Martin Gulbrandsen Aalien

Development and Optimization of a Contact Tracing Wearable

Master's thesis in Electronic Systems Design
Supervisor: Carsten Wulff
June 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Martin Gulbrandsen Aalien

Development and Optimization of a Contact Tracing Wearable

Master’s thesis in Electronic Systems Design
Supervisor: Carsten Wulff
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Preface

In 2016, I started studying Electrical Engineering, and I knew right a way that I had found a career path I wanted to proceed. I finished my bachelor's degree with a specialization in Electronics in 2019. After the bachelor's degree, I was eager to learn more before heading off into the working life, so I went for a master's degree in Electronic Systems Design at the Norwegian University of Science and Technology. This master's thesis marks the end of my 2-years long degree. My master's thesis work has been conducted during the COVID-19 pandemic, and it has greatly impacted my choice of work. Hopefully, I have made a contribution that can help reduce the spread of infectious diseases, like COVID-19.

This thesis is structured for two-sided print, meaning that all the chapters starts at odd page numbers. The software developed for this thesis is included in the attachments on Inspera.

Acknowledgment

I would first like to thank my main supervisor Carsten Wulff for his invaluable help. He has always been available for questions and discussions which have brought my work to a higher level.

I would also like to thank Nordic Semiconductor for lending me the equipment I needed for my work.

In addition, I would like to thank my friend, Martin Falang, for peer-reviewing my code during development, implementing the GAENS cryptography library and creating the appearance model of the wearable.

Problem Description

At the time of writing this thesis, the whole world is affected by the Coronavirus Disease 2019 pandemic. Contact tracing is one of the measures taken to reduce the spread of infection. Digital Exposure Notification Systems (ENSs) makes contact tracing more efficient and accurate than a manual approach. Most ENSs require individuals to own and wear a smartphone. However, many people do not own, know how to use, or are not able to carry a smartphone around at all times.

The aim of this thesis is to develop and optimize a small, specialized, non-internet-connected wearable for contact tracing. The wearable should have a battery life of several months on a single coin cell battery. The software shall be based on Google/Apple Exposure Notification System, and the Wearable Exposure Notification Service. Such a solution can make it easier for more people to take part in the existing ENSs, and reduce the spread of infection.

Abstract

The wearables market for healthcare and activity monitoring is rapidly evolving. Lately, it has gained attention as an alternative method to perform contact tracing to help reduce the spread of Coronavirus Disease 2019 (COVID-19) [1]. Several governments and companies have already developed smartphone applications for doing contact tracing, so-called Exposure Notification Systems (ENSs) [2]. However, a large portion of the population does not own or use a smartphone.

In this thesis a low-power, specialized, non-internet-connected contact tracing wearable is introduced. The wearable is intended for adaptation into the already established ENSs. It is based on the Google/Apple Exposure Notification System, which is used in several European countries [2, 3]. To make it possible for the non-internet-connected wearable to participate in an ENS, it will support the Wearable Exposure Notification Service, which Bluetooth Special Interest Group released a preliminary specification draft of in December 2020 [1].

An engineering prototype of the wearable was designed and optimized, based on a Proof of Concept (PoC) prototype. The PoC, without optimizations, achieved an average current consumption of approximately 532µA. The optimization analysis of the engineering prototype indicates that it should be possible to achieve an average current of 42.17µA, with the optimizations suggested in this thesis. By powering the wearable with a CR2032 coin cell battery, one can expect a battery life of about 197 days.

A Printed Circuit Board (PCB) for the engineering prototype was designed with a circular shape. It has a diameter of 26mm, and a height of approximately 8mm with its components. The major components on the board are a nRF52833 System-on-Chip, a 32Mb external flash memory, an accelerometer and a battery holder for a CR2032 coin cell battery. A casing for the PCB was also designed in the shape of a circular tag, and is 28.5mm in diameter, and has a height of 11mm.

The software and hardware developed for the low-power, non-internet-connected wearable contributes to realizing a wearable that can participate in already existing ENSs and reduce the spread of infectious diseases like COVID-19.

Sammendrag

Markedet for kroppsbårne elektroniske enheter for helse- og aktivitets-monitorering er i kraftig utvikling. Nylig har det fått oppmerksomhet som en alternativ måte å gjennomføre smittesporing på for å redusere spredningen av Coronavirus Disease 2019 (COVID-19) [1]. Flere lands myndigheter og firmaer har allerede utviklet applikasjoner til smarttelefoner for smittesporing [2]. Et problem med denne løsningen er at en stor andel av befolkningen ikke eier eller bruker smarttelefoner.

I denne masteroppgaven introduseres en spesialisert, ikke-internett-tilkoblet, kroppsbåren elektronisk enhet for smittesporing med lavt strømforbruk. Den kroppsbårne elektroniske enheten er ment for å bli tatt i bruk i de allerede etablerte systemene for eksponeringsvarsling. Enheten er basert på Google/Apple Exposure Notification System som er systemet for eksponeringsvarsling som er tatt i bruk av flere europeiske land [2, 3]. For å gjøre det mulig for en ikke-internett-tilkoblet, kroppsbåren elektronisk enhet å delta i et system for eksponeringsvarsling vil enheten støtte Wearable Exposure Notification Service, som Bluetooth Special Interest Group lanserte et foreløpig utkast av spesifikasjonene av i desember i 2020 [1].

En teknisk prototype av den kroppsbårne elektroniske enheten ble designet og optimalisert, basert på en Proof of Concept (PoC)-prototype. PoC-prototypen oppnådde et gjennomsnittlig strømtrekk på tilnærmet 532µA, uten optimaliseringer. Analysene av optimaliseringene for den tekniske prototypen, som er foreslått i denne masteroppgaven, indikerer at det skal være mulig å oppnå et gjennomsnittlig strømtrekk på 42.17µA. Ved å drive enheten med et CR2032 knappebatteri kan man forvente å oppnå en batterilevetid på rundt 197 dager.

Kretskortet for den tekniske prototypen ble designet med en sirkulær form, og har en diameter på 26mm og en høyde på cirka 8mm medregnet komponenter. Den kroppsbårne elektroniske enheten består av en nRF52833 Bluetooth-modul, et 32Mb eksternt flashminne, et akselerometer og en batteriholder for et CR2032 knappebatteri. En innkapsling med form av en tag ble designet, og den har en diameter på 28.5mm og en høyde på 11mm.

Program- og maskinvaren utviklet for den spesialiserte, ikke-internett-tilkoblede, kroppsbårne elektroniske enheten med lavt strømforbruk, bidrar til å realisere en kroppsbåren enhet som kan delta i allerede eksisterende system for eksponeringsvarsling, og redusere spredningen av smitte av sykdommer som COVID-19.

Contents

Figures

Tables

Code Listings

Acronyms

AEM Associated Encrypted Metadata.

AEMK Associated Encrypted Metadata Key.

AES Advanced Encryption Standard.

API Application Programming Interface.

BAS Battery Service.

BLE Bluetooth Low Energy.

Bluetooth SIG Bluetooth Special Interest Group.

BMS Bond Management Service.

COVID-19 Coronavirus Disease 2019.

CRNG Cryptographic Random Number Generator.

CS Chip Select.

DIS Device Information Service.

DTS Device Time Service.

ENS Exposure Notification System.

FEP Functional End Point.

GAENS Google/Apple Exposure Notification System.

HKDF Hashed Message Authentication Code (HMAC)-based Key Derivation Function.

LED Light Emitting Diode.

MISO Master In Slave Out.

MOSI Master Out Slave In.

NFC Near Field Communication.

PCB Printed Circuit Board.

PI Proximity Identifier.

PoC Proof of Concept.

PPK Power Profiler Kit.

RPI Rolling Proximity Identifier.

RPIK Rolling Proximity Identifier Key.

RSSI Received Signal Strength Indicator.

RTC Real Time Clock.

RTOS Real-Time Operating System.

SARS-CoV-2 Severe Acute Respiratory Syndrome Coronavirus 2.

SCLK Serial Clock.

SoC System-on-Chip.

SPI Serial Peripheral Interface.

SWD Serial Wire Debug.

TEK Temporary Exposure Key.

UART Universal Asynchronous Receiver-Transmitter.

USB Universal Serial Bus.

UUID Universally Unique Identifier.

WENS Wearable Exposure Notification Service.

Chapter 1

Introduction

In December 2019, in Wuhan City, China, a "Pneumonia of unknown cause" was discovered [4]. The disease was later named Coronavirus Disease 2019 (COVID-19) and is caused by a virus named Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2). The outbreak of the virus resulted in a global pandemic. The highly contagious virus proved to cause severe illness to a significant portion of the infected individuals. It is of critical importance to slow the spread of the virus, as the consequences can be fatal. To slow the spread of the virus, governments have implemented different measures, such as the use of face masks, lockdowns, close-contact restrictions and contact tracing.

The wearables market for healthcare and activity monitoring is rapidly evolving, and has gained attention lately as an alternative method to do contact tracing to help reduce the spread of COVID-19. Several governments and companies have already developed smartphone applications for doing contact tracing [2]. However, many people do not own or use a smartphone. This is a limitation of smartphone applications as a digital contract tracing technology, because of the lack of coverage. By including wearables to the digital contact tracing solutions that is already in use, one would increase the coverage and efficiency of the systems.

Most contact tracing smartphone applications use Bluetooth. This is done by devices broadcasting their IDs, and scanning to look for other devices nearby. Google and Apple were early in standardizing how an Exposure Notification System (ENS) should operate by introducing their Google/Apple Exposure Notification System (GAENS) [3]. The Norwegian Smittestopp app is based on GAENS, and several other European countries have adopted the standard as well [2].

In December 2020 Bluetooth Special Interest Group (Bluetooth SIG) published a draft describing a Wearable Exposure Notification Service (WENS) [1]. The draft describes a Bluetooth service that makes it possible for wearables to participate in the already established ENSs. It is reasonable to believe that this Bluetooth service draft will eventually be a part of the existing ENSs.

By combining the use of GAENS and WENS into a small wearable device, even more people can participate in the already established ENSs.

1.1 Specialization Project

Prior to this master's thesis, a Proof of Concept (PoC) prototype of a contact tracing wearable was developed in a specialization project named "*A Contact Tracing Wearable*". The project presented a low-cost, specialized, non-internet-connected, contact tracing wearable using Bluetooth Low Energy (BLE). The schematics for the PoC prototype can be seen in Appendix B, and Figure 1.1 shows a picture of it. The PoC prototype was made with the purpose of helping to track and minimize the spread of COVID-19 without having to use a smartphone. This is similar to what was introduced with the publication of WENS [1].

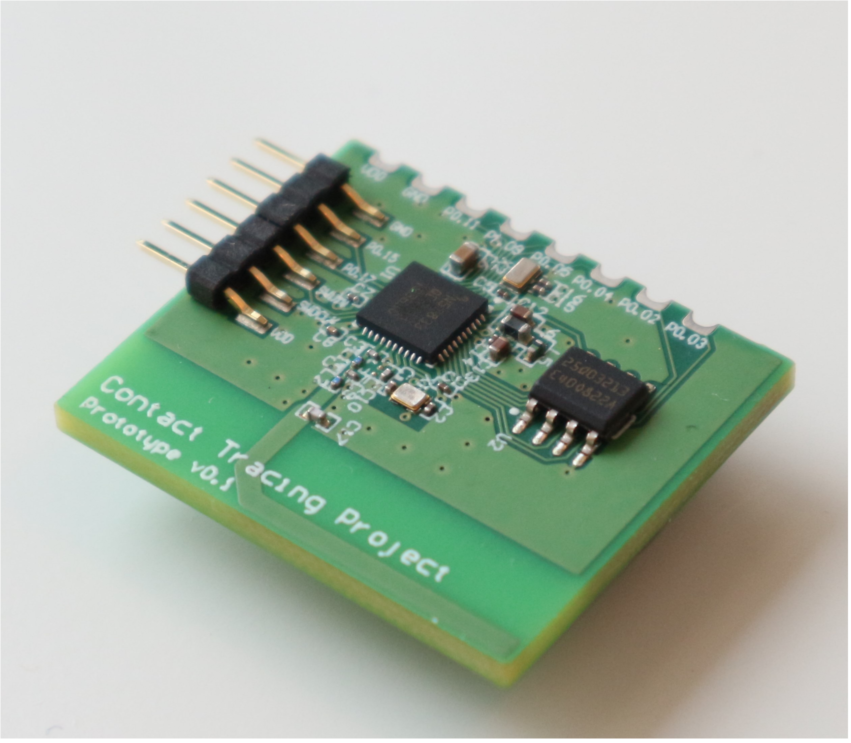


Figure 1.1: The PoC prototype that was designed prior to this master's thesis.

The PoC prototype was meant to be used on an institutional basis, and function without any use of a smartphone at all. Instead of using a smartphone to do the contact exposure risk calculations, the PoC prototype system used internet-connected hubs. However, after the WENS was introduced, it opened up the possibility to design a wearable for everybody, not only on an institutional basis. Even though the PoC prototype system works differently to GAENS and WENS, there are some similarities, and the PoC prototype wearable can be used for a solution based on GAENS and WENS, without any modifications.

The PoC prototype wearable uses the nrf52833 BLE SoC from Nordic Semiconductor. It also has an external flash memory chip and a battery holder for an CR2032 battery. The PoC prototype has castellated holes and a header for debugging purposes. The length of the PoC prototype is 29.5mm, the width is 25.2mm and the height is approximately 10mm. Making it small enough to be embedded into a wearable. Nevertheless, there are several things that can be optimized further.

1.2 Scope and Limitations

The objective of this thesis consists of several tasks. One of which is to develop software based on the specification documentation of GAENS and WENS. It should be verified that the software works according to the specification. Having the software working as intended, the current consumption of a minimum viable product solution should be estimated and/or measured. The estimation and measurements of the current consumption will be used to determine the main contributing processes to the current consumption.

After the main contributing processes to the current consumption have been found, methods to reduce the average current consumption should be investigated. The goal is to achieve a realistic battery life of the wearable so that it lasts for several months on a single CR2032 coin cell battery.

The findings from the optimization methods and previous work, will be used to develop an optimized version of the PoC prototype, which will be referred to as an engineering prototype. The engineering prototype should have the main functionality of a contact tracing wearable and be visually representative for a desired final product.

There are some limitations to the tasks. The full WENS specification is not official yet, so an implementation of it will not be complete. It should however be implemented to a point where it can take part in the other aspects of the study. Another limitation of the scope of the work is that the engineering prototype hardware will be designed, but not manufactured. The manufacturing of it is planned in future work.

1.3 Key Contributions

This thesis presents a low-power, specialized, non-internet-connected contact tracing wearable solution that has the potential to participate in already established ENSs to help reduce the spread of COVID-19. The key contributions of this thesis are:

- A software solution based on GAENS and WENS was developed for Nordic Semiconductors BLE System-on-Chip (SoC)s.
- A Printed Circuit Board (PCB) for the contact tracing wearable was designed. Together with the components, the circular PCB had a diameter of 26mm and a height of 8mm.
- An engineering prototype of a wearable suitable for contact tracing was introduced. The wearable is shaped as a tag, and has a diameter of 28.5mm, a height of 11mm, and a small mount hole.
- Optimization methods were presented, which decreases the current consumption from approximately 532µA to 42.17µA. Giving the wearable an estimated lifetime of 197 days on a CR2032 coin cell battery.

1.4 Organization of the Thesis

In Chapter 2 the background, information about the material that will be used in the development and optimization of the contact tracing wearable is presented. The background chapter is followed by a description of the development and optimization methodology, in Chapter 3. In Chapter 4 the implementation of the wearable's hardware and software are described. In Chapter 5 the storage requirements of the wearable are determined. This is followed by current consumption estimations and measurements in Chapter 6, and optimizations in Chapter 7. After looking into the possible optimization strategies, the hardware design for the engineering prototype is explained in Chapter 8. The results are presented and discussed in Chapter 9. In Chapter 10, tasks that need to be investigated in the future are listed. Lastly, the conclusion of the thesis is presented in Chapter 11.

Chapter 2

Background

In this chapter, some key background topics for this thesis is covered. Such as the principles of infection, why contact tracing is important, how contact tracing works, and how contact tracing is implemented. In addition, a few key topics is introduced, like the relevant BLE services, Zephyr, empirical facts on coin cell batteries and a definition of the phases of product prototyping.

Throughout this thesis, there will be made several references to Bluetooth-specific terminology. Background theory about BLE was covered in the specialization project, and is included in Appendix A.

2.1 Spread of Infection

An individual infected with COVID-19 can spread the disease to other individuals through a number of different transmission routes [5]. The disease is most commonly spread during close contact. Infection mainly occurs through exposure to respiratory droplets when an individual is in close contact with someone who is infected with COVID-19. When infected individuals talk, sing, breath, sneeze, etc., they produce respiratory droplets containing the virus which will be exposed to other individuals in close contact. The risk of infection increases with the time people spend in close contact with infected individuals. This is why social distancing is crucial to reduce the spread.

An individual can be infectious even before the individual develops symptoms [6]. Meaning it is important to identify and isolate infected individuals as soon possible. By linking the close contacts, one can notify people who have a risk of being infected, so that they can quarantine themselves to prevent further transmission of the disease.

2.2 Contact Tracing

Contact tracing is the process of identifying and managing the individuals that have been exposed to a disease to avoid onward transmission [7]. Individuals exposed to the disease are typically quarantined or isolated for the maximum incubation period from the date of the most recent exposure.

Contact tracing can be performed in a number of different ways. At the beginning of the COVID-19 pandemic, the contact tracing was performed manually by the authorities. This involves contacting the infected individuals and asking for their close contacts and whereabouts in the last couple of days. This technique is error-prone, as it is difficult to remember all the persons one meets, and the exact time one has been to a location. The technique is also time demanding, meaning it takes time to determine an infected individual’s close contacts, and contact them. With more and more infected individuals, this process could grow at a tremendous rate, putting an unbearable workload on the individuals performing the contact tracing.

To make contact tracing easier than manually contacting those who are at risk of being infected, there have been launched digital contact tracing solutions from local companies and governments all over the world. This automates the contact tracing process and makes it significantly faster than doing manual contact tracing. It makes it possible to notify individuals that have a risk of being infected earlier, which could reduce the spread of the disease.

2.2.1 Centralized Versus Decentralized Architectures

Two main architectures have been designed for wireless contact tracing applications, centralized and decentralized architectures [8]. With centralized architectures the main information, such as temporary IDs and timestamps, is stored and processed on a central server. Usually, the centralized architectures perform the risk analysis (calculating the probability of infection) on the central server. While with a decentralized architecture, the information is stored and processed on the user’s devices. The decentralized solution only contacts a central server to download the temporary IDs or report an infection. The decentralized architectures, in contrast to the centralized architectures, perform the risk analysis on the user’s devices. Meaning that the central server on decentralized architectures does not store any data about the user’s contacts.

2.3 Exposure Notification System

Digital contact tracing technologies have two main tasks, logging encounters and reporting infections. Exposure Notification Systems (ENSs) defines how to do the logging of encounters, while the infection reporting is being delegated to the individual implementations. In Norway, and several other European countries, the Google/Apple Exposure Notification System (GAENS) protocol is being used [2].

2.4 Google/Apple Exposure Notification System

GAENS is a framework and protocol specification on how to facilitate digital contact tracing developed by Google and Apple [3]. The protocol is a decentralized reporting-based protocol that uses BLE and privacy-preserving cryptography.

2.4.1 How it Works

If a smartphone has an application for ENS that is using GAENS, the device will generate random IDs for your phone. These random IDs change about every 15 minutes, so that they cannot be used to identify a person or the person's location. The ENS application will run in the background and advertise the random IDs, so-called Rolling Proximity Identifiers (RPIs), and scan for other devices nearby that are also advertising their respective RPIs. The received RPIs is stored on the device. The device will periodically check all the random IDs that are associated with COVID-19 cases and compare them to their own list. If there is a match, the user gets a notification with instructions from the public health authority about what to do. Typically, this means testing and isolation.

