_uart_data);

        #endif

        #if LORA_I2C_ON_OFF

        NRF_LOG_DEBUG("INIT TWI connection\r\n\r\n");
        err_code = nrf_drv_twi_init(&m_twi_lora, &twi_config_lora, NULL, NULL);
        APP_ERROR_CHECK(err_code);

        NRF_LOG_DEBUG("Enable TWI connection for LoRa\r\n\r\n");
        nrf_drv_twi_enable(&m_twi_lora);

        NRF_LOG_DEBUG("TWI connection LoRa closing to leave it for other modules\r\n\r\n");
        twi_close();

        #endif

        #if  TAG_ON_OFF

        NRF_LOG_DEBUG("Creating timer for NFC-reads\r\n\r\n");
        err_code = app_timer_create(&reading_nfc_timer, APP_TIMER_MODE_SINGLE_SHOT, time_out_nfc_reader);
        APP_ERROR_CHECK(err_code);

```

```c
  NRF_LOG_DEBUG("Starting timer for NFC-reads\r\n\r\n");
  err_code = app_timer_start(reading_nfc_timer, APP_TIMER_TICKS(NFC_SCAN_INTERVAL_MS), NULL);
  APP_ERROR_CHECK(err_code);

  NRF_LOG_DEBUG("Starting init of NFC-reader\r\n\r\n");
  err_code = adafruit_pn532_init(false);
  APP_ERROR_CHECK(err_code);

  NRF_LOG_DEBUG("TWI connection NFC closing to leave it for other modules\r\n\r\n");
  nrf_drv_twi_disable(ada_twi_instance());
  nrf_drv_twi_uninit(ada_twi_instance());

  #endif

  NRF_LOG_DEBUG("Creating timer for sound readings\r\n\r\n");
  err_code = app_timer_create(&mic_time_out, APP_TIMER_MODE_SINGLE_SHOT, mic_in_handler);
  APP_ERROR_CHECK(err_code);

  NRF_LOG_DEBUG("Start timer for sound readings\r\n\r\n");
  err_code = app_timer_start(mic_time_out, APP_TIMER_TICKS(SOUND_STARTING_INTERVAL_MS), NULL);
  APP_ERROR_CHECK(err_code);

  //Gets the device mac address
  NRF_LOG_DEBUG("Getting Mac address\r\n\r\n");
  char m_mac_addr[SUPPORT_FUNC_MAC_ADDR_STR_LEN];
  err_code = support_func_ble_mac_address_get(m_mac_addr);
  APP_ERROR_CHECK(err_code);
  memcpy(gps_datax.device_id, m_mac_addr, sizeof(gps_datax.device_id));

  NRF_LOG_DEBUG("Turn of the LED\r\n\r\n");
  drv_ext_light_rgb_intensity_t rgb_color;
  rgb_color.r = 0;
  rgb_color.g = 0;
  rgb_color.b = 0;
  set_led_for_event(rgb_color,BLE_UIS_LED_MODE_CONST);

  NRF_LOG_DEBUG("GPS_tracker_INIT is finished\r\n\r\n");
}

/**@brief Application main function.
 */
int main(void)
{
    uint32_t err_code;
    err_code = NRF_LOG_INIT(NULL);
    APP_ERROR_CHECK(err_code);

    NRF_LOG_INFO(NRF_LOG_COLOR_CODE_GREEN"===== Thingy started! =====\r\n");

    // Initialize.
    APP_SCHED_INIT(SCHED_MAX_EVENT_DATA_SIZE, SCHED_QUEUE_SIZE);
    err_code = app_timer_init();
    APP_ERROR_CHECK(err_code);

    board_init();
    thingy_init();
    gps_tracker_init();

    for (;;)
    {
       app_sched_execute();
```

```
                if (!NRF_LOG_PROCESS()) // Process logs
                {
                    power_manage();
                }
            }
        }
```

9 Appendix 2

Code snippet from the sdk_config.h file in the Thingy:52 FW.