2.4.2 Cryptography Specification

The GAENS cryptography specification addresses how the ENS data should be encrypted before it is sent over BLE [9]. To ensure the privacy of the users of the devices, the GAENS uses pseudorandom identifiers, called RPIs. Each of the RPI is derived from a Temporary Exposure Key (TEK) and a discretized representation of time. The RPI changes a random time interval larger than 10 minutes and less than 20 minutes. It is changed at the same time as the Bluetooth randomized address to prevent wireless tracking. In addition to the RPI, there are also some data about the wearable named Associated Encrypted Metadata (AEM) that is to be sent together with the RPI. The AEM can be decrypted later when the user test positive.

The cryptography of GAENS is dependent on four external functions. The first external function is a Cryptographic Random Number Generator (CRNG), which is essentially a function where one inputs the size of the desired cryptographic number you want to retrieve, and it returns a cryptographic random number of the same size. The CRNG is a random generator with properties that makes it suitable for use in cryptography. The function is used for generating the 16-byte TEK.

Another one of the external cryptographic functions is called Hashed Message Authentication Code (HMAC)-based Key Derivation Function (HKDF). The purpose of HKDF is to derive an encryption key from a pass phrase. The pass phrase in the GAENS specification is the TEK. The HKDF function is used to generate the Rolling Proximity Identifier Key (RPIK) and Associated Encrypted Metadata Key (AEMK).

The RPIK and a discretized representation of time (in 10-minute intervals), $ENIntervalNumber$, is used as input to a function called Advanced Encryption Standard (AES) to encrypt the data and generate the RPI. The RPI, metadata and the AEMK is then used as input to an AES in Counter Mode (AES-CTR) to generate the AEM to be sent over BLE together with the RPI.

To better illustrate how the cryptography is organized, the operations are illustrated in Figure 2.1. The figure is based on the GAENS cryptography specification [9].

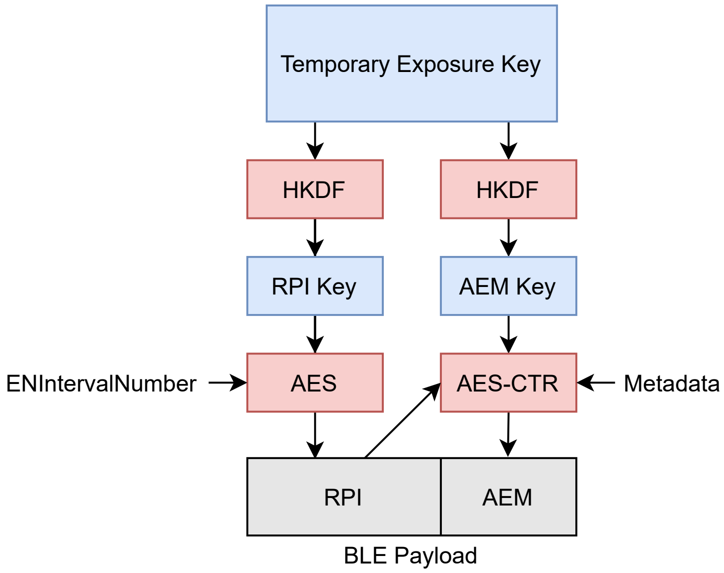


Figure 2.1: An illustration of how the cryptography specification should work [9].

In the following sections the cryptography implementation, and the generation of the RPI and the AEM is explained.

ENIntervalNumber

In the GAENS protocol, the time is discretized in 10-minute intervals that are enumerated starting from Unix Epoch Time. The number of the interval is called $ENIntervalNumber$, and is encoded as a 32-bit unsigned little-endian value. Equation (2.1) gives the function for getting a number for each 10-minute time window, where $n_{interval}$ is equal to 10.

$$ENIntervalNumber(Timestamp) = \frac{Timestamp}{60 \cdot n_{interval}} \tag{2.1}$$

TEKRollingPeriod

The $TEKRollingPeriod$ is the duration of which an Temporary Exposure Key (TEK) is valid (in multiples of 10 minutes). The GAENS documentation specifies that the key validity should be 24 hours, making $TEKRollingPeriod$ equal to 144.

Temporary Exposure Key

When the first TEK is generated, it is associated with an $ENIntervalNumber$, i, which is the time from which the key is valid. The $ENIntervalNumber$, i, is defined by Equation (2.2). The value is to be stored together with the TEK.

$$i = \left\lfloor \frac{ENIntervalNumber(Time\ at\ Key\ Generation)}{TEKRollingPeriod} \right\rfloor \cdot TEKRollingPeriod \tag{2.2}$$

The TEK is generated from the CRNG, and is 16 bytes long. This is done as shown in Equation (2.3). Using 16 bytes long TEK makes the risk for false positives low. Since the TEK roll daily, the device needs to store one key per day.

$$tek_i = CRNG(16) \tag{2.3}$$

Rolling Proximity Identifier Key

A Rolling Proximity Identifier Key (RPIK) needs to be generated before the RPI, since the key is used for generation of the RPI. The RPIK is generated using HKDF as shown in Equation (2.4).

$$RPIK_i = HKDF(tek_i, NULL, UTF8(''EN - RPIK''), 16) \tag{2.4}$$

Rolling Proximity Identifier

When the RPIK is generated one can derive a RPI as shown in Equation (2.5).

$$RPI_{i,j} = AES_{128}(RPIK_i, PaddedData_j) \tag{2.5}$$

Where:

- j is the Unix Epoch Time at the moment the roll occurs
- $PaddedData_j$ is a 16 bytes long sequence constructed as:
  - $PaddedData_j[0...5]$ = UTF8("EN-RPI")
  - $PaddedData_j[6...11]$ = 0x000000000000
  - $PaddedData_j[12...15] = ENIntervalNumber(j)$

Associated Encrypted Metadata Key

The RPI is not the only data that needs to be broadcasted. The BLE packets need to include some Associated Encrypted Metadata (AEM) according to the GAENS specification. Before encrypting the metadata, a key is needed. The Associated Encrypted Metadata Key (AEMK) is generated by Equation (2.6).

$$AEMK_i = HKDF(tek_i, NULL, UTF8(''EN - AEMK''), 16) \tag{2.6}$$

Associated Encrypted Metadata

The AEM is encrypted as shown in Equation (2.7).

$$AEM_{i,j} = AES_{128} - CTR(AEMK_i, RPI_{i,j}, Metadata) \tag{2.7}$$

The RPI and AEM is used in the BLE broadcast packets. In this way, the contact tracing can be done while preserving the user's privacy.

2.4.3 Positive Diagnosis

If a participant in a GAENS test positive, the person can register it. When the positive diagnosis is registered, a set of TEKs and their respective $ENIntervalNumber$, i, is uploaded to a diagnosis server. These TEKs are the keys the user has used when the user could have been exposing others. This is typically the TEKs from the past 14 days. The set of TEKs is referred to as diagnosis keys. Otherwise, as long as the user does not test positive, the keys will never leave the device.

The diagnosis server distributes the diagnosis keys from all the users that have tested positive to all the other devices participating in the ENS. The participants in the ENS periodically fetch the list of diagnosis keys from the diagnosis server to see if they have a match with the RPIs they have received.

2.4.4 Bluetooth Specification

The GAENS Bluetooth specification provides the details on the BLE behavior of the exposure notification service [3]. The service is registered with a 16 bit Universally Unique Identifier (UUID) by the Bluetooth SIG. Devices participating in the ENS will broadcast and scan for the UUID. The service data shall contain the RPI and the AEM.

The structure of the BLE advertising packet can be seen in Table 2.1. The packet is 31 bytes long, which is the maximum length of an BLE advertising packet. It has three types of data, namely flags, complete 16-bit service UUID and service data. The flags section specifies general discovery mode. The complete 16-bit service UUID section of the packet includes the GAENS UUID, 0xFD6F. In the service data section, the data relevant for the GAENS is located. This includes the RPI and the AEM. The AEM is structured as follows:

- Byte 0: Versioning
  - Bits 7:6: Major version (01)
  - Bits 5:4: Minor version (00)
  - Bits 3:0: Reserved for future use
- Byte 1 - Transmit power level
  - This is the measured radiated transmit power of the BLE packets. The field ranges from -127dBm to +127dBm.
- Byte 2 - Reserved for future use
- Byte 3 - Reserved for future use

Flags			**Complete 16-bit Service UUID**			**Service Data**				
Length	**Type**	**Flags**	**Length**	**Type**	**UUID**	**Length**	**Type**	**Service data**		
0x02	0x01	0x1A	0x03	0x03	0xFD6F	0x17	0x16	0xFD6F	RPI	AEM

Table 2.1: A complete advertising packet according to the GAENS specification.

According to the GAENS Bluetooth specification, the recommended broadcasting interval is 200-270 milliseconds. The advertising packets shall be non-connectable.

As mentioned, the RPI and the AEM have to rotate at a random time interval. This goes for the Bluetooth address too. This is due to that all the information is device-dependent, so to avoid any privacy issues they have to rotate synchronously. If they are not changed synchronously they can be linked, and that would make it possible to track the device. The rotation interval should be a random value that is greater than 10 minutes and less than 20 minutes. Making the average rotation period 15 minutes.

The device will also have to scan for other devices broadcasting the GAENS UUID. The discovered packets will be stored on the device until they expire. Along with the discovered packets, the RSSI value has to be stored with their respective packets. This is used for distance estimation between the devices, and is used in the exposure risk calculations. There are no exact scanning interval and window that are specified by the documentation, but the device should scan often enough and long enough to discover ENS advertisements nearby within 5 minutes.

2.5 Wearable Exposure Notification Service

Wearable Exposure Notification Service (WENS) specification draft was released 03.12.2020 by Bluetooth Special Interest Group (Bluetooth SIG). The specification defines "*a standardized method to enable a non-Internet-connected wearable device to operate in a manner complementary with one or more deployed Client-based Exposure Notification Systems (ENSs), therefore enabling significantly more individuals to participate in an ENS. The methods defined in this specification are applicable for the containment of a wide variety of infections, including SARS-CoV-2.*" [1].

Although the specification is a working draft, the potential in the Bluetooth service is substantial. Given the situation the world finds itself in by the time of writing, it is a good chance that it will be taken into use. It would make it possible for many more individuals to participate in the already established ENSs. Meaning that the coverage will increase.

The WENS specification states that it requires two other services, the Device Time Service (DTS) and the Device Information Service (DIS). The DTS is used for keeping track of time, and setting the time if it has drifted. While the DIS provides information about the device itself. All the three services together make up the profile that can be seen in Figure 2.2. In addition to those services, the specification also strictly recommends Battery Service (BAS) and Bond Management Service (BMS), but since they are not mandatory, they will not be discussed further.

The wearable have to broadcast connectable advertising packets for other devices to connect to it and access the services. The structure of each connectable advertising packet is shown in Table 2.2.

Flags			**Complete 16-bit Service UUID**		
Length	**Type**	**Flags**	**Length**	**Type**	**Service data**
0x02	0x01	0x06	0x03	0x03	0xFF00

Table 2.2: A table showing the structure of a connectable WENS advertising packet.

WENS has six mandatory characteristics, and two characteristics where one has to select at least one of them, making it a total of seven mandatory characteristics. The two characteristics that one can choose to support are the Temporary Key List characteristic and the ENS Advertisement List characteristic. Since the wearable will be capable of generating its own PIs, the Temporary Key List characteristic will be used, and the ENS Advertisement List characteristic will not be further discussed.

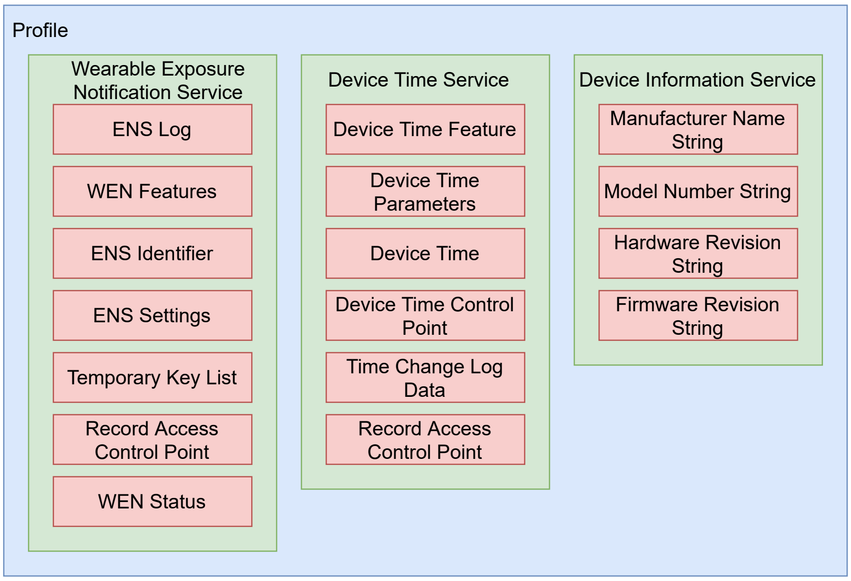


Figure 2.2: The Bluetooth services and their respective characteristics required by WENS.

2.5.1 Characteristics

The seven characteristics of the WENS all have their different purposes. In Table 2.3 the characteristics are described.

Characteristic	Description
ENS Log	The ENS Log characteristic provides a mechanism for transferring a set of ENS Records to a client, such as a smartphone. The characteristic allows segmentation to make it possible to transfer a large amount of data through notifications. It can transfer a full ENS Record, multiple ENS records or a part of an ENS record.
WEN Features	This characteristic provides information about the supported features of the wearable.
ENS Identifier	The ENS Identifier characteristic represents the ENS that is in use by the WENS wearable.
ENS Settings	The ENS Settings characteristic provides information about the settings specific to the ENS. The settings can be changed through this characteristic.
Temporary Key List	The Temporary Key List characteristic is representing a list of timestamps and temporary key pairs. This is for enabling the client to read the recently used temporary keys used to generate PIs and the client can also provide a schedule of temporary keys that the wearable can use.
Record Access Control Point	This characteristic is a control point that is used for basic management functionality for a record database. The characteristic enables functionality like counting records, clearing records and transmitting records based on filtering criteria.
WEN Status	This characteristic shares similarity with a control point, except that this characteristic also can be read by a client.

Table 2.3: All the characteristics of the WENS that will be used, and their corresponding description according to the specification [1].

2.6 Device Information Service

This service provides information about the device, and is one of the two additional services required by WENS [1, 10]. The DIS makes information such as manufacturer, model number, hardware revision and firmware revision, accessible by a client.

2.6.1 Characteristics

The documentation of DIS specifies many characteristics to choose from [10]. Only four of them are mandatory according to the WENS specification [1]. Table 2.4 describes the mandatory characteristics. All the characteristics has mandatory read property so that they can be read by a client.

Characteristic	Description
Manufacturer Name String	This string represents the name of the manufacturer of the device.
Model Number String	The Model Number String characteristic represents the model number that is assigned by the device vendor.
Hardware Revision String	This characteristic represents the hardware revision of the hardware within the device.
Firmware Revision String	This characteristic represents the firmware revision for the firmware within the device.

Table 2.4: All the characteristics of the DIS that will be used, and their corresponding description according to the specification [10].

2.7 Device Time Service

The DTS is also required by WENS. The service exposes the device's Real Time Clock (RTC) to the client for time synchronization.

2.7.1 Characteristics

The DTS documentation specifies six characteristics, and the WENS specification states that all of them should be used [1, 11]. The characteristics are explained in Table 2.5.

Characteristic	Description
Device Time Feature	This characteristic provides a description of the supported features of the device.
Device Time Parameters	The Device Time Parameters characteristic is used for revealing the device's behavioral thresholds and its capabilities.
Device Time	This characteristic is the device's time, and is used for indicating and reading the device time. The characteristic also reveals the time status.
Device Time Control Point	This control point characteristic is used for executing supported procedures on the server.
Time Change Log Data	This characteristic is used to send Time Change Log Data records.
Record Access Control Point	This characteristic is used to retrieve Time Change Log Data, and can be used to filter out what parts of the records one would want to retrieve.

Table 2.5: All the characteristics of the DTS that will be used, and their corresponding description according to the specification [11].

2.8 Zephyr

On an embedded platform, it is common to program directly on the microcontroller unit without any operating system. However, there do exist Real-Time Operating System (RTOS), like Zephyr, that provide a low-level operating system [12]. This allows the developer to focus on the actual features, and reuse existing open source code. Zephyr is a RTOS for resource-constrained, connected and embedded devices. It is a Linux Foundation project and supports more than 200 boards. The RTOS is written in C. Nordic Semiconductor is one of the contributors to the project, so the Zephyr RTOS supports their BLE SoCs. The RTOS includes relevant source code, like a BLE API, $mbedTLS$ cryptography library and power management features.

2.9 CR2032 Coin Cell Battery

The CR2032 coin cell battery is one of the most common coin cell batteries, and is often used to power small electronics devices. In 2011 an investigation was performed by Energizer and Nordic Semiconductor for estimating the battery capacity of a CR2032 coin cell battery for wireless applications [13]. The rest of this section is based on the findings from their investigation called, "High pulse drain impact on CR2032 coin cell battery capacity".

The basic equation for battery life is shown in Equation (2.8). However, in a real-life implementation it is not as simple as that. The actual capacity of a battery is dependent on how it is used. A battery like CR2032 is rated by draining a small stable current of about 200µA. The capacity changes when pulses of high peak currents are drawn. When designing an ultra low power wireless solution the focus is often on making sure the average current consumption is low, but focusing only on the average current assumes that the capacity of the battery would be equal for all other conditions, which is not true.

$$Battery\ lifetime\ (h) = \frac{Battery\ capacity\ (mAh)}{Average\ current\ consumption\ (mA)} \tag{2.8}$$

One of the major factors to the batteries' available capacity is the rate of discharge. The higher the rate of discharge is, the smaller the effective capacity of the battery gets. Coin cell batteries are often used for applications which draws a small, stable amount of current, which results in a long battery life. Such applications as traditional watches. Unlike traditional watches, the current in wireless devices is drawn in short bursts of currents in the range of 10-80mA. This will impact the effective capacity of the battery.

The actual capacity for a low-power wireless application such as this has to be tested in the future, but one can expect around 200mAh.

2.10 Three Phases of Prototyping

There are many ways to look at the timeline in product prototyping, and one of them is by dividing the timeline into three phases; alpha phase, beta phase and pilot phase [14]. These three phases will be used to explain the progress of the work in this thesis. Each phase represents a step forward along the product roadmap. What stage you find yourself in along the process requires different tools, methods and decisions. In the next sections the three phases are explained.

2.10.1 Alpha Phase

In the alpha phase, one seeks to answer the question if the product will work or not, and how it will look and be used. In this phase, one typically creates a PoC prototype to find out if the product will work. In addition to the PoC prototype, an appearance model prototype is often created as well. The appearance model prototype often lacks functionality, but is visually representative of the desired final look of the product.

2.10.2 Beta Phase

In the beta phase, the product is becoming fully formed. This includes that the product is functional and visually representative. In the beta phase, one works on improving and refining the product design and functionality based on earlier iterations of the product, like the PoC prototype. There are often created two new versions of the product in this phase; the engineering prototype and the production prototype. They are somewhat different in their purpose. The engineering prototype is a direct successor from the PoC prototype, and it should begin to look like the desired appearance as well. This prototype is meant to be deployed and tested by potential customers for demonstrating the viability of the product in an operational environment.

If the engineering prototype is successful, the next step is to create a production prototype. This is the successor to the engineering prototype. The production prototype is the last prototype that is created before the product is released for mass-production tooling. The product should be fully functional and have a virtually indistinguishable appearance from the final product.

2.10.3 Pilot Phase

The last phase of prototyping is essentially the final product, which is the start of mass production, and is often called the pilot. It is still referred to as a prototype because it needs final product testing, certification, quality reviews and approvals.

Chapter 3

Methodology

In this chapter, the approach to answer the problem description is outlined, and the tools required to develop a solution are presented.

3.1 Approach

Previous to this master's thesis, the author conducted a specialization project on the same topic. As a result of the specialization project, a PoC prototype of the wearable's hardware and software was designed. This can be counted as a part of the alpha phase of the product prototyping. It showed that there is possible to design a small wearable that can be used for contact tracing.

According to the three phases of prototyping from Section 2.10, what is left from the alpha phase is developing the appearance model. As the appearance of the wearable does not affect the software implementation for the engineering prototype, it was decided to do the software implementation and optimization before designing the appearance model.

Implementing the software for the engineering prototype started of by reading the documentation of the GAENS and WENS to get familiar. This was followed by creating a diagram of the modules and public functions that was required, before the actual software was implemented in the Zephyr RTOS environment. The code was documented using Doxygen, and Jira Kanban board was used to keep track of the tasks that needed to be done.

In the specialization project, it was found that the wearable needed an external memory module for storing ENS records. The required storage space was something that needed to be calculated before optimizations. Based on the software implementation, and the specification documents, the storage requirements was determined. This was important because the choice of external memory affects the current consumption of the device.

After having implemented the software, and determined the storage requirements, the current consumption of the PoC prototype was measured when it was programmed with the software. This set the base for the optimizations. It was also necessary to know what processes that contributed the most to the current consumption. This was done by isolating all the processes, and measuring or estimating each of their current contributions.

With the results from the current consumption estimation and measurements, methods for reducing the current consumption was investigated.

After investigating different optimizations methods, and implementing them in software, the appearance model was created. The appearance model marked the end of the alpha phase.

When the appearance model was finished, the beta phase of the product development was entered. The software implementation, optimization methods, and the appearance model laid the foundation for development of the hardware for the engineering prototype.

3.1.1 Tools

To develop and optimize the engineering prototype, several tools were used. For the software development, the Zephyr RTOS environment were used to aid the development. By reusing and modifying existing open source code, it was possible to speed up the development.

The functionality of the software was tested on nRF52833 development kits from Nordic Semiconductor and the PoC prototype to validate that everything works as it should. The BLE communication was monitored using the nRF Connect application from Nordic Semiconductor. With nRF Connect, one can for example see the content and interval of advertising packets in the application. In nRF Connect there are several apps for different purposes. The app that is called "Bluetooth Low Energy" was used for scanning for advertising packets, connecting to devices and interacting with the characteristics and services. The "Power Profiler" app was used together with a Power Profiler Kit (PPK) to measure the current consumption of the development kit. Nordic Semiconductor's "*Online Power Profiler for BLE*" was used to validate the measurements [15]. This was used during the estimation and optimization of the wearable.

For designing the hardware of the engineering prototype, Altium Designer is used. Altium Designer is one of the most comprehensive end-to-end solutions for PCB designers. Altium Designer was used to designing the hardware of the PoC prototype as well, and was proven as a suitable tool to develop such a solution.

Chapter 4

Software Development

In this chapter, the software implementation of a combined solution of GAENS and WENS is presented and verified.

4.1 Implementation

The software for the wearable has to comply with the GAENS specifications from Google and Apple, and the WENS specification from the Bluetooth SIG. The specification documents explain the implementation in detail, but leaves some room for individual variations. A diagram of the different modules and public functions required for the implementation is illustrated in Figure 4.1. It gives an overview that makes it easier to develop the software in a structured manner. The diagram is merely a template to assist the development, and does not give an exact view of how the real implementation will be. The software is implemented in Zephyr RTOS, as it is widely used and supports Nordic Semiconductor's BLE SoCs. The Zephyr development environment is set up following their getting started guide.

In the next section, how the external memory communication is configured is covered. This is followed by an explanation of how the random rotation interval between 10 and 20 minutes is calculated. Both sections explain parts of the implementation that is not covered by the documentation. While in Section 4.1.3 the WENS and GAENS part of the implementation is presented.

4.1.1 External Flash Memory Communication

In Zephyr there are drivers for many products. The flash memory chip on the PoC prototype, N25Q032A13ESC40G, is a NOR flash memory, which Zephyr has a driver for [16]. The library `drivers/flash.h` needs to be included to access the flash memory API. In addition to that, three configuration options need to be defined in the project's configuration files. The three configurations is `CONFIG_FLASH=y`, `CONFIG_SPI=y` and `CONFIG_SPI_NOR=y`. This enables the flash memory API, the SPI and configures the SPI

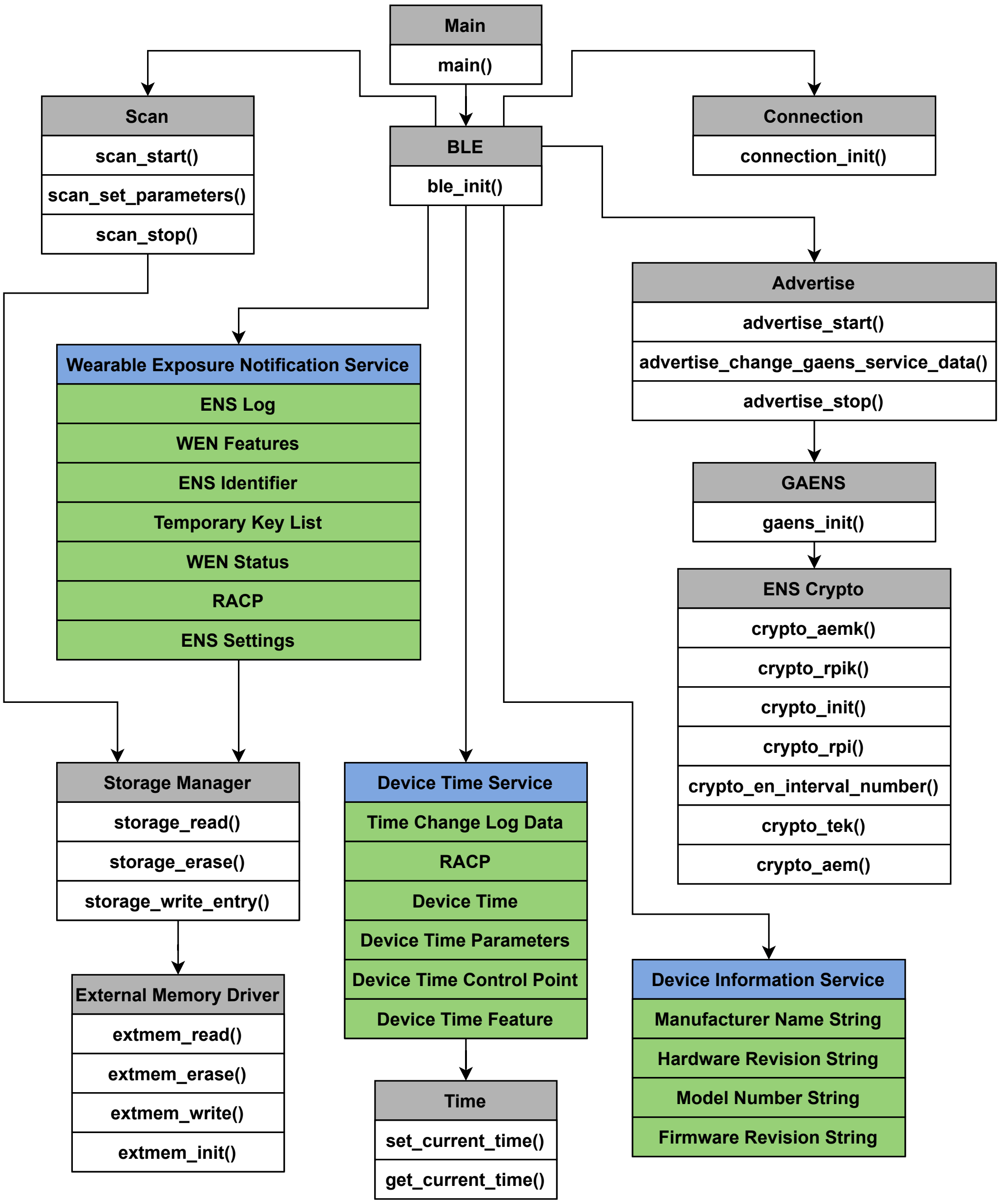


Figure 4.1: A diagram of the modules and public functions in the software implementation.

for a NOR flash memory. To actually communicate with the external memory, one has to overwrite the default hardware describing files to get the pinout and settings set correctly. This goes for the UART communication as well, where the TX pin will be rerouted to one of the pins on the PoC prototype's header for easier accessibility and debugging. The overlay file that is used for configuring the SPI and UART communication is showed in Code listing 4.1.

Code listing 4.1: An overlay file to overwrite the default hardware description of the nRF52833 to get the correct pinout and settings for UART and SPI.

```
/* This file consist of overlays which overwrite the corresponding entries in the
   hardware-describing nrf52833dk_nrf52833.dts (Device Tree Source) file. */


/* An overlay of uart0 to enable logging on P0.15 */
&uart0 {
        compatible = "nordic,nrf-uarte";
        status = "okay";
        current-speed = <115200>;
        tx-pin = <15>;
        rx-pin = <8>;
        rts-pin = <5>;
        cts-pin = <7>;
};

/* An overlay of SPI3 to enable the N25Q32 external memory */
&spi3 {
        status = "okay";
        sck-pin = <31>;
        miso-pin = <29>;
        mosi-pin = <30>;
        cs-gpios = <&arduino_header 2 GPIO_ACTIVE_LOW>; /* <28> */
        N25Q032: N25Q032A13ESC40G@0 {
                compatible = "jedec,spi-nor";
        spi-max-frequency = <80000000>;
                reg = <0>;
                label = "N25Q032";
                jedec-id = [20 BA 16];
                size = <33554432>;
        };
};
```

4.1.2 Random Interval

The Bluetooth address, together with the RPI and AEM, has to change about every 15 minutes. The interval should be larger than 10 minutes and less than 20 minutes. To achieve this without using too much power, one can set up a Real Time Clock (RTC) timer interrupt to trigger when it is time to shift the Bluetooth address, RPI and AEM. This can be done by using the CRNG function to retrieve a random 32-bit number and performing the operation as showed in Equation (4.1).

$$t_r = CRNG(4) \bmod (T_u - T_l - 1) + T_l + 1 \tag{4.1}$$

Where:

- t_r is the random rotation interval larger than 10 minutes and less than 20 minutes.
- $CRNG(4)$ is the cryptographic random number generator specified with 32-bit length.
- T_u is the upper limit of the desired interval (1200 seconds).
- T_l is the lower limit of the desired interval (600 seconds).

4.1.3 Bluetooth and Cryptography Implementation

The rest of the code is mostly Bluetooth or cryptography-related. As both are covered and described in the specification documents provided by Google, Apple and Bluetooth SIG, the code will not be described in detail [1, 3, 9]. The cryptography code implementation was implemented and verified by a friend of mine, Martin Falang, while the rest of the implementation is done by myself. The Doxygen-generated documentation of the code can be viewed in Appendix I. The implementation results in the state diagram that is illustrated in Figure 4.2. The state diagram is a simplified version of the implementation, but it shows the overall basic processes.

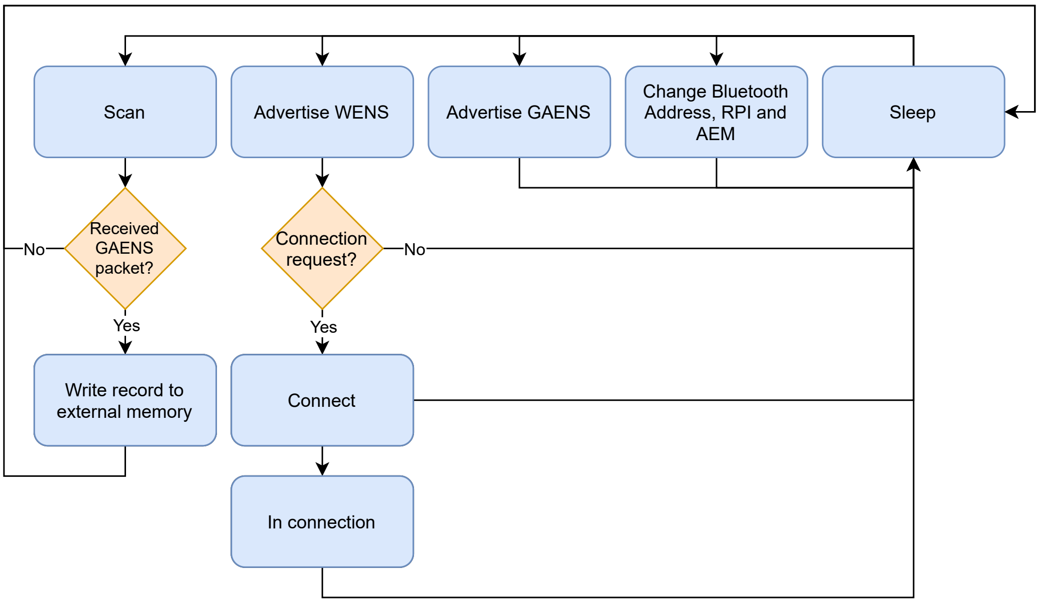
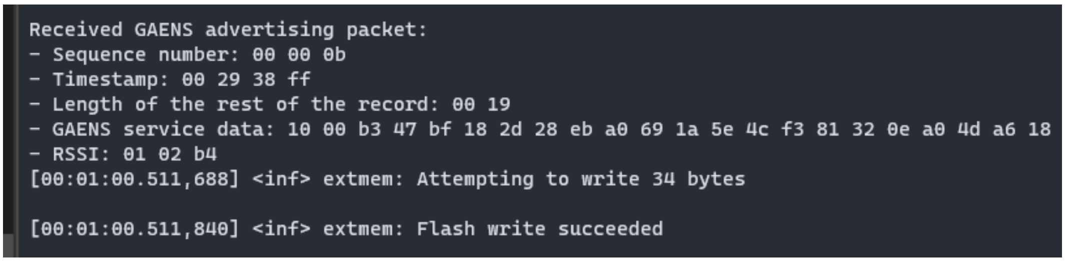


Figure 4.2: A simplified state diagram of the implementation after the initialization is finished.

The non-connectable GAENS advertising is set up with the suggested parameters from the documentation. The specification suggests a minimum advertise interval of 200ms and a maximum interval of 270ms. The scan interval is 1 minute, and the scan window is configured to be 300ms. For the connectable WENS advertising, 1s advertise interval is used.

4.2 Confirming the Functionality of the Implementation

After developing the software and flashing a development kit with it, the solution had to be tested. To test if the wearable was able to filter out and receive GAENS advertising packets from the Smittestopp app, the wearable was placed near a smartphone with Smittestopp active. The wearable was set up to print the received packets in the terminal window. In Figure 4.3 a screenshot from the terminal window can be viewed. It displays the received GAENS service data (RPI + AEM), the sequence number that is assigned by the wearable, the timestamp for which the packet was received and the RSSI value. The wearable attempts to store the record and writes it successfully to the external memory. This shows that the scanning works as it should, and the received records from nearby ENS-participating devices are stored successfully.

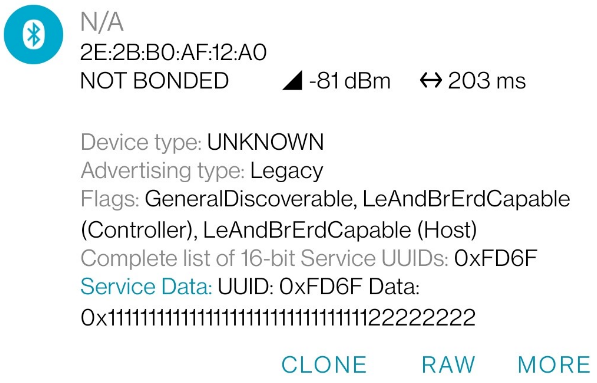


```
Received GAENS advertising packet:
- Sequence number: 00 00 0b
- Timestamp: 00 29 38 ff
- Length of the rest of the record: 00 19
- GAENS service data: 10 00 b3 47 bf 18 2d 28 eb a0 69 1a 5e 4c f3 81 32 0e a0 4d a6 18
- RSSI: 01 02 b4
[00:01:00.511,688] <inf> extmem: Attempting to write 34 bytes

[00:01:00.511,840] <inf> extmem: Flash write succeeded
```

Figure 4.3: A screenshot of the terminal window displaying a received advertising packet from a device with the Smittestopp app active. All the values are written in their hexadecimal value.

Verifying that the GAENS advertising packets is configured properly can be done with the nRF Connect application. As many people are using the Smittestopp app, the RPI was configured to be only ones and the AEM to be only twos, so that the advertising packets from the wearable can be separated from all the other GAENS advertising packets nearby. In Figure 4.4 a screenshot from the nRF Connect application is shown. It displays that it receives the GAENS non-connectable advertising packet every 203ms, with the correct UUID and service data. This illustrates that it works according to the specification. The

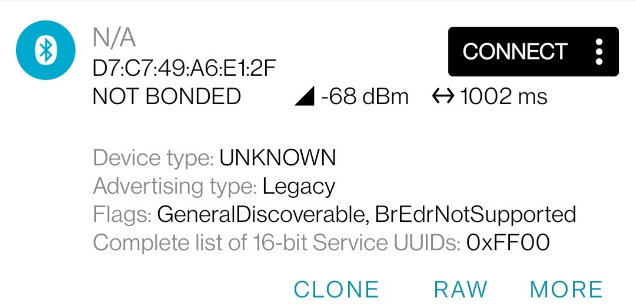


N/A
2E:2B:B0:AF:12:A0
NOT BONDED ◢ -81 dBm ↔ 203 ms

Device type: UNKNOWN
Advertising type: Legacy
Flags: GeneralDiscoverable, LeAndBrErdCapable (Controller), LeAndBrErdCapable (Host)
Complete list of 16-bit Service UUIDs: 0xFD6F
Service Data: UUID: 0xFD6F Data: 0x1111111111111111111111111111111122222222

Figure 4.4: A screenshot from nRF Connect showing that the smartphone receives the GAENS advertising packets from the development kit.

WENS advertising packets should be connectable, and it should be possible to connect to the wearable and access the characteristics. This can also be tested in the nRF Connect app. Figure 4.5 shows that the wearable advertises connectable packets every 1002ms. The UUID (0xFF00) is just a temporary ID selected for testing purposes, as the service UUID has not been released yet.

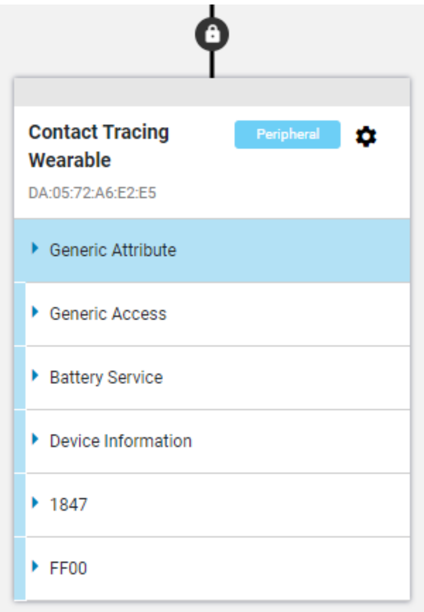


N/A
D7:C7:49:A6:E1:2F
NOT BONDED -68 dBm ↔ 1002 ms

Device type: UNKNOWN
Advertising type: Legacy
Flags: GeneralDiscoverable, BrEdrNotSupported
Complete list of 16-bit Service UUIDs: 0xFF00

Figure 4.5: A screenshot form nRF Connect showing that the smartphone receives the WENS connectable advertising packets from the development kit. The UUID is just a temporary value used for testing, as the UUID is not defined yet by Bluetooth SIG.

In Figure 4.6 one can see that the computer is connected to the development kit and has access to six services. *Generic Attribute* and *Generic Access* are default services that are required. nRF Connect does not recognize two of the services, the DTS (0x1847) and the WENS (0xFF00). This is because the DTS is a new service, and the WENS UUID is a temporary one used for testing purposes. In Appendix C the characteristics of each of the services is shown.