```
/**
 * Copyright (c) 2017 - 2017, Nordic Semiconductor ASA
 *
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice, this
 *    list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form, except as embedded into a Nordic
 *    Semiconductor ASA integrated circuit in a product or a software update for
 *    such product, must reproduce the above copyright notice, this list of
 *    conditions and the following disclaimer in the documentation and/or other
 *    materials provided with the distribution.
 *
 * 3. Neither the name of Nordic Semiconductor ASA nor the names of its
 *    contributors may be used to endorse or promote products derived from this
 *    software without specific prior written permission.
 *
 * 4. This software, with or without modification, must only be used with a
 *    Nordic Semiconductor ASA integrated circuit.
 *
 * 5. Any software provided in binary form under this license must not be reverse
 *    engineered, decompiled, modified and/or disassembled.
 *
 * THIS SOFTWARE IS PROVIDED BY NORDIC SEMICONDUCTOR ASA "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL NORDIC SEMICONDUCTOR ASA OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */

#ifndef SDK_CONFIG_H
#define SDK_CONFIG_H
// <<< Use Configuration Wizard in Context Menu >>>\n
#ifdef USE_APP_CONFIG
#include "app_config.h"
#endif

#define SOUND_THRESHOLD 117

#define NUMBER_OF_STEP_DETECTION 100

#ifndef UART_ON_OFF
#define UART_ON_OFF 1
#endif

#ifndef TAG_ON_OFF
#define TAG_ON_OFF 1
#endif

#ifndef LORA_I2C_ON_OFF
#define LORA_I2C_ON_OFF 1
```

```
#endif

#ifndef LIGHT_ON_SENSOR_REACTION
#define LIGHT_ON_SENSOR_REACTION 1
#endif

//==========================================================
// <q> APP_FIFO_ENABLED  - app_fifo - Software FIFO implementation


#ifndef APP_FIFO_ENABLED
#define APP_FIFO_ENABLED 1
#endif

// <e> APP_UART_ENABLED - app_uart - UART driver
//==========================================================
#ifndef APP_UART_ENABLED
#define APP_UART_ENABLED 1
#endif
#if  APP_UART_ENABLED
// <o> APP_UART_DRIVER_INSTANCE  - UART instance used

// <0=> 0

#ifndef APP_UART_DRIVER_INSTANCE
#define APP_UART_DRIVER_INSTANCE 0
#endif

#endif //APP_UART_ENABLED
// </e>

// <q> RETARGET_ENABLED  - retarget - Retargeting stdio functions


#ifndef RETARGET_ENABLED
#define RETARGET_ENABLED 1
#endif


       //==========================================================
       // <o> MAX_NDEF_RECORDS - Maximal number of NDEF records that can be stored in NDEF message descriptor.
       #ifndef MAX_NDEF_RECORDS
       #define MAX_NDEF_RECORDS 10
       #endif

       // <o> MAX_TLV_BLOCKS - Maximal number of TLV blocks that can be stored in Type 2/4 Tag descriptor.
       #ifndef MAX_TLV_BLOCKS
       #define MAX_TLV_BLOCKS 10
       #endif

       // <o> TAG_AFTER_READ_DELAY - Delay (in ms) after Tag read.
       #ifndef TAG_AFTER_READ_DELAY
       #define TAG_AFTER_READ_DELAY 2000
       #endif

       // <o> TAG_DETECT_TIMEOUT - Timeout (in ms) for function which searches for a Tag.
       #ifndef TAG_DETECT_TIMEOUT
       #define TAG_DETECT_TIMEOUT 5000
       #endif

       // <o> TAG_TYPE_2_DATA_BUFFER_SIZE - Buffer size for data from a Type 2 Tag Platform.
       #ifndef TAG_TYPE_2_DATA_BUFFER_SIZE
```

```
#define TAG_TYPE_2_DATA_BUFFER_SIZE 1024
#endif

// </h>
//==========================================================

// <h> Third Parties

//==========================================================
// <e> ADAFRUIT_PN532_ENABLED - adafruit_pn532 - Adafruit PN532 implementation for nRF5x
//==========================================================
#ifndef ADAFRUIT_PN532_ENABLED
#define ADAFRUIT_PN532_ENABLED 1
#endif
#if  ADAFRUIT_PN532_ENABLED
// <o> PN532_IRQ  - Pin number