Contact Tracing Wearable
DA:05:72:A6:E2:E5

- Generic Attribute
- Generic Access
- Battery Service
- Device Information
- 1847
- FF00

Figure 4.6: A screenshot form nRF Connect showing that the smartphone can connect to the development kit and get access to the services.

Since the WENS' UUIDs has not been released yet, temporary ones is used for testing purposes. It was chosen not to finish implementing all the functionality of the characteristics because the documentation is only preliminary, and their functionality is not relevant for the rest of this thesis. This is something that is saved for future work.

Chapter 5

Storage Requirements

The GAENS require memory space to store all the ENS records for 14 days. In this chapter, the storage requirements of the wearable will be calculated.

5.1 Memory Estimations

The ENS data received from nearby transmitters will need a place to be stored. The BLE SoC cannot provide enough storage space alone. Hence, an external memory is needed. Using GAENS, one record entry takes up 34 bytes, as represented in Table 5.1.

Field	**Bytes**
Sequence Number	3
Timestamp	4
Length	2
ENS Data	22
RSSI	3
Total	**34**

Table 5.1: The size of each record that is being stored using GAENS.

An external memory unit adds a significant contribution to the wearable's power consumption, size, and price. This makes the choice of what external memory chip to use important. Before one starts looking into what external memory chip to use, one has to estimate how much data needs to be stored, and how the external memory chip will be used.

How many people you meet on average every day will impact the required storage. A person who meets many others will receive more packets than those who meets few. Estimating how many transmitters are nearby is difficult, as everyone lives different lives. For most people, one can divide a typical day into different types of social settings. The WENS specification suggests those to be home, work, commute and social. In Table 5.2 one can see the table from the WENS specification. This is their scenario:

"An example is shown below of a working adult who lives with a family of five people. From 7 p.m. to 7 a.m., this person is home and within range of at most four other devices, perhaps less while sleeping. The rest of the day involves activities with more people nearby, although even in crowded scenarios, there is a limit to how many people will typically be within a 2-meter radius. Even on crowded subways, a device held on the body will have a limit to how many other devices it can successfully receive." [1]

	Transmitter in Range	**Hours per Day**
Home	4	12
Work	12	8
Commute	14	2
Social	10	2

Table 5.2: An estimate for how many transmitters that are in range under different situations during a day. This gives an average of 8 transmitters in range [1].

How many packets are received per scan interval on average is dependent on several factors. For the calculations, there is made the assumption that there are no packet collisions, and that all the packets advertised by nearby ENS devices will be received successfully. The scan interval is 300ms and the advertising interval is between 200ms and 270ms. The device will try to advertise if it has available resources, and most of the time it can advertise immediately, but not always. For the calculations, 205ms will be used as an average value.

A simulation was developed in order to decide the statistical average number of packets that will be received by a device during a scan window if only one advertising device is nearby, advertising at a specific advertising interval. In Figure 5.1 the resulting graph from the simulation is shown. The simulation shows that a scan interval of 300ms and an advertise interval of 205ms give an average number of received packets per device nearby of 1.46 packets. Assuming no collision, all packets are received successfully, and an average of 8 devices nearby, this results in receiving around 12 advertisement packets per scan window on average.

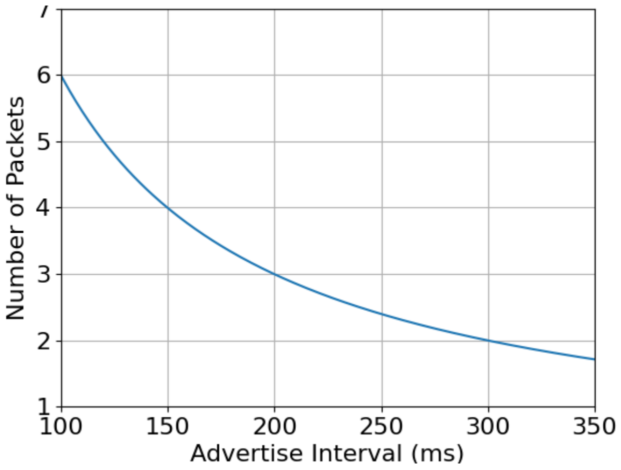


Figure 5.1: The statistical average number of packets received with different advertise intervals for a 300ms scan window. The figure is generated from the script in Appendix E.

The Python script used to perform the simulation is shown in Appendix E. It works by simulating a scan window and a nearby device advertising at specific advertising intervals, and by shifting the timing of the advertising packets one can find the average number of received packets for all cases of timing shifts.

In Table 5.3 one can see the numbers used for estimation of memory requirements for a contact tracing device. This table is slightly different from the one suggested in the WENS specification. Instead of using the RPI interval (which is variable), and an expected number of receptions per RPI interval that is not explained, an estimation of how many packets being received per scan window (which is fixed) is used. With the specifications in Table 5.3 the device would require a minimum memory size of 4113kB. This corresponds to about 32Mb. This means that a 32Mb external flash memory should be sufficient.

	Value	**Unit**
Scan Off Time	60	Seconds
Scan On Time	300	ms
Advertisement Interval	205	ms
RPI Interval	~15	minutes
Average Number of Transmitters in Range	8	devices
Estimated Receptions per Scan Window	12	records
Records per Day	17280	records
Storage per Day	588	kB
Days of Storage	7	days
Minimum Memory Size	**4113**	**kB**

Table 5.3: Estimation of minimum memory size without compression.

In the calculations, it is assumed that the records are stored on the wearable for 7 days. In reality, it could be shorter, because one would want to connect the wearable to a client, such as a smartphone, to upload the records to it every day or two. The 144 TEKs and their corresponding $ENIntervalNumber$, that needs to be stored, is disregarded in the calculations, as their required storage space is insignificant when compared to the ENS records.

Chapter 6

Current Consumption Estimations and Measurements

With wearables powered by batteries, it is important to keep the current consumption as low as possible. There are many techniques that can be used to achieve low power consumption. From a hardware design perspective, this generally means developing a minimalist design with components suitable for low power consumption. However, how the hardware is put into use is equally important. The software needs to be efficient and put the device into low-power modes when possible, otherwise, battery capacity is wasted. To come up with an ideal solution, one first needs to figure out what the device needs to do.

To start off, the current consumption of the PoC prototype is measured. This will give a base for how much current an unoptimized solution would draw. Next, the current consuming operations that the wearable is required to perform is measured on a nRF52833 development kit, to isolate the contributions of each of the operations which later can be used for optimization calculations for the engineering prototype.

6.1 PoC Prototype Current Consumption

Before any estimation of the current consumption contribution of each operation, the total consumption for the PoC prototype is measured. To do so, the PoC prototype was flashed with the software described in Chapter 4, and connected to the PPK. The resulting current consumption graph of the PoC prototype is showed in Figure 6.1. The average current consumption is approximately 532µA. If it was powered by a CR2032 coin cell battery, with 200mAh capacity, it would last for close to 16 days. This is far less than the desired battery life, which is several months.

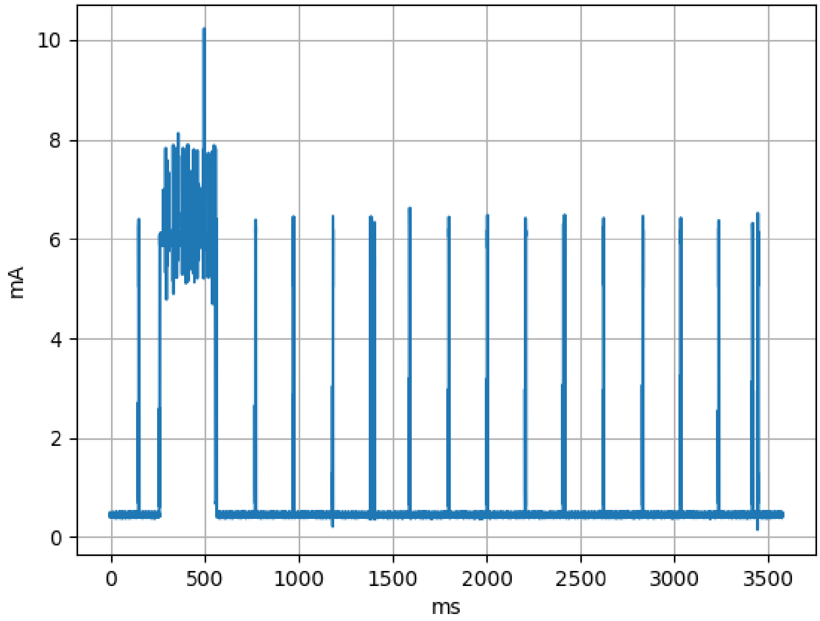


Figure 6.1: This measurement shows the current profile of the PoC prototype, programmed with the GAENS and WENS implementation described in Chapter 4.

6.2 GAENS Current Consuming Operations

The GAENS contribution to the power consumption comes in the form of BLE communication and cryptography operations that need to be performed at certain intervals. In addition, it contributes in the form of writing to the external memory to store ENS records. The following is the required operations that will contribute the most to the current consumption for GAENS:

- Bluetooth
  - Non-connectable advertising
    - Suggested advertisement interval is 200ms (min) - 270ms (max).
  - Scanning
    - Suggested scan on time 300ms.
    - Suggested scan off time is 60 seconds.
    - Store received GAENS advertisement packets.
- Cryptography
  - Generate new RPI and AEM in a randomized interval larger than 10 minutes and less than 20 minutes.
  - Generate new RPIK, AEMK and TEK every 24 hours.
- Memory operations
  - Store ENS records.

6.3 WENS Current Consuming Operations

From WENS most of the energy consumption comes in the form of BLE communications and memory operations. It needs to advertise connectable advertisement packets for devices to connect, and during connections data is exchanged over BLE and ENS records are read and manipulated. The following is the required operations for WENS with its corresponding suggested parameters:

- Bluetooth
  - Connectable advertising
    - There is not specified a required or suggested interval, but since it is not critical to connect with it immediately 1 second will be used.
  - Connections
    - During connections one will typically read and write to the different characteristics, and transfer ENS records.
    - It will also read and manipulate the external memory's data.

6.4 Estimations and Measurements

Nordic Semiconductor offers a PPK for the current consumption of their BLE SoC on their development kits. To be able to calculate the power consumption during different circumstances and parameters, it is necessary to find out how much current the BLE SoC consumes during the different operations. The different operations that contribute to the current consumption are as follows:

- Scanning
- Advertising
- Connections
- Sleep mode
- Randomized rotation of GAENS service data
- External memory

In the next sections, all the operations will be looked into. Their average current consumption and duration will be measured or estimated. All the estimations and measurements are conducted with 3.0V power supply for the nRF52833 BLE SoC at 0dBm transmit power.

6.4.1 Non-connectable GAENS Advertising

In Figure 6.2 one can see the current consumption during a non-connectable packet. The packet is configured to transmit with 0dBm transmit power and has 31 bytes of payload. The average current for each non-connectable advertising packet is 3.00mA, and it takes 3.1ms to transmit. This corresponds well with the online power profiler from Nordic Semiconductor. The large spike in the current is possibly caused by the PPK shunt resistor switching to increase the resolution. However, the spike does not make too much impact on the measurement and is disregarded.

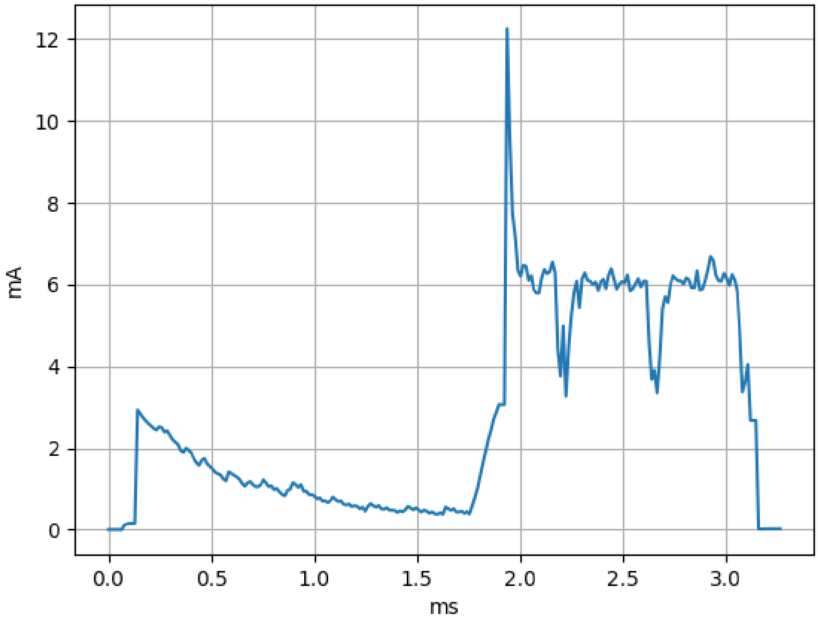


Figure 6.2: Current consumption of one non-connectable GAENS advertising packet transmitting at 0dBm transmit power and is 31 bytes long.

6.4.2 Connectable WENS Advertising

During connectable advertising, the radio both transmits and scans for a response. This means the radio switches several times during connectable advertising, since it has to advertise and scan on all the three advertise channels. In Figure 6.3 the current consumption of a connectable advertisement packet can be seen. The advertisement packet consists of 7 bytes of data, as shown earlier in Table 2.2.

Most likely, the radio switching causes the PPK to measure inaccurately. As mentioned, the PPK switches resistance when the current changes significantly in order to obtain a higher resolution. It could be that the current changes too fast back and forth for the PPK to follow. When compared to the online power profiler, there is a significant difference in the current consumption [15]. This is confirmed by using a multimeter to verify the measurements. A more accurate solution would be to use the estimate from the online power profiler. The online power profiler says that a connectable advertisement takes 3.78ms and has an average current during transmission of 2.70mA.

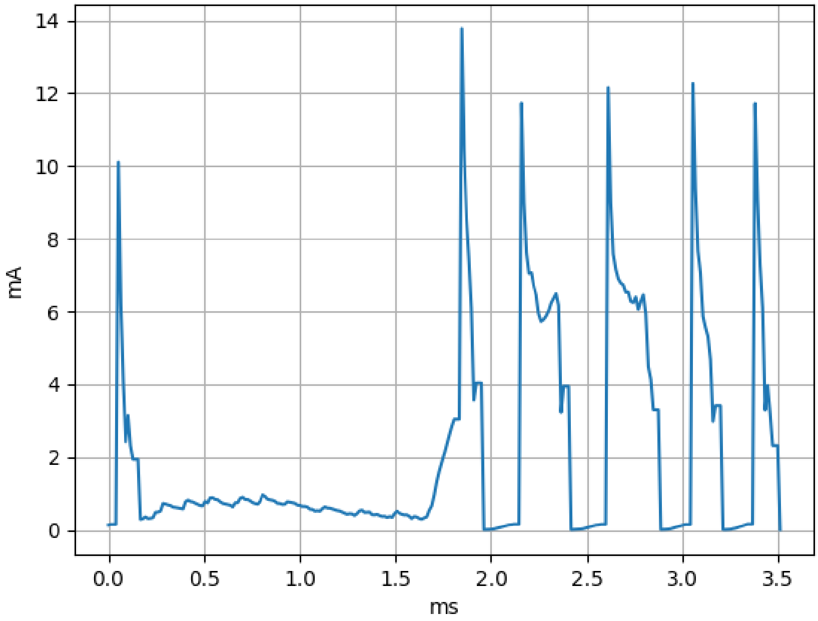


Figure 6.3: Current consumption of one connectable WENS advertising packet transmitted at 0dBm transmit power and with 7 bytes of payload.

6.4.3 Scanning

In Figure 6.4 one can see one period of scanning for 300ms. The current consumption is quite stable at 5.65mA. According to the nRF52833 specification, the SoC will use 6.0mA when receiving at 1Mbps BLE mode [17]. This confirms that the PPK measurements are quite accurate, but for further calculations 6.0mA will be used.

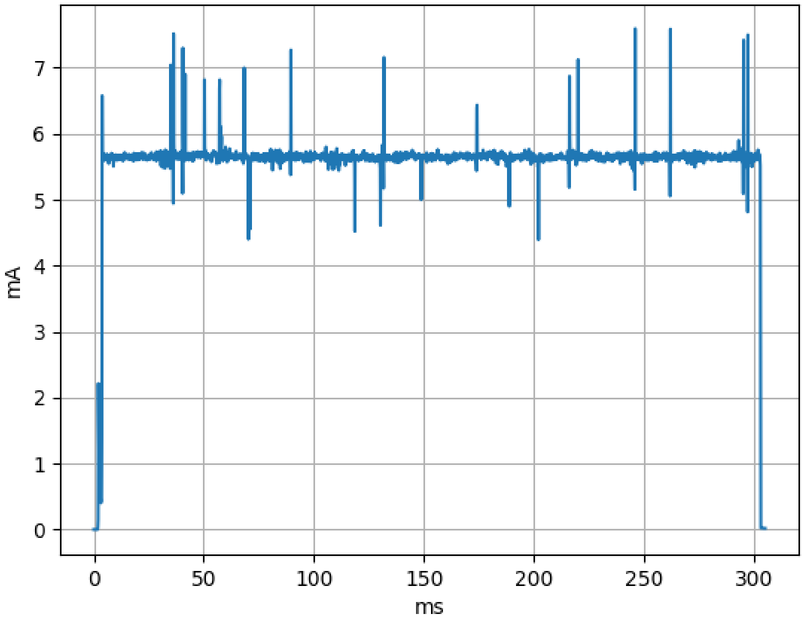


Figure 6.4: Current consumption of one 300ms scanning interval.

6.4.4 Sleep mode

A wearable participating in an ENS will spend most of its time in a low power state, such as sleep mode. The wearable has a limited battery capacity, so one would want to spend as much time in the lowest power state that it can as possible. The average sleep mode current for nRF52833 is measured to be around 2.4µA. In Figure 6.5 one can see the current consumption during sleep mode. The measured current corresponds well with both the online power profiler and the nRF52833 product specification [15, 17]. The current consumption could have been closer to 2.0µA, but the GAENS implementation requires "wake on RTC" to be enabled in order to wake up the device for changing the RPI, BLE address and AEM at a randomized rotation interval, which increases it slightly.

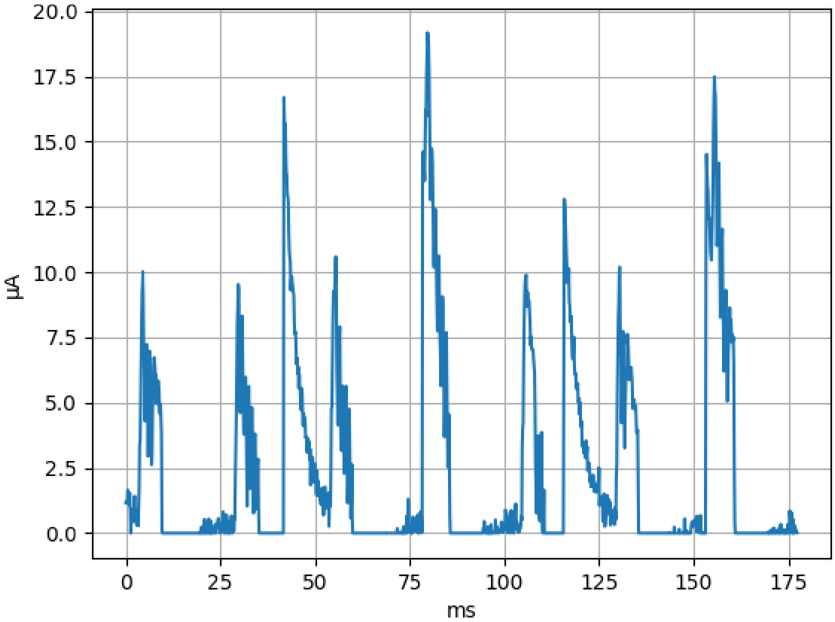


Figure 6.5: Current consumption during sleep mode with "wake on RTC" enabled.