// <0=> 0 (P0.0)
// <1=> 1 (P0.1)
// <2=> 2 (P0.2)
// <3=> 3 (P0.3)
// <4=> 4 (P0.4)
// <5=> 5 (P0.5)
// <6=> 6 (P0.6)
// <7=> 7 (P0.7)
// <8=> 8 (P0.8)
// <9=> 9 (P0.9)
// <10=> 10 (P0.10)
// <11=> 11 (P0.11)
// <12=> 12 (P0.12)
// <13=> 13 (P0.13)
// <14=> 14 (P0.14)
// <15=> 15 (P0.15)
// <16=> 16 (P0.16)
// <17=> 17 (P0.17)
// <18=> 18 (P0.18)
// <19=> 19 (P0.19)
// <20=> 20 (P0.20)
// <21=> 21 (P0.21)
// <22=> 22 (P0.22)
// <23=> 23 (P0.23)
// <24=> 24 (P0.24)
// <25=> 25 (P0.25)
// <26=> 26 (P0.26)
// <27=> 27 (P0.27)
// <28=> 28 (P0.28)
// <29=> 29 (P0.29)
// <30=> 30 (P0.30)
// <31=> 31 (P0.31)
// <4294967295=> Not connected

#ifndef PN532_IRQ
//#define PN532_IRQ 0  // SX_IOEXT_0
#define PN532_IRQ 2  //ANA_DIG0
#endif

// <o> PN532_RESET  - Pin number

// <0=> 0 (P0.0)
// <1=> 1 (P0.1)
// <2=> 2 (P0.2)
// <3=> 3 (P0.3)
// <4=> 4 (P0.4)
// <5=> 5 (P0.5)
// <6=> 6 (P0.6)
```

```
        // <7=> 7 (P0.7)
        // <8=> 8 (P0.8)
        // <9=> 9 (P0.9)
        // <10=> 10 (P0.10)
        // <11=> 11 (P0.11)
        // <12=> 12 (P0.12)
        // <13=> 13 (P0.13)
        // <14=> 14 (P0.14)
        // <15=> 15 (P0.15)
        // <16=> 16 (P0.16)
        // <17=> 17 (P0.17)
        // <18=> 18 (P0.18)
        // <19=> 19 (P0.19)
        // <20=> 20 (P0.20)
        // <21=> 21 (P0.21)
        // <22=> 22 (P0.22)
        // <23=> 23 (P0.23)
        // <24=> 24 (P0.24)
        // <25=> 25 (P0.25)
        // <26=> 26 (P0.26)
        // <27=> 27 (P0.27)
        // <28=> 28 (P0.28)
        // <29=> 29 (P0.29)
        // <30=> 30 (P0.30)
        // <31=> 31 (P0.31)
        // <4294967295=> Not connected

        #ifndef PN532_RESET
        #define PN532_RESET 1 // SX_IOEXT_1
        //#define PN532_RESET 3 //ANA_DIG1
        #endif

        // <o> PN532_CONFIG_SCL  - Pin number

        // <0=> 0 (P0.0)
        // <1=> 1 (P0.1)
        // <2=> 2 (P0.2)
        // <3=> 3 (P0.3)
        // <4=> 4 (P0.4)
        // <5=> 5 (P0.5)
        // <6=> 6 (P0.6)
        // <7=> 7 (P0.7)
        // <8=> 8 (P0.8)
        // <9=> 9 (P0.9)
        // <10=> 10 (P0.10)
        // <11=> 11 (P0.11)
        // <12=> 12 (P0.12)
        // <13=> 13 (P0.13)
        // <14=> 14 (P0.14)
        // <15=> 15 (P0.15)
        // <16=> 16 (P0.16)
        // <17=> 17 (P0.17)
        // <18=> 18 (P0.18)
        // <19=> 19 (P0.19)
        // <20=> 20 (P0.20)
        // <21=> 21 (P0.21)
        // <22=> 22 (P0.22)
        // <23=> 23 (P0.23)
        // <24=> 24 (P0.24)
        // <25=> 25 (P0.25)
        // <26=> 26 (P0.26)
        // <27=> 27 (P0.27)
        // <28=> 28 (P0.28)
        // <29=> 29 (P0.29)
        // <30=> 30 (P0.30)
        // <31=> 31 (P0.31)
        // <4294967295=> Not connected