6.4.5 Connections

It is difficult to say exactly how much current is drawn during connections as there is no solution to test this against. One factor is how often one would connect the wearable to a smartphone, and another one is how it is used, which is controlled by the user. It is also dependent on the smartphone that is connected to the wearable (i.e. if it is supporting data packet length extension and 2Mbps PHY).

However, it is possible to make some assumptions and get a rough estimate. As the wearable will spend very little of its time in connections, it should not make too much of a difference. For the estimations, the online power profiler will be used.

It is important that the wearable has a high transmit throughput to increase the ease of use, because much of the connection time would be spent transferring records for the client to process. The parameters used for calculating the current consumption during connections are:

- Data Packet Length Extension
    - By enabling Data Packet Length Extension, one increases the allowable payload size of application data while connected to 251 bytes. This significantly increases throughput, as the default is 27 bytes.
- 2 Mbps PHY
    - Using 2 Mbps mode opposed to the default 1 Mbps mode.
- Slave latency 0
- 251 byte TX payload
    - This is the maximum number of bytes that can fit in one event.
- 128 byte RX payload
    - As the wearable will mostly be receiving smaller packets from the client, in the form of operation codes, 128 bytes would be an overestimate for how much it will receive in reality.
- Connection interval 7.5ms
    - This is the shortest connection interval that is allowed, which enables faster communication.
- 0 dBm radio TX power
- 2µA idle current

When inserting these settings into the online power profiler one gets an average current of 2.02mA, a TX throughput of 267.73kbps, and a RX throughput of 136.53kbps. The time it takes to transfer all the ENS records over BLE is given by Equation (6.1). If using an external memory chip of 32 Mb, as estimated in Table 5.3, and assuming it is full, it would take approximately 120 seconds to transfer all the records. To make room for other operations to be performed as well, one can estimate that the connections last for three minutes a day. It is important to notice that if one connects to a client each day, the external memory would not be full of new records, so three minutes should be a significant overestimate.

$$Complete\ record\ transfer\ time = \frac{Size\ of\ records}{Throughput} \tag{6.1}$$

6.4.6 Randomized Rotation of GAENS Service Data

According to the GAENS Bluetooth specification, the BLE address shall change at a randomized rotation timeout interval larger than 10 minutes and less than 20 minutes. When the BLE address changes, the RPI and AEM also has to change immediately, so that they cannot be linked. This is done by the `_rotate_rpi_handler` as shown in Appendix D. Zephyr changes the advertising BLE address automatically when stopping and starting to advertise.

Assuming that the randomized rotation timeout interval is truly random, it will execute the handler every 15 minutes on average. This will contribute to the current consumption.

To determine how much it will affect the current consumption, one has to find out the average current when executing the handler, and how long it takes before the device is put back to sleep. In Figure 6.6 one can see the current consumption during the rotation of GAENS service data.

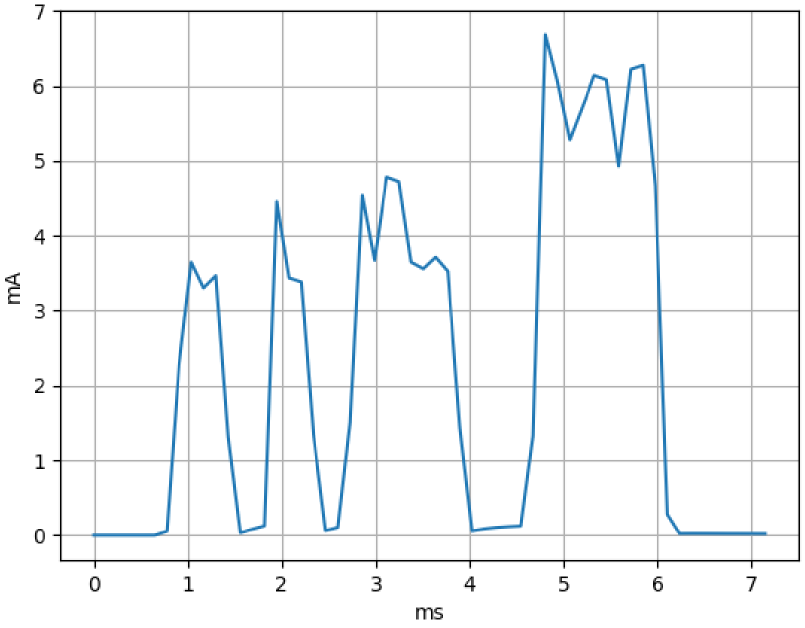


Figure 6.6: Current consumption during rotation of GAENS service data.

The graph does not show any major spikes, and the current consumption seems to correspond with the nRF52833 product specification [17]. The rotation of GAENS service data and BLE address takes approximately 5.5 ms and consumes 3.0mA on average.

6.4.7 External Memory

The external memory unit will also add to the current consumption. The contribution depends on the implementation and the characteristics of the chip. The most important factor to consider is the current the external memory consumes when it is not doing anything, since most of the time it will not be in use. The external memory will only be used if the device receives GAENS advertising packets from nearby devices, and during connections, to read out or make changes to the records.

The flash memory chip on the PoC prototype is probably not the most optimal solution. It does not support sleep modes, which results in a high average current consumption. So, to find a more suitable example to use for power consumption estimations, DigiKey was searched for memory chips with the following criteria:

- Part Status: Active

- Memory Type: Non-Volatile
  - It is important that memory can retain stored information after power is removed so that one does not lose any records unintentionally.
- Memory Size: 32Mbit
  - According to memory usage estimations.
- Memory Format: FLASH
- Memory Interface: SPI
- Voltage Supply: ~3V
- In stock

The top result, sorted after stock, is W25Q32JVZPIQ [18]. This chip supports low power modes and fulfills every criterion. It is also among the lowest priced ones. Based on the criteria, W25Q32JVZPIQ is chosen as an example for further calculations. The flash memory chip support power-down mode with a power consumption of 1µA. It takes 5ms to wake up the device from power-down mode, but that should not be a problem for this WENS and GAENS implementation. One can wake it up at the beginning of scanning windows and connections so that it is accessible when it is needed, and power it back down again when the operations are finished. The standby current is 10µA.

Read operations can be performed at different speeds, 50MHz, 80MHz and 104MHz. These speeds are significantly higher than what the nRF BLE SoCs support. Those have a maximum data rate of 8Mbps. During 50MHz read operations, the memory chips consume 8mA, so that can be used as an estimate, even though it could be lower than that because of the decrease in data rate. Writing to the memory uses 20mA.

As the records will be placed consecutively in memory, it is enough to send one instruction and address when reading and writing because the address is automatically incremented so that one allows for a continuous stream of data. Though for reading and transferring data over Bluetooth, one cannot read all the data at once, because the BLE SoC does not have enough memory to buffer the data all at once before it is transmitted. The time that is not spent reading from the memory during connections, the external memory chip will spend in standby mode. The reason that it will be in standby mode is so that the wearable can respond quickly upon request for ENS records.

Writing operations can be done one page at a time (256 bytes), so the operation codes and addresses that are required in the SPI transactions to read and write will not be considered for calculations, as they are not significant.

Given that the device will receive around 12 records per scan window, it will have to write 360 bytes to storage every scan window. According to the datasheet it takes 0.4ms to program a page, and 360 bytes would take up a little less than two pages, but to approximate let say it takes up two. This means that storing the received packets in each scan window takes 0.8ms at 20mA.

During connections, the BLE TX throughput and RX throughput were estimated to be 267.73 kbps and 136.53 kbps. This means that reading and writing from the external memory is significantly faster with its 8Mbps. As the size of the memory is 32Mb and the SPI data rate is 8Mbps, one can read the whole memory in 4 seconds. However, as mentioned, these readings will have to be performed in bursts because the data cannot be buffered on the BLE SoC before transferring the data over BLE. Assuming that one reads the whole memory each day, it will spend 4 seconds a day reading the external memory at 8mA. This should be an overestimate, as the memory will most likely not be filled each day. In addition to this, let us say that the client chooses to clear all ENS data each day by writing to the WEN Status characteristic of the WENS. A chip erase requires 20mA and takes 10 seconds.

6.5 Total Current Consumption

Now that all the largest contributors to the current consumption have been accounted for, measured or estimated, one can put their contributions together and estimate the total current consumption of the wearable, averaged over 24 hours. In Table 6.1, the current contribution of each isolated operation averaged over 24 hours is showed. The current consumption of the BLE SoC is 92.14µA, and the current consumption of the external flash memory is 3.97µA. This makes the total consumption 96.11µA. Powered by a CR2032 coin cell battery, with an estimated battery capacity of 200mAh, one can expect around 87 days of operation.

BLE SoC				
	Current (mA)	**Duration (ms)**	**Interval (ms)**	**Average Current (µA)**
Non-Connectable GAENS Advertising	3	3.1	205	45.3659
Connectable WENS Advertising	2.7	3.78	1000	10.2060
Scanning	6	300	60000	30.0000
Connections	2.02	180000	86400000	4.2083
Rotation of GAENS Keys and Identifier	3	5.5	900000	0.0183
Sleep	0.0024	84154343.41	86400000	2.3376
				92.1361
External Memory				
Write	20	0.8	60000	0.2667
Read	8	4000	86400000	0.3704
Erase	20	10000	86400000	2.3148
Standby	0.01	166000	86400000	0.0192
Power-Down	0.001	86218848	86400000	0.9979
				3.9690
			Total (µA):	96.1051

Table 6.1: A table showing the current consumption contributions of each isolated operation averaged over all time.

An important note is that this is not the current consumption of the unoptimized PoC prototype from Figure 6.1. The external memory is different, and the device does not go to sleep between the operations, which it should do. Why the PoC prototype does not enter sleep mode is discussed further in Section 7.1.

Chapter 7

Optimization

In the previous chapter, the average current consumption of the unoptimized PoC prototype was found to be approximately 532µA. In this chapter some techniques to reduce the current consumption, and increase the battery life, will be investigated. The measurements are done using the PPK together with a nRF52833 development kit.

7.1 Disabling Serial Communication

Having serial communication active, such as SPI or UART, will prevent Zephyr RTOS from putting the wearable in sleep mode when it has nothing to do, because it waits for possible communication. This is what is happening in Figure 6.1, where one can see that the wearable is not put in sleep mode in between advertising and scanning packets. Not putting the device into sleep mode increases the power consumption drastically and has to be avoided. To fix this issue, one has to disable the serial communication after finishing the operations done to the external memory.

The resulting current consumption curve for the nRF52833 development kit after disabling serial communication can be viewed in Figure 7.1. With the serial communication disabled, the Zephyr can put the device in sleep mode in between the BLE communication operations. The development kit does not have an external memory that would otherwise contribute to the current consumption. After this optimization, the current consumption for the engineering prototype would be closer to the estimated current consumption from Table 6.1, 96.11µA.

The current consumption of the separate operations can be compared in a histogram to see what of the operations contributes the most to the consumption. The histogram can be seen in Figure 7.2. The histogram shows that the BLE advertising and scanning are the major contributors to the current consumption. The non-connectable advertising, connectable advertising and scanning consume 89.04% of the total current consumption and is probably the most important operations to optimize for current consumption.

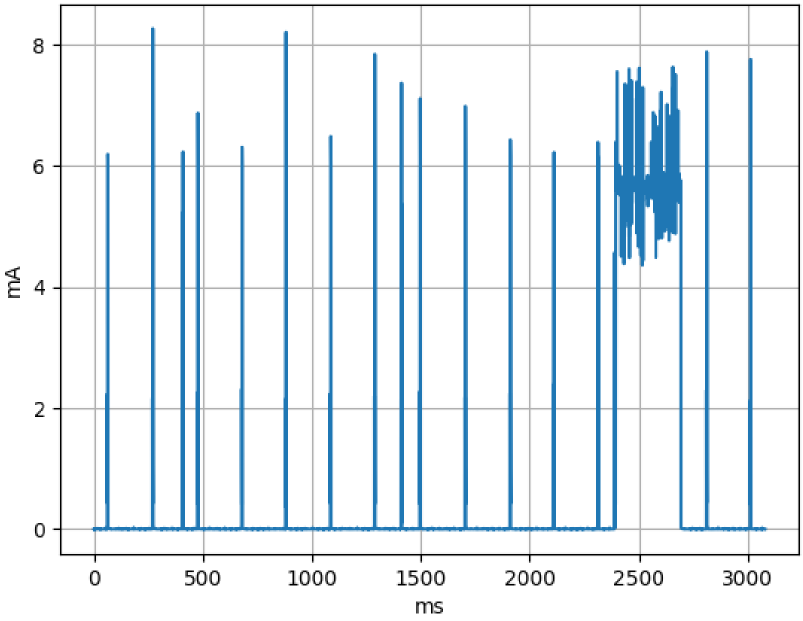


Figure 7.1: This measurement shows the current profile of the nRF52833 development kit with the GAENS and WENS implementation after disabling serial communication.

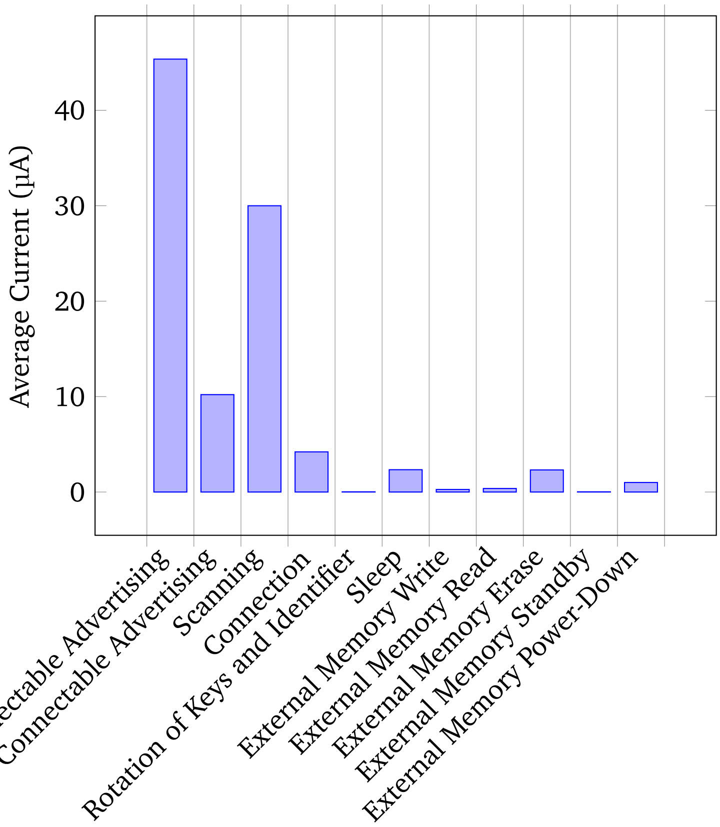


Figure 7.2: A histogram showing the average current consumption of each isolated operating of the BLE SoC and the external memory averaged over 24 hours.

7.2 Accelerometer

A large part of the day it is not necessary to perform contact tracing. When the wearable is not worn, the wearable could be put into a lower power state, instead of operating as normal. One way of determining if the wearable is worn is using an accelerometer. When the wearable has not been moving for some time, the wearable can be put into sleep mode until it is moving again. However, an accelerometer would add size to the wearable, and increase the cost. Also, it is important that the accelerometer does not add more to the power consumption than it saves. If an accelerometer with low power consumption, low cost and small size is found, the power consumption could be reduced.

There exist ultra low-power accelerometers that are suitable for an application like a contact tracing wearable. To give an example it was searched for active, in stock, accelerometers on DigiKey, and a component called LIS331DLH was found [19]. This is an ultra low-power accelerometer with a current consumption of 250µA in normal mode, 10µA in low-power mode and 1µA in power-down mode. With those numbers, it should be possible to reduce the overall power consumption of the contact tracing device. The sensor supports setting an acceleration-triggered interrupt with a specific trigger value in terms of magnitude and duration, even if the accelerometer is in power-down mode (wake on acceleration). As the wearable does not need to know anything about the acceleration itself, other than it happened and was larger in magnitude and longer in duration as specified in the LIS331DLH interrupt configuration registers, no data needs to be transferred other than an interrupt on a pin. The interrupt can be used to wake up the wearable from sleep. When waking up, the wearable starts a timer for putting the device into power-down mode again, and the timer is reset each time the wearable receives an interrupt from the accelerometer. By doing so, the wearable should be in sleep mode when it is not used, and active when it is used.

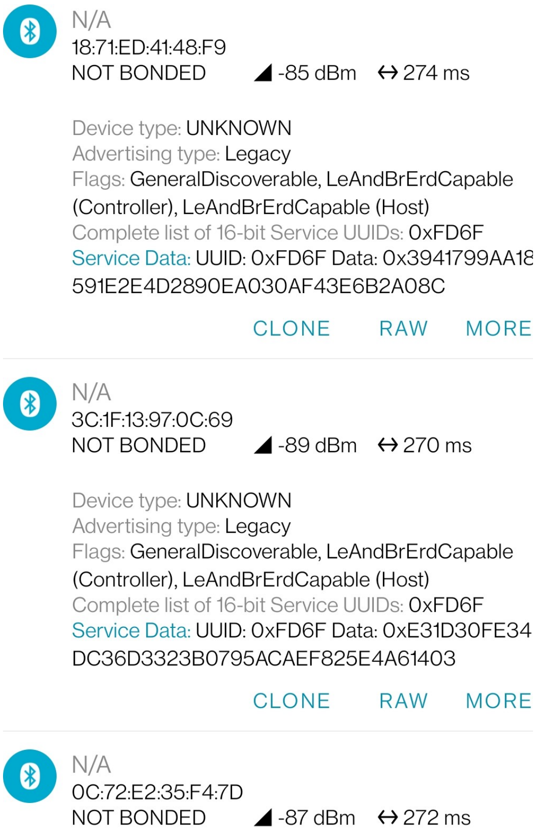
With the accelerometer operating as explained, it will be in power-down mode almost all the time. It will consume some more energy when being configured, and when triggering interrupts. Since the power-down current consumption is 1µA and the current consumption in low-power mode is 10µA for the example accelerometer LIS331DLH, and the device spend most of the time in power-down mode, one can estimate that the average current consumption of the accelerometer is around 3µA.

A regular person sleeps around 8 hours each day. When sleeping, one would most likely not need to wear the contact tracing wearable. In addition to that, one does not need to wear it all the time at home either, especially during weekends with the family and no one else around. So one can estimate that the wearable will not be used for at least 10 hours a day on average.

If the wearable sleeps 10 hours a day on average the advertising, scanning, reading and writing to the external memory will be reduced by $10/24 \cdot 100 = 41.67\%$. While the BLE SoC and the external memory will spend more time in sleep mode, hence increasing the sleep mode current contribution slightly, but that is not significant and will not be considered. The realistic reduction in power consumption is slightly less than 41.67%, because of the current consumption the of the accelerometer itself. The current consumption of the wearable with only an accelerometer as optimization technique can be seen in Table F.1. The current consumption is reduced from 96.11µA to 63.18µA, which is a reduction of 34.26%.

7.3 Scan and Advertise Adjustments

There are no fixed restrictions to what the advertise and scan intervals should be. There have been observed scan intervals up to 4 minutes and advertise intervals of 250ms for GAENS implementations [20]. To test what advertisement interval is being used by the Norwegian Smittestopp app, three smartphones with Smittestopp activated was placed near a development kit. By using nRF Connect, the developement kit picked up their advertisement packets as shown in the screenshot in Figure 7.3. From the figure, it looks like all the smartphones were using around 270ms advertise interval for their GAENS advertisement packets. This interval will probably vary, because advertising intervals are not exactly fixed, but with a minimum interval and a maximum interval. The minimum and maximum interval could be the same as the GAENS specification suggests. The wearable has a low computational load and can most of the time advertise at the minimum interval because the device does not have any high-priority task to do. Advertising at an interval of 270ms, as opposed to 205ms (average), will decrease the power consumption of GAENS non-connectable advertising by 24.07%.



N/A
18:71:ED:41:48:F9
NOT BONDED ◢ -85 dBm ↔ 274 ms

Device type: UNKNOWN
Advertising type: Legacy
Flags: GeneralDiscoverable, LeAndBrErdCapable (Controller), LeAndBrErdCapable (Host)
Complete list of 16-bit Service UUIDs: 0xFD6F
Service Data: UUID: 0xFD6F Data: 0x3941799AA18
591E2E4D2890EA030AF43E6B2A08C

CLONE RAW MORE

N/A
3C:1F:13:97:0C:69
NOT BONDED ◢ -89 dBm ↔ 270 ms

Device type: UNKNOWN
Advertising type: Legacy
Flags: GeneralDiscoverable, LeAndBrErdCapable (Controller), LeAndBrErdCapable (Host)
Complete list of 16-bit Service UUIDs: 0xFD6F
Service Data: UUID: 0xFD6F Data: 0xE31D30FE34
DC36D3323B0795ACAEF825E4A61403

CLONE RAW MORE

N/A
0C:72:E2:35:F4:7D
NOT BONDED ◢ -87 dBm ↔ 272 ms

Figure 7.3: A screenshot of the nRF Connect app, when scanning and filtering for GAENS advertisement packets. It is done when being around people who use the Norwegian Smittestopp app.

According to the GAENS Bluetooth specification, the device should have sufficient coverage to discover nearby ENS devices within 5 minutes [3]. In the previous calculations a scan interval of 1 minute, a scan window of 300ms and an advertise interval of 205ms have been used. These are only recommended values picked from the WENS specification. With those numbers, the wearable will receive 1.46 packets from nearby ENS devices on average per scan window. In 5 minutes there are 5 scan windows if 1 minute scan intervals is used. This means that the wearable has on average 7.3 chances to receive an advertisement packet from the nearby ENS in 5 minutes. One could argue that this is more than enough. As long as the advertise interval is shorter than the scan window, it will still get at least one chance at receiving a packet from each other ENS wearable nearby.

How many attempts an advertise packet needs to be discovered by another ENS device to discover the device within 5 minutes depends on the setting. In a crowded area with several devices operating in the 2.4GHz band, there will be more packet collisions and noise, so that receiving a packet is more difficult. A study was conducted upon determining the chances of packet collisions with various advertise intervals for a network of BLE nodes of different sizes [21]. The study, "*Detailed Examination of a Packet Collision Model for Bluetooth Low Energy Advertising Mode*", found the probabilities to be as illustrated in Figure 7.4. With those findings, one can make a better judgment on what scan interval and window that is suitable for an advertise interval of 270ms.

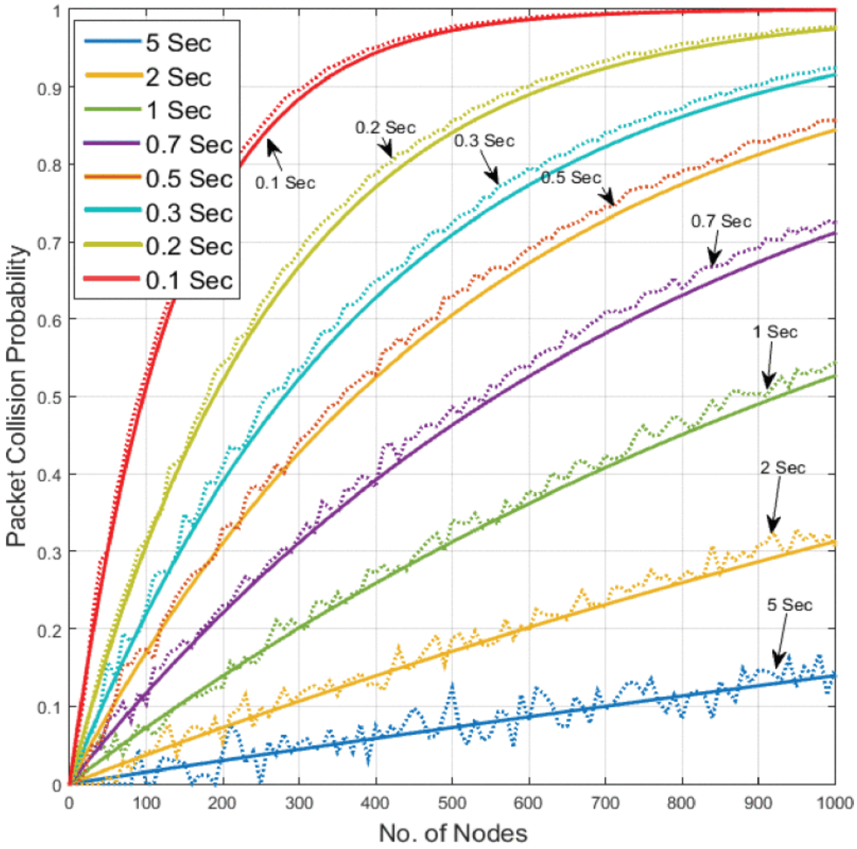


Figure 7.4: Packet collision probability for different advertise intervals (0.1s to 5s) and number of nodes [21].

In today's society, there are many wireless devices around us most of the time. Exactly how many depends on the situation. In public buildings and apartment blocks, you can easily have 100 BLE devices within range. Let us assume it is 200 devices in range on average, which would probably be an overestimate. All those devices use different advertise intervals, ranging from about 100ms to 2 seconds, depending on the situation. It is assumed, for the following calculations, that the average scan interval of the devices in range, is 300ms. With 200 devices in range and an average advertise interval of 300ms there is a packet collision probability of around 40% according to Figure 7.4.

With a packet collision probability of 40% on average through the day, it implies that there is a 40% risk of each advertisement packet not reaching another device in range, disregarding other disturbances. If one treats the event of a packet collision as independent events, the probability of two consecutive packet collisions occurring is the probability of the first collision times the probability of the second collision. That means that there is a 16% probability of two consecutive advertisement packets both colliding with other advertisement packets and get lost, and a 6.4% probability that three consecutive packets will be lost. So if an ENS device has three chances of receiving an advertisement packet from another ENS device nearby within 5 minutes, there is a 6.4% risk that it will not receive any of them, but a 93.6% probability that it will receive at least one of them. Those odds should be sufficient. That could be done by scanning for 300ms every 100s (1m and 40s). Since the scan window, in this case, is longer than the 270ms advertisement interval, the device should have at least one chance at receiving an ENS advertisement packet from nearby devices in each scan window. By extending the scan interval from 1 minute to 1 minute 40 seconds, the scan current consumption is reduced by 40%.

The WENS' connectable advertising current consumption can also be reduced more. It has no strictly specified advertising interval that needs to be used. Until now in the calculations 1 second advertise interval has been used, but it could also be increased to for example 1.5 seconds or 2 seconds. It will then of course take more time to connect to the wearable. By using 1.5 second advertisement interval one would reduce the connectable advertisement average current by 33.33%, and increasing it to 2 seconds would reduce it by 50%. Increasing it to more than 2 seconds could make it difficult to connect to.

In Table F.2 the calculation of the current consumption of the wearable optimized with adjusting the scanning and advertising intervals is shown.

7.4 Reducing Transmit Power

The purpose of contact tracing is to determine your close contacts and notify those who have a risk of being infected. As one does not care for those people that are far away, because there are little to no risk of getting infected by a person that is far away from you, there is no point in transmitting ENS BLE packets that reach people far away from you. In other words, the transmit power could perhaps be reduced while the functionality of the ENS stays the same.

There are multiple different configurations that can be done to the transmit power. One could change the transmit power for all radio communication, change transmit power only for non-connectables or other configurations. What is the best solution needs to be tested. It could be that these parameters will be more strictly set by the specification in the future.

Since the connectable advertising packets are transmitted with a relatively long interval, it could be wise to not reduce the transmit power beyond 0dBm because it would make it even more difficult to connect to the wearable. Also, you would want a stable connection during connections, meaning that reducing the transmit power for communication during connections probably should be avoided.

For the GAENS non-connectable advertising packets, the transmit power could possibly be reduced without making any sacrifices to the accuracy of the ENS. A paper, "*Range of Bluetooth Low Energy Beacons in Relation to Their Transmit Power*", suggests that even with -12dBm transmit power the BLE communication sustains for several meters, even through thick walls [22]. Though it is important to notice that their testing were performed with few other obstructions around. Having the BLE device in the pocket, or sitting on it, will reduce the range of the device significantly. However, being able to communicate using -12dBm BLE transmit power through with the devices placed 4 meters apart with a 182cm thick concrete wall between them, and also being able to communicate at more than 7 meters apart through a 166cm thick drywall, should suggest that -12dBm transmit power will work for the GAENS broadcasting as well.

In Figure 7.5 the difference in current consumption between 0dBm and -12dBm GAENS non-connectable advertising is compared. There is a significant reduction. The average current for the approximately 3.1s long advertising sequence has dropped from 3.0mA to 2.3mA. In other words, the non-connectable GAENS advertising operation current contribution, has dropped by 23.33%. The current consumption of the wearable, only optimized by reducing the transmit power for GAENS advertising, is shown in Table F.3.

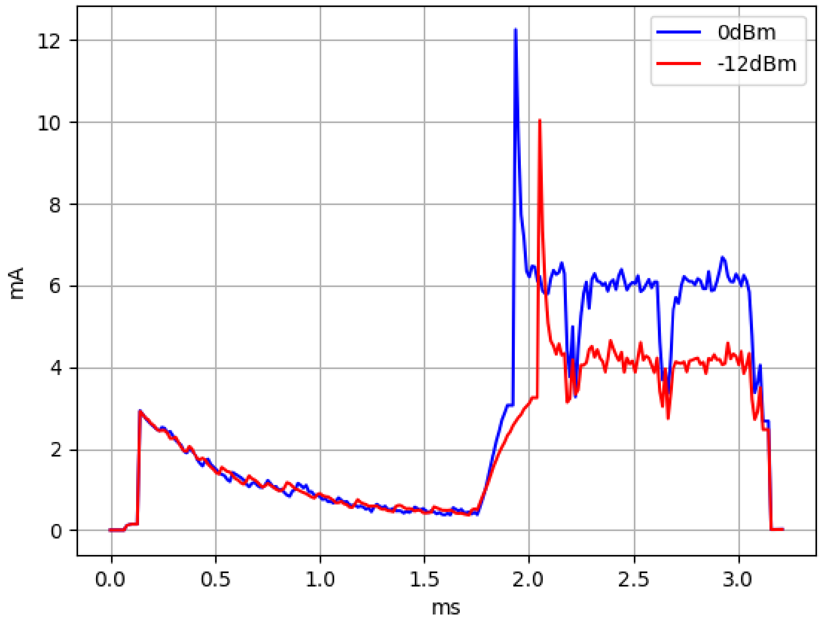


Figure 7.5: A comparison of 0dBm and -12dBm transmit power for non-connectable GAENS advertising.

Chapter 8

Hardware Implementation

In this chapter, the process of designing the hardware for the engineering prototype is discussed. That includes the selection of design, introducing the appearance model, selecting components and presenting the development of the PCB.

8.1 Wearable Designs

As a part of the alpha phase of the prototyping timeline, an appearance model needs to be designed. This will set the base for the design of the electronics hardware. The PoC prototype gives an indication of the size of a final contact tracing wearable design. There are many possibilities to the design to make it convenient for the end-user. In the following sections, possible designs that have been considered for the wearable will be evaluated, and one of the designs will be chosen for the engineering prototype.

8.1.1 Wristbands

The initial idea for the wearable was to design it as a wristband. However, after a market analysis was performed in the specialization project, it was found that some people would rather have a different design. Many already use wristwatches, which leaves less room for a contact tracing wearable. Some were also worried that wrist-worn devices might introduce a greater risk of getting infected. Another point is that a wrist-worn device is difficult to design so that it fits all end-users.

8.1.2 Access Cards

Many public buildings and workplaces already require one to wear an access card around in the building. By embedding the electronics required to perform contact tracing inside the access card, one can participate in an ENS without the users having to wear another wearable. Access cards would most likely require more hardware and software to function as both contact tracing devices and access cards if the cards is used to open doors.

8.1.3 Tag

Tags have become a popular design choice for small electronic wearables, especially for Radio-Frequency Identification (RFID) devices. A tag is a small capsuled device, which is often flat, round and has a hole in it so that you can connect it too for example key rings.

A recent example of such a design is Apple's AirTag. Apple's AirTag was released in April 2021, and after the release iFixit did a teardown of the device to look inside [23]. The AirTag has a nRF52832 BLE SoC, 32Mb serial NOR flash and is driven by a CR2032 coin cell battery. In addition to this it has several other components as well such as an ultra-wideband transceiver and an audio amplifier. In other words, it has all the necessary components to serve as a contact tracing wearable, and more. It shows a possible design of a contact tracing wearable in the form of a tag, and gives a picture of how small it is possible to design such a solution.

A tag can be worn in many ways as it is small. One could for example connect it to key rings, pants, a bracelet or just have it in your pocket.

8.1.4 Bracelet

A bracelet design is simple, and is one of the possible designs where it is possible to create a single solution that would most likely fit all. However, not everyone is comfortable wearing something around the neck all day.

8.1.5 Embedded Into Clothing

A convenient wearable is a wearable you do not feel that you are wearing. From the market analysis performed in the specialization project, some people wished to have a contact tracing wearable embedded into clothing for convenience. It would also be easier to make people who suffer from dementia wear a contact tracing wearable, because they often do not understand, or forget why they have to wear the device. A solution that is embedded into clothing would most likely have to be designed in cooperation with a clothing company, and would possibly introduce issues with changing batteries.

8.2 Choosing a Design

There are pros and cons with all the possible design variations. Designing the wearable as a tag is the option that seems the most promising. It is a small design and could be worn in many ways. When it was decided to go for a design similar to a tag, the author of this thesis asked a friend of his, Martin Falang, to create an appearance model using Fusion 360, with instructions on the desired look. The resulting appearance model is shown in Figure 8.1. With the appearance model developed, the alpha phase of the prototyping timeline is finished.

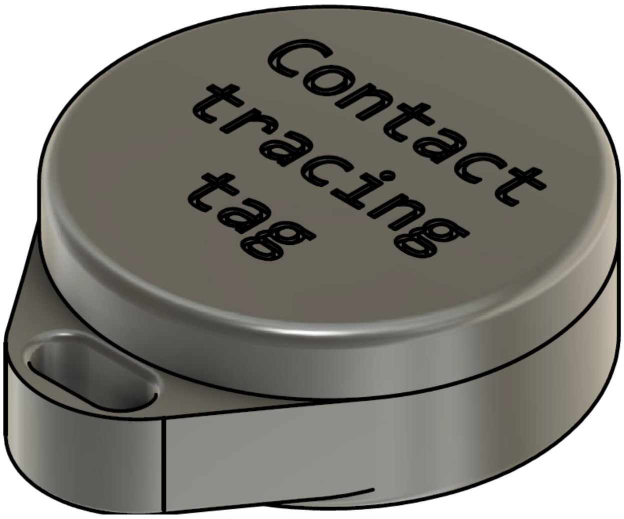


Figure 8.1: An appearance model of the wearable.

8.3 Selection of Hardware Components

In the Chapter 7 it was found that an accelerometer could reduce the current consumption significantly. Hence, an optimized solution of the wearable should include an accelerometer. From earlier it was also found that the required components of the wearable is the BLE SoC, and its circuitry, a 32Mb external flash memory supporting SPI, a CR2032 battery holder and an antenna.

8.3.1 BLE SoC

The nRF52833 BLE SoC is chosen for the development of the engineering prototype. The reasons are the same as for the selection of the chip for the PoC prototype. It is one of the more feature-rich BLE SoC and does not have as many constraints as some of the less complex ones. The BLE SoC can be replaced in the future if it turns out that the full software implementation and requirements allow it.

8.3.2 Antenna

The engineering prototype needs a compact, reliable and low-cost antenna. There are two common antenna solutions, using a chip antenna or a PCB trace antenna [24]. Both solutions have pros and cons. Chip antennas are in general small of size and are available with all kinds of configurations. They are also more resistant to environmental interference than PCB trace antennas. The main drawback of chip antennas is that they often result in a higher cost. PCB trace antennas are embedded in to the PCB. This means that they have a low profile. While it has a low profile, it usually occupies more area than a chip antenna. PCB trace antennas are also more difficult to design and are highly susceptible to changes to the board layout, which means it may require tuning after each change.

The PoC prototype has a PCB trace antenna, which works quite well. However, the PCB trace antenna takes up a large portion of the PCB area. For the engineering prototype, a chip antenna will be used instead. It is easier to design and takes up less area. The chip antenna that will be used is the 2450AT18A100E chip antenna from Johanson Technology [25]. This is the same antenna that is used inside Nordic Thingy, which suggests that it should be suitable [26].

8.3.3 External Memory

As mentioned in Section 6.4.7, the external memory chip in the PoC prototype design is not optimal, and has to be replaced. During the optimization research, it was also discovered that the external flash memory chip that was used in the optimization calculation is not optimal either. This is because of its narrow voltage operating range. The Functional End Point (FEP) of the W25Q32JVZPIQ is 2.7V which is a lot higher than the FEP of the BLE SoC with its 1.7V [17, 18]. This restricts the battery lifetime a great deal. This is further discussed in Section 9.2.1.

The same search parameters as in Section 6.4.7 are used to find a new, more suitable external flash memory chip. Except this time the focus were on finding one with a wider operating range. The search led to the AT45DB321E-MHF-T chip [27]. This 32Mb SPI flash memory has a FEP of 2.3V, which would improve the battery life of the wearable. The new flash memory chip also has approximately the same current consumption as the one used in the optimization calculations.

8.3.4 Battery Holder

The battery holder on the PoC prototype, Harwin S8421-45R, is simple and does not add that much to the size of the wearable, since the battery needs to be there anyway [28]. The same battery holder will be used for the engineering prototype.

8.3.5 Accelerometer

The ultra low-power accelerometer from Section 7.2 used in the optimization calculations, LIS331DLH, seems promising. It has a wide supply voltage range and meets all the other requirements discussed in Section 7.2. The same accelerometer will be used in the hardware design for the engineering prototype.

8.4 Circuit Schematic Design

With all the components selected, the circuit schematic can be designed in Altium Designer. The circuit schematics of the engineering prototype can be found in Appendix G. In the following sections, some comments about the design choices are discussed.

8.4.1 Serial Peripheral Interface

Both the accelerometer and the external memory chip supports SPI communication. This means that they can share some of the communication lines. The SPI protocol specifies four logic signals, Serial Clock (SCLK), Master Out Slave In (MOSI), Master In Slave Out (MISO) and Chip Select (CS) (active low). In a SPI system, the master supplies the clock to the slaves. The accelerometer and the external memory can be connected together as illustrated in Figure 8.2.

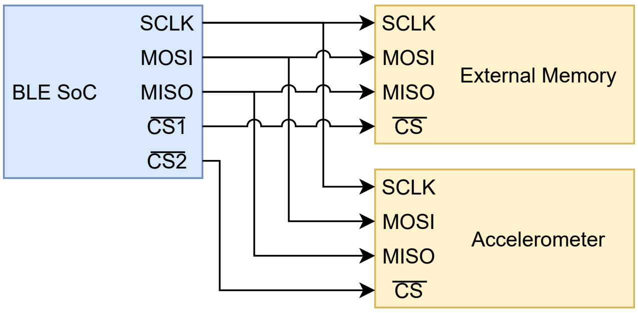


Figure 8.2: An illustration of the SPI configuration between the BLE SoC, the external memory and the accelerometer.

8.4.2 Programming and Debugging

The wearable needs connection points so that it can be flashed with code and debugged. The PoC prototype solved this by using a pin header and castellated holes. The pin header is large and is something one would want to avoid. The castellated holes are probably one of the connection point types that requires the least space. However, castellated holes are quite expensive to manufacture. For the engineering prototype, small circular pads, called testing points, will be used. These points can be used to monitor the board's circuitry when it is being tested, but they can also be used to inject signals into the PCB, like flashing.