        #ifndef PN532_CONFIG_SCL
        #define PN532_CONFIG_SCL 15   //thingy ext scl
        #endif

        // <o> PN532_CONFIG_SDA  - Pin number

        // <0=> 0 (P0.0)
        // <1=> 1 (P0.1)
        // <2=> 2 (P0.2)
        // <3=> 3 (P0.3)
        // <4=> 4 (P0.4)
        // <5=> 5 (P0.5)
        // <6=> 6 (P0.6)
        // <7=> 7 (P0.7)
        // <8=> 8 (P0.8)
        // <9=> 9 (P0.9)
        // <10=> 10 (P0.10)
        // <11=> 11 (P0.11)
        // <12=> 12 (P0.12)
        // <13=> 13 (P0.13)
        // <14=> 14 (P0.14)
        // <15=> 15 (P0.15)
        // <16=> 16 (P0.16)
        // <17=> 17 (P0.17)
        // <18=> 18 (P0.18)
        // <19=> 19 (P0.19)
        // <20=> 20 (P0.20)
        // <21=> 21 (P0.21)
        // <22=> 22 (P0.22)
        // <23=> 23 (P0.23)
        // <24=> 24 (P0.24)
        // <25=> 25 (P0.25)
        // <26=> 26 (P0.26)
        // <27=> 27 (P0.27)
        // <28=> 28 (P0.28)
        // <29=> 29 (P0.29)
        // <30=> 30 (P0.30)
        // <31=> 31 (P0.31)
        // <4294967295=> Not connected

        #ifndef PN532_CONFIG_SDA
        #define PN532_CONFIG_SDA 14   //thingy ext sda
        #endif

        // <o> PN532_CONFIG_TWI_INSTANCE  - TWI instance to be used

        // <0=> 0
        // <1=> 1
        // <2=> 2

        #ifndef PN532_CONFIG_TWI_INSTANCE
        #define PN532_CONFIG_TWI_INSTANCE 0
        #endif

        // <e> ADAFRUIT_PN532_LOG_ENABLED - Enables logging in the module.
        //==========================================================
        #ifndef ADAFRUIT_PN532_LOG_ENABLED
        #define ADAFRUIT_PN532_LOG_ENABLED 1
        #endif
        #if  ADAFRUIT_PN532_LOG_ENABLED
        // <o> ADAFRUIT_PN532_LOG_LEVEL  - Default Severity level

        // <0=> Off
        // <1=> Error
        // <2=> Warning
        // <3=> Info
        // <4=> Debug

        #ifndef ADAFRUIT_PN532_LOG_LEVEL
        #define ADAFRUIT_PN532_LOG_LEVEL 0
        #endif

        // <o> ADAFRUIT_PN532_INFO_COLOR  - ANSI escape code prefix.

        // <0=> Default
        // <1=> Black
        // <2=> Red
        // <3=> Green
        // <4=> Yellow
        // <5=> Blue
        // <6=> Magenta
        // <7=> Cyan
        // <8=> White

        #ifndef ADAFRUIT_PN532_INFO_COLOR
        #define ADAFRUIT_PN532_INFO_COLOR 0
        #endif

        #endif //ADAFRUIT_PN532_LOG_ENABLED
        // </e>

        // <o> PN532_PACKBUFF_SIZE - Size of the buffer used for sending commands and storing responses.
        #ifndef PN532_PACKBUFF_SIZE
        #define PN532_PACKBUFF_SIZE 256
        #endif

        #endif //ADAFRUIT_PN532_ENABLED
        // </e>

```