The connection points that are required are the ones used for Serial Wire Debug (SWD), ground and power. The SWD pins on the BLE SoC, `SWDIO` and `SWDCLK`, can be used for debugging and flashing the device. One would also want some other pins in the case of more extensive testing. The `P0.18/RESET` on the BLE SoC could be useful too, as reset functionality can be a good tool for testing. In addition, a couple of other pins could be good to have available too, for example for using UART communication. `P0.15` and `P0.17` were selected for this purpose because of their location near the SWD pins.

8.4.3 Antenna Matching Network

The circuitry around the BLE SoC is based on a reference layout for the nRF52833 QDAA QFN40 [17]. In the reference layout there is a pi matching network to match the BLE SoC to 50Ω. It also has the beginning of the transmission line. The chip antenna also has to be matched to 50Ω. That is done according to the datasheet of the 2450AT18A100E chip antenna [25]. The matching networks are shown in Figure 8.3.

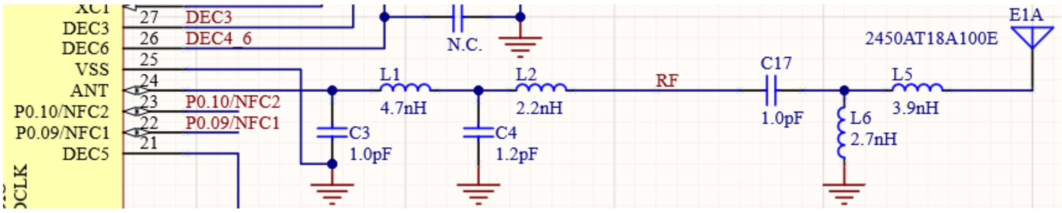


Figure 8.3: The pi matching networks to match both the BLE SoC and the chip antenna to 50Ω.

8.4.4 Changes to the Reference Layout

Nordic Semiconductor supplies several reference layouts for the nRF52833 QDAA QFN40 to use [17]. None of them are ideal for the engineering prototype. The reference layouts comes with different configurations for DC/DC, Universal Serial Bus (USB) and Near Field Communication (NFC). For the engineering prototype, DC/DC converter functionality is desirable because without it the power consumption would increase drastically. Both USB and NFC are not required. The USB will not be in use, so the USB pins can be assigned to ground. When the USB is not in use, the `VDDH` and `VDD` can be connected together. The NFC pins are left disconnected.

8.5 PCB Design

The PCB was designed with the goal of being small and fit into a circular tag design. The appearance model is circular, and so is the CR2032 coin cell battery, so the shape of the PCB was chosen to be circular as well. The coin cell battery holder is the largest part of the design and is placed on the back of the PCB. This makes room for all the other components on the top layer. The connection points are placed along the edge of the board on the top layer so that they are easily accessed. The accelerometer and the external memory chip share some of the same pins, so it was chosen to put them close together. Since the power supply is placed on the backside of the PCB the bottom layer was chosen to be a power plane, while the other three planes are all ground. The finished PCB is presented in Section 9.3, together with the other results.

Chapter 9

Results and Discussion

In this chapter, the results of the work will be presented and discussed. First, the functionality of the engineering prototype will be looked into. This is followed by the results of combining all the optimizations methods, to see what the final estimated current consumption and lifetime of the device is. At the end of the chapter, the final hardware design is introduced and examined.

9.1 Functionality of the Software

The functionality of the software were already verified in Section 4.2, as it had to be tested before the rest of the work was conducted. The GAENS cryptography and Bluetooth specification were successfully implemented on the prototype wearable. The wearable was able to receive advertise packets from the Norwegian Smittestopp app, and store the records. It was also able to advertise GAENS packets and encrypt the RPI and AEM according to the GAENS specification, so that it is ready to be used together with Smittestopp in the future.

The WENS was partly implemented and tested. The wearable advertised the connectable packets at the specified interval, and it was possible to connect to the wearable and view it services. When connected to the wearable, the smartphone can request to read out the characteristics, and also write to them. It was not important to complete the implementation of the functionality of the characteristics to verify that the solution would work, nor performing the current measurements and estimations. Finishing the development of WENS will have to be done in the future when the final specification comes. Considering the situation the world finds itself in, it is likely that the WENS will be finalized and taken into use by existing ENSs, so that wearable can participate a well.

The findings in this thesis suggests that the wearable can take part in already existing ENSs, given that they support WENS. The software implementation for the wearable is not complete, but the main framework is. After a final publication of the WENS comes, it can be completed. The software is documented, and peer-reviewed, so it should have a good enough quality for further development by someone with experience in BLE and

C-programming.

9.2 Optimizations

Without any optimizations, it was found that the PoC prototype had an average current consumption of around 532μA through measurements. This was significantly improved by disabling serial communications when it is not used. That gave an estimated average current consumption of 96.11μA for the engineering prototype, as calculated in Table 6.1.

There was suggested several other optimization methods too:

- Use an accelerometer to determine if the wearable is worn.
- Increasing GAENS advertising interval to an average of 270ms.
- Increasing WENS advertising to 2s.
- Increasing scan interval to 100s.
- Reduce the GAENS advertising transmit power to -12dBm.

By combining all those current consumption reduction techniques, one can achieve an even greater reduction. The combination of all the optimizations results in the average current consumption of each isolated operation as shown in Figure 9.1. The numbers are the average current consumption when it is averaged over all time. This gives a total average current consumption for the engineering prototype of 42.17μA. The calculations are showed in Table 9.1. With the assumption that the CR2032 delivers an effective 200mAh, this implies a battery life of around 197 days. An estimated battery life of 197 days is more than half a year, which is well within the requirements of the problem statement. Even though it is only an estimated value, it should indicate that "*a battery life of several months on a single coin cell battery*" is achievable.

BLE SoC				
	Current (mA)	**Duration (ms)**	**Interval (ms)**	**Average Current (μA)**
Non-Connectable GAENS Advertising	2.3	3.1	270	15.4043
Connectable WENS Advertising	2.7	3.78	2000	2.9768
Scanning	6	300	100000	10.5000
Connections	2.02	180000	86400000	4.2083
Rotation of GAENS Keys and Identifier	3	5.5	900000	0.0183
Sleep	0.0024	84804976	86400000	2.3557
				35.4634
External Memory				
Write	20	0.8	60000	0.1556
Read	8	4000	86400000	0.2160
Erase	20	10000	86400000	2.3148
Standby	0.01	166000	86400000	0.0192
Power-Down	0.001	86218848	86400000	0.9979
				3.7035
			Total:	42.1670

Table 9.1: The calculation of the average current of each operation isolated operation averaged over all time where the wearable is using all optimization techniques that is discussed in this thesis.

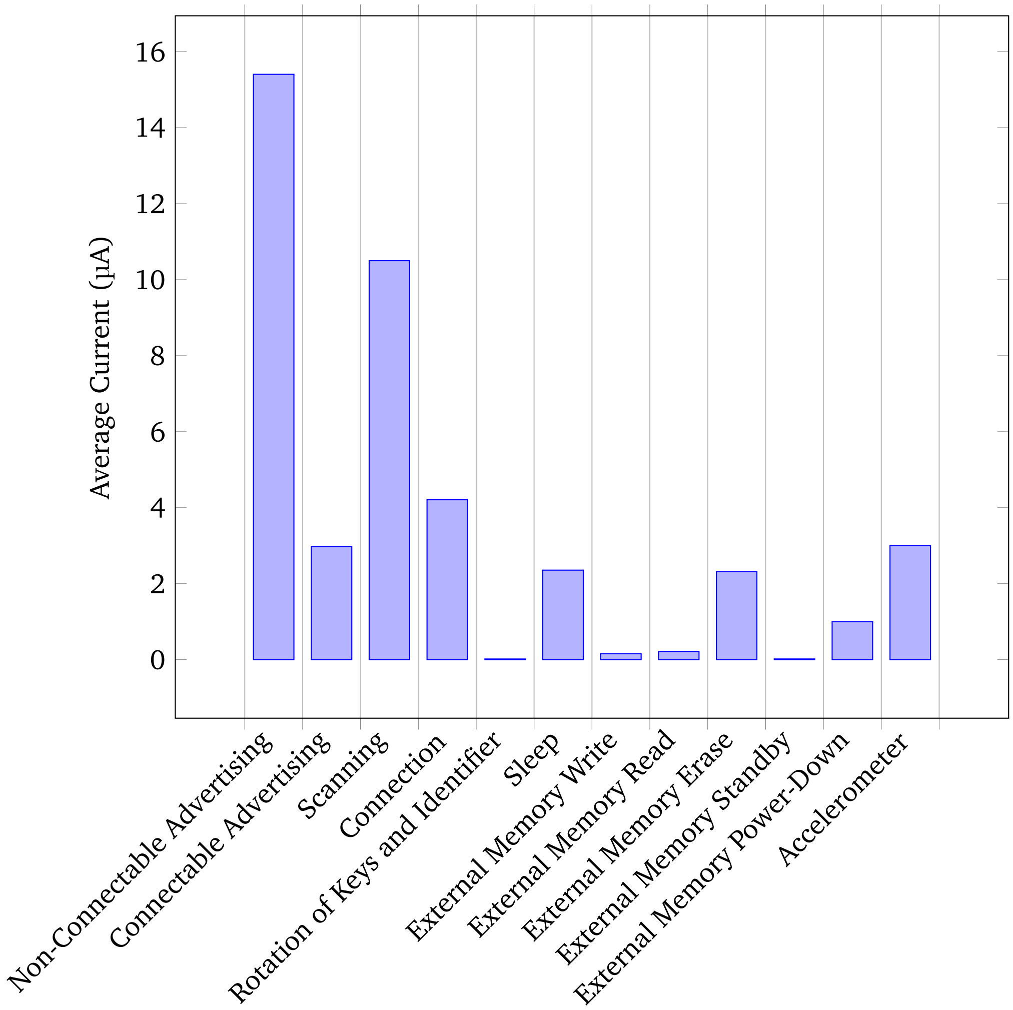


Figure 9.1: This histogram shows the 24-hour average current consumption of each isolated operating of the BLE SoC, the external memory and the accelerometer after combining all the optimization methods.

9.2.1 Considerations

The current consumption analysis done in this thesis is an estimate of much current the device will draw from the power supply on average. There are several assumptions to the calculations, but most of them are based on overestimates. In this section, some factors affecting the current consumption are discussed.

Variations in Usage of the Wearable

With a solution using an ultra low-power accelerometer, there could be large variations in the actual lifetime of the device. This is due to how the device is being used. Everyone lives different lives, meaning some will be using the wearable more than others. In the calculations there are made the assumption of the wearable not being used for 10 hours a day on average. The less the wearable is in use, the longer it will be running. For the combined optimization solution, the expected battery life of the wearable is shown in Figure 9.2. It shows how the expected battery life varies with the average amount of hours per day the device is in use. If the device is used 24 hours a day it will have a battery life of about 130 days, while if it is not used 16 hours a day on average you can expect about 280 days of battery life. This also shows the great effect the accelerometer has on reducing the current consumption and increasing the battery life, even when the other optimizations are implemented.

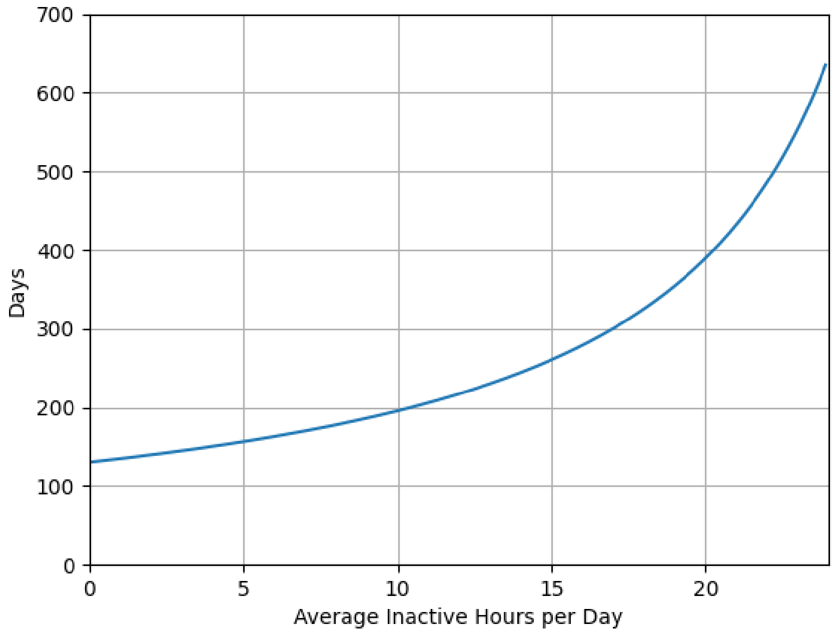


Figure 9.2: The expected number of days of operation for the optimized wearable, based on the average hours it is used per day.

Possible Accelerometer Issues

A potential issue with using an accelerometer as a sensor to determine if the wearable is worn or not, is that the user could sit still for a while and not trigger the accelerometer even though the wearable is worn. If that happens, the wearable will stay in sleep mode while the user is wearing it. This could lead to situations where someone walks by, or sits next to the user, while the wearable still is in sleep mode. Meaning that the wearable will not advertise nor receive advertising packets from the other person. So this has to be tested in the future to see if it is an issue. Considering that accelerometers are often used for similar low-power purposes, they should also work for the contact tracing application as well. The parameters for magnitude and duration of the acceleration used for interrupt generation also need to be determined through testing.

Another uncertainty regarding the calculations in this thesis, is the average current consumption used for the accelerometer (3µA). It could be that the use of the wearable will significantly affect the current consumption. One can not be sure what the actual current consumption of it is before it is tested.

External Memory Selection

After looking into the effects of high current drains on the CR2032 coin cell battery it was found that the FEP of the external memory used in the optimization calculations (W25Q32JVZPIQ), which was 2.7V, should have been lower [18]. The FEP of the NRF52833 is 1.7V, which means that there is a big difference in their FEP [17]. The external memory chip would make the wearable stop operating when the voltage of the CR2032 drops below 2.7V, which limits the accessible capacity from the battery significantly. By selecting an external memory chip with a lower FEP, but still in the 3.0V operating range, one can extend the battery life of the wearable.

A more suitable SPI flash memory was found when designing the hardware for the engineering prototype. Namely, the AT45DB321E-MHF-T, which is also a 32Mb SPI flash memory [27]. The AT45DB321E-MHF-T also has approximately the same current consumption as the one used in the optimization calculations. The main difference is that it has an operating range from 2.3V to 3.6V. The FEP is notably lower than the one used in the optimization calculations, and closer to the FEP of the BLE SoC. In Figure 9.3 the impact the FEP has on the obtainable capacity from the CR2032 coin cell battery, is illustrated. With an FEP of 2.3V instead of 2.7V the expected battery capacity is far greater.

Transmit Power

In Section 7.4 it was mentioned that the non-connectable GAENS advertising did not need to use 0dBm transmit power, and that for the connectable WENS advertising and connection communication it should keep the transmit power at 0dBm. Considering that when one wants to connect to a device one generally is close to the device, one could argue that the wearable possibly could get away with reducing the transmit power for

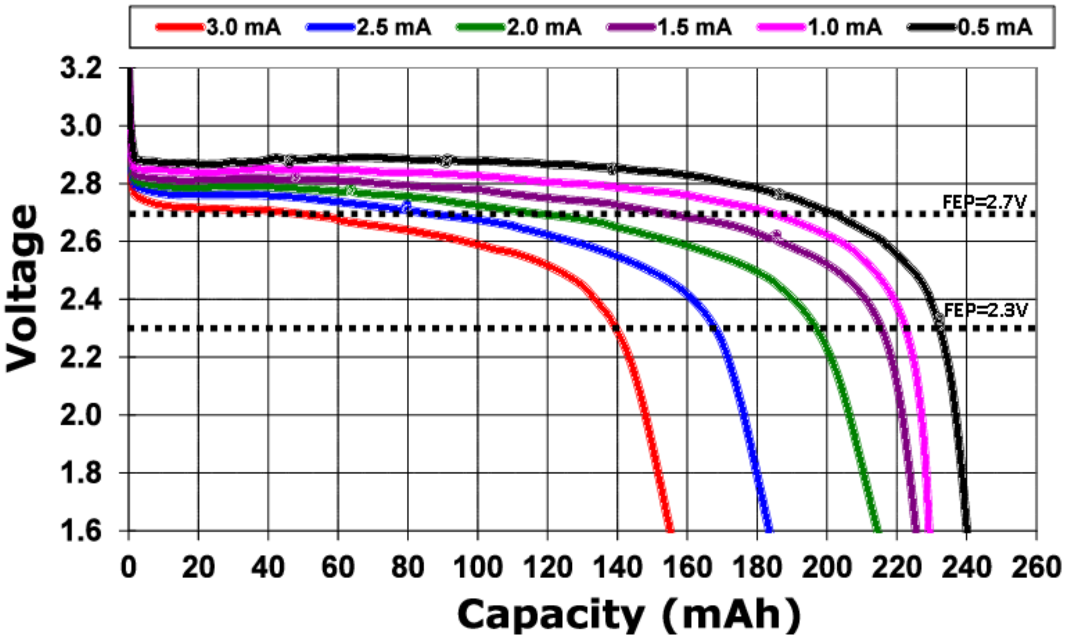


Figure 9.3: The figure of a continuous discharge of CR2032 coin cell battery, from "High pulse drain impact on CR2032 coin cell battery capacity", with the different FEPs for the external memory [13]. The figure illustrated the significant impact the FEP of the wearable has on the effective capacity the wearable can obtain from a CR2032 coin cell battery.

connections and connectable advertising as well. Doing so would reduce the average current consumption further. The effects of reducing the transmit power have to be looked into in the future.

External Memory Erase

Another thing that will reduce the current consumption is that the external memory most likely will not be erased each day. When the wearable is connected to a client, the records are transferred to the client using the WENS. However, it is generally not necessary to delete all the records upon every connection. The records will be automatically deleted by the device after 14 days. It is also useful to leave the records on the device if more than one client bonds with the wearable. If only the 14 days old records is erased every day, the average current consumption of the external memory erase operation will be reduced to 1/14 of the value used in the calculations. Changing it from 2.315µA to 0.165µA. This reduces the total current consumption by 2.15µA, which is significant.

Time Spent in Connections

In Section 6.4.5 there were made several assumptions to the factors regarding connections. How much time will be spent in connection with a client will vary from user to user. The numbers used in the calculations are most likely an overestimate, meaning that the average current consumption from connections should be less. It was assumed that one would spend three minutes in connection with a client, and transfer all the records, each day. The external memory would probably not be filled with new ENS records each day, as it is intended to be able to hold records for 14 days.

Device Information Service and Device Time Service

The two services, DIS and DTS, was chosen not to be considered for the current consumption analysis because they will be used, and their characteristics accessed, during connections. Meaning that they are already indirectly included in the current consumption analysis.

9.2.2 Memory Compression

The WENS specification defines how ENS records should be transmitted on-air, but it does not have any restrictions on how the records are stored internally on the wearable. With lossless compression, it is possible to store the same records on the wearable, only taking up less space [1]. The RPI changes on average every 15 minutes, this means that it is likely that the wearable will pick up several equal RPIs. Instead of storing all the identical records separately, they can be compressed by storing one larger record for each RPI. This means that the common information only will be stored once, while the information that varies, such as the RSSI and the timestamps, will be store in relation to the common information. This will save a significant amount of space. When the wearable is to transmit the records to the client it can convert the compressed records into regular records according to the WENS specification, hence, truly lossless compression. How much the storage requirements is reduced is dependent on if the wearable will be receiving multiple packets from the same transmitters. For example at home or at work people will be around the same persons for a while, indicating that the lossless compression could save storage space.

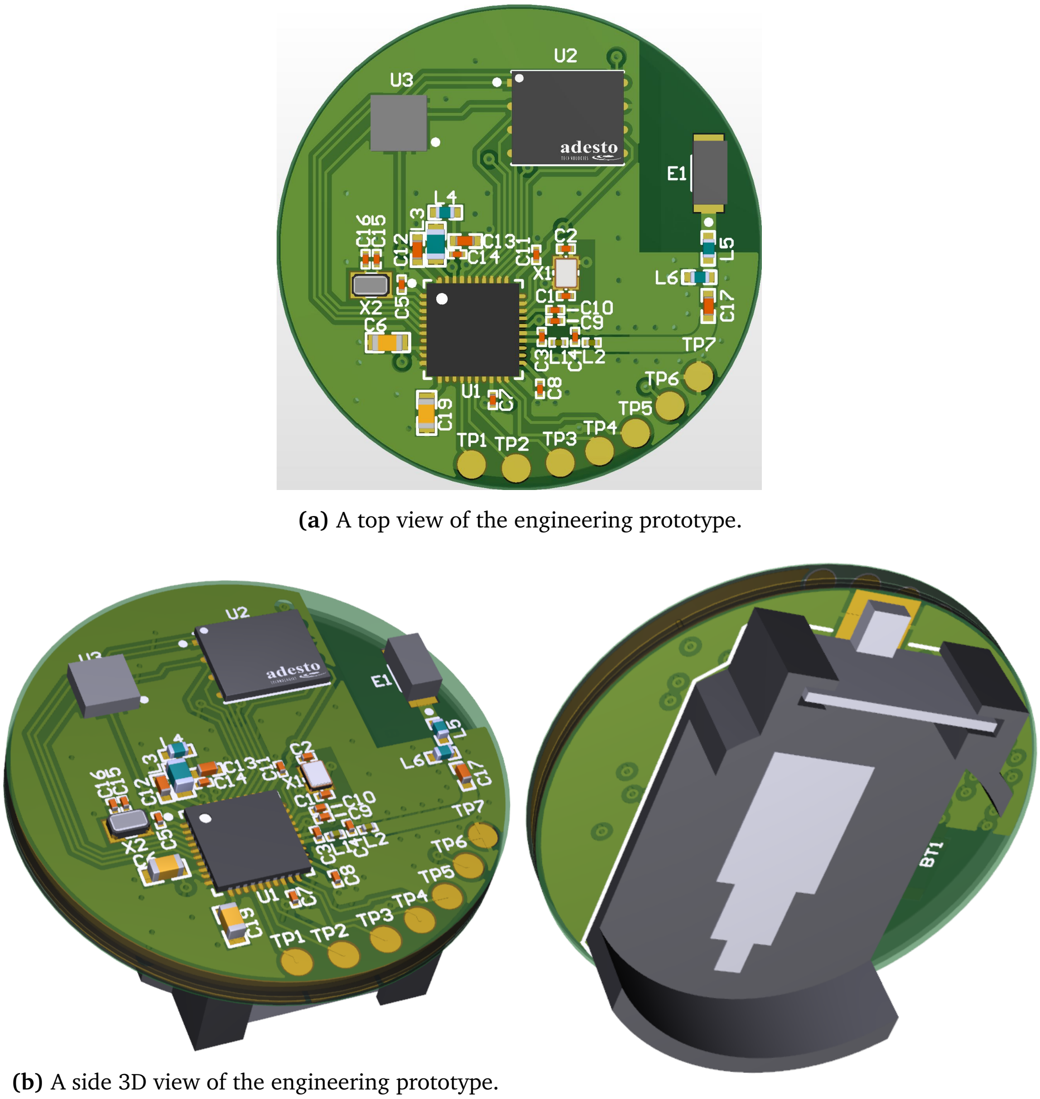
The Optimization's Effect on Storage Requirements

The optimization measures in increasing the scan interval and the advertising interval will reduce the memory storage requirements as the wearable will receive fewer packets. The same goes for an implementation using an accelerometer to turn the device off when it is not used. When the wearable is sleeping, it will not be receiving any packets. If the wearable is regularly connected to a smartphone for uploading the records to it, the memory requirements will drastically go down as well. Instead of storing the information for all 14 days, they can be transferred to a client to store them for the wearable. If the wearable would reliably upload the data every 2 days, one could reduce the storage by a factor of 7. This again depends on the implementation. As mentioned in Section 9.2.1, it could be beneficial to keep the records on the wearable in case the wearable will be connected to more than one client.

9.3 Hardware

The hardware was successfully designed according to the specifications. The resulting engineering prototype of the PCB can be seen with a top view in Figure 9.4a, with a side view in 3D in Figure 9.4b and from the bottom in 3D in Figure 9.4c. In Appendix H the PCB print of all the layers in the PCB design can be seen.

The 3D model of the PCB was imported into Fusion 3D and the appearance model was adjusted to serve as a casing for the PCB. The resulting production drawing of the engineering prototype can be seen in Figure 9.5. The PCB itself is around 26mm in diameter and approximately 8mm high. With the casing, the wearable has a diameter of 28.5mm and a height of 11mm. In addition, there is a mounting point along the edge of the wearable so that it could be connected to a key ring or something similar. With this design it is small enough to be used as a wearable, and can be worn in many ways.



(a) A top view of the engineering prototype.

(b) A side 3D view of the engineering prototype.

(c) A bottom 3D view of the engineering prototype.

Figure 9.4: The engineering prototype PCB designed in Altium Designer.

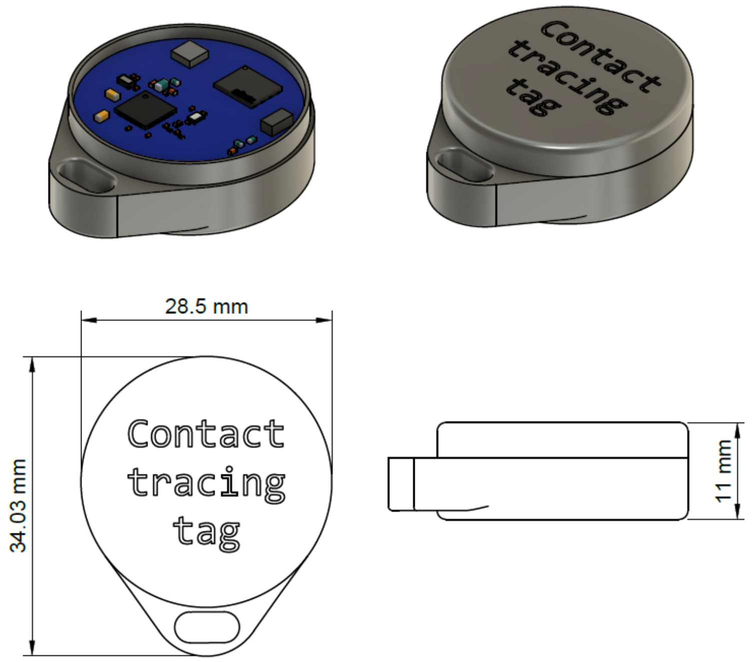


Figure 9.5: A production drawing of the engineering prototype with its tag-formed shape.

9.3.1 Battery Holder

The battery holder is the largest component of the wearable and can probably be replaced by building the casing of the wearable to hold the battery itself. The AirTag is an example of a solution where the casing serves as a battery holder [23]. The battery holder is not especially bulky, but the diameter could possibly be reduced by a couple of millimeters, and the height might be reduced as well.

9.3.2 Remove Optional Crystal

In the circuitry schematics for the engineering prototype shown in Appendix G, one can see that there is an optional crystal oscillator. It can be removed to save cost, since crystal oscillators are costly, and it would reduce the size of the wearable. The reason it is included in the design is that it significantly increases the accuracy of the clock. The internal oscillator has an accuracy of ±500ppm, while the optional one has an accuracy of ±40ppm. In 30 days 2592000 seconds pass. With an accuracy of ±500ppm, the time can fluctuate up to 1296s (21 minutes 36 seconds). While with an accuracy of ±40ppm the time can fluctuate up to 104 seconds (1 minute 44 seconds). All the ENS records are timestamped, and have to be accurate within the 10-minute resolution, because the time in GAENS is discretized in 10-minute intervals. If the error is corrected each time the wearable is connected to a client, both with and without the optional crystal oscillator should be accurate enough.

9.3.3 Visual Feedback

As the design of the wearable is minimalist to save cost and power consumption, there is no visual feedback from it. This means that one needs to add a feedback solution to enable the user to find out if the device is powered on, and have battery capacity left. It could for example be in the form of Light Emitting Diodes (LEDs), or the BAS. Either way, some additional changes have to be done to the PCB, and it would add to the current consumption.

An issue with LEDs is that it has to be visible to the user. That implies that the casing has to be adapted so that the LEDs are visible. LEDs does not show the battery capacity, but it could serve as an indicator for showing if the wearable is powered on or not. The LED can be toggled on and off, so that the average current will be low.

The BAS needs a way of measuring the voltage of the coin cell battery to tell the battery capacity. This can be done using different methods, such as using the comparator, low-power comparator or the analog-to-digital converter on the nRF52833. Using the BAS, together with one of the methods for measuring voltage, one could check the battery capacity by connecting to the wearable.

9.3.4 Antenna Placement

The antenna has some restrictions regarding its placements. All the constraints come from the 2450AT18A100E datasheet [25]. In the schematics design it was made sure that the antenna was matched to 50Ω with its matching circuit, but another requirement is that the antenna has to be placed at some distance to the ground plane. It should also not be any power or signal planes beneath the antenna either. In the datasheet the antenna is placed on a square corner with 4mm distance to the top layer ground plane. For the circular wearable, it is not possible to have a square at the edge of the board. The design of the antenna is shown in Figure 9.6.

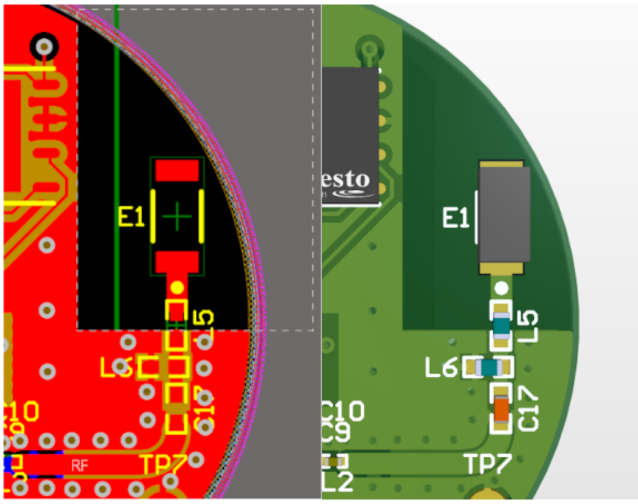


Figure 9.6: On the left the antenna PCB design is shown from 2D mode in Altium Designer, and on the right the same design is shown in 3D mode.

The antenna is placed 3mm away from the top layer ground plane. Deviating from the reference layout introduces risk, as it is likely that the antenna performance will suffer. However, in this application, the range of the communication is not a key parameter, so a trade-off is possible. Although, the antenna performance and electromagnetic emissions will have to be confirmed by measurements.

Chapter 10

Future Work

In this chapter, possible future tasks that have to be investigated in the future is presented.

10.1 Finish WENS Implementation

The WENS needs to be fully implemented in a final solution. This is something that could to be done when a final version of the service is published by Bluetooth SIG.

10.2 Test Advertising and Scanning Parameters

In this thesis it is done estimates on what advertising and scanning parameters for both the WENS and GAENS should use. These are subject to be changed by Google, Apple or Bluetooth SIG in the future. For now there is room for individual implementations, so more practical testing in public should be conducted to find the optimal parameters so that the contact tracing is reliable, but does not use excessive power.

10.3 Finish Wearable Hardware Design

According to the product development timeline, the engineering prototype needs to be manufactured and tested so that a production prototype can be designed in the future, and later the pilot.

10.4 Accelerometer Interrupt Trigger Parameters

For a contact tracing wearable with an accelerometer to be beneficial, the wearable has to use the correct interrupt trigger parameters for the accelerometer. These parameters have to be found through testing.

10.5 Transmit Power

Whether the transmit power can be lower than 0dBm for connectable WENS advertising, non-connectable GAENS advertising and connections has to be investigated. It is possible to make a considerable amount of current consumption savings by reducing the transmit power.

10.6 Device Firmware Update

For a final product it would be beneficial, or necessary, to add support for BLE device firmware update so that the wearable's software can be updated without having to connect physically to the device.

10.7 Antenna

The antenna and its circuitry have to be tested and modified if necessary.

Chapter 11

Conclusion

An engineering prototype of a low-power, non-internet-connected wearable for contact tracing, based on GAENS and WENS, has been developed and optimized. The software were tested on a PoC prototype, and it had a current consumption of approximately 532µA. The optimization analysis of the engineering prototype indicates that it should be possible to achieve an average current consumption of 42.17µA, with the optimizations suggested in this thesis. By powering the wearable with a CR2032 coin cell battery, one can expect a lifetime of the contact tracing wearable of about 197 days. This shows that it should be possible to achieve a battery life of several months.

The PCB for the engineering prototype was designed with a circular shape, and has a diameter of 26mm and a height of approximately 8mm with its components. The wearable has a nRF52833 BLE SoC, a 32Mb external flash memory, an accelerometer and a battery holder for a CR2032 coin cell battery. A casing for the PCB was also designed in the shape of a circular tag, and is 28.5mm in diameter and has a height of 11mm. Indicating that the design is small enough to be used as a wearable.

The software and hardware developed for the engineering prototype contributes to realizing a wearable that can participate in already existing ENSs and reduce the spread of infectious diseases like COVID-19.

Bibliography

[1] Bluetooth SIG, *Wearable exposure notification service*, `https://www.bluetooth.com/wp-content/uploads/2020/12/WENS_2020-12-03.pdf` [Accessed: 27.03.2021], Dec. 2020.

[2] Folkehelseinstituttet (FHI), *Om smittestopp*, `https://www.fhi.no/om/smittestopp/om-smittestopp/` [Accessed: 03.05.2021], Dec. 2020.

[3] Google and Apple, *Exposure notification bluetooth specification*, `https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf` [Accessed: 22.05.2021], Apr. 2020.

[4] World Health Organization (WHO), *Listings of who's response to covid-19*, `https://www.who.int/news/item/29-06-2020-covidtimeline` [Accessed: 11.06.2021], Jun. 2020.

[5] P. Wang, 'Mechanisms of sars-cov-2 transmission and pathogenesis,' *Trends in Immunology*, Oct. 2020. DOI: `10.1016/j.it.2020.10.004`.

[6] X. He, E. Lau, P. Wu, X. Deng, J. Wang, X. Hao, Y. Lau, J. Y. Wong, Y. Guan, X. Tan, X. Mo, Y. Chen, B. Liao, W. Chen, F. Hu, Q. Zhang, M. Zhong, Y. Wu, L. Zhao and G. Leung, 'Temporal dynamics in viral shedding and transmissibility of covid-19,' *Nature Medicine*, vol. 26, May 2020. DOI: `10.1038/s41591-020-0869-5`.

[7] World Health Organization (WHO), *Contact tracing in the context of covid-19*, `https://apps.who.int/iris/bitstream/handle/10665/332049/WHO-2019-nCoV-Contact_Tracing-2020.1-eng.pdf` [Accessed: 03.06.2021], May 2020.

[8] V. Shubina, A. Ometov and E. S. Lohan, 'Technical perspectives of contact-tracing applications on wearables for covid-19 control,' Oct. 2020. DOI: `10.1109/ICUMT51630.2020.9222246`.

[9] Google and Apple, *Exposure notification cryptography specification*, `https://blog.google/documents/69/Exposure_Notification_-_Cryptography_Specification_v1.2.1.pdf` [Accessed: 22.05.2021], Apr. 2020.

[10] Bluetooth SIG, *Device information service 1.1*, `https://www.bluetooth.com/specifications/specs/device-information-service-1-1/` [Accessed: 03.06.2021], Nov. 2011.

[11] Bluetooth SIG, *Device time service 1.0*, `https://www.bluetooth.com/specifications/specs/device-time-service-1-0/` [Accessed: 03.06.2021], Dec. 2020.

[12] The Linux Foundation, *The zephyr project*, `https://www.zephyrproject.org/`.

[13] P. Hoffman and K. Furset, *High pulse drain impact on cr2032 coin cell battery capacity*, https://www.dmcinfo.com/Portals/0/Blog%20Files/High%20pulse%20drain%20impact%20on%20CR2032%20coin%20cell%20battery%20capacity.pdf [Accessed: 06.05.2021], 2011.

[14] Inertia Engineering, *The three phases of product prototyping*, https://inertiaengineering.com/three-phases-prototyping/ [Accessed: 11.03.2021].

[15] Nordic Semiconductor, *Online power profiler for ble*, https://devzone.nordicsemi.com/nordic/power/w/opp/2/online-power-profiler-for-ble [Accessed: 19.04.2021].

[16] Micron, *N25q032a13esc40g*, https://www.micron.com/products/nor-flash/serial-nor-flash/part-catalog/n25q032a13esc40g [Accessed: 23.03.2021], May 2018.

[17] Nordic Semiconductor, *Nrf52833*, https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52833.html [Accessed: 10.06.2021], Jun. 2021.

[18] Winbond, *W25q32jv*, https://www.winbond.com/resource-files/w25q32jv%20revg%2003272018%20plus.pdf [Accessed: 08.04.2021], Mar. 2018.

[19] STMicroelectronics, *Lis331dlh*, https://media.digikey.com/pdf/Data%20Sheets/ST%20Microelectronics%20PDFS/LIS331DLH(TR).pdf?src-supplier=Digi-Key [Accessed: 04.05.2021], Jul. 2009.

[20] D. J. Leith and S. Farrell, 'Measurement-based evaluation of google/apple exposure notification api for proximity detection in a commuter bus,' *PLOS ONE*, vol. 16, no. 4, pp. 1–16, Apr. 2021. DOI: 10.1371/journal.pone.0250826. [Online]. Available: https://doi.org/10.1371/journal.pone.0250826.

[21] M. Ghamari, E. Villeneuve, C. Soltanpur, J. Khangosstar, B. Janko, R. S. Sherratt and W. Harwin, 'Detailed examination of a packet collision model for bluetooth low energy advertising mode,' *IEEE Access*, vol. 6, pp. 46 066–46 073, 2018. DOI: 10.1109/ACCESS.2018.2866323.

[22] Q. A. Budan, A. Naderi and D. L. Deugo, 'Range of bluetooth low energy beacons in relation to their transmit power,' *Int'l Conf. Internet Computing and Internet of Things | ICOMP'17 |*, 2017, https://csce.ucmss.com/cr/books/2017/LFS/CSREA2017/ICM3063.pdf [Accessed: 09.05.2021].

[23] iFixit, *Airtag teardown: Yeah, this tracks*, https://www.ifixit.com/News/50145/airtag-teardown-part-one-yeah-this-tracks [Accessed: 10.05.2021], May 2021.

[24] Altium Designer, *Embedded rf design: Ceramic chip antennas vs. pcb trace antennas*, https://resources.altium.com/p/embedded-rf-design-ceramic-chip-antennas-vs-pcb-trace-antennas [Accessed: 08.04.2021], Feb. 2018.

[25] Johanson Technology, *2450at18a100*, https://www.mouser.com/datasheet/2/611/2450AT18A100-277016.pdf [Accessed: 04.05.2021], Jul. 2018.

[26] Nordic Semiconductor, *Nordic thingy:91*, https://infocenter.nordicsemi.com/topic/ug_thingy91/UG/thingy91/hw_description/nRF52840.html [Accessed: 11.03.2021], Nov. 2020.

[27] Adesto Technologies, *At45db321e*, `http://www.adestotech.com/wp-content/uploads/doc8784.pdf?src-supplier=Digi-Key` [Accessed: 11.03.2021], Mar. 2019.

[28] Harwin, *S8421-45r*, `https://www.harwin.com/products/S8421-45R/` [Accessed: 10.06.2021].

[29] C. Gomez, J. Oller Bosch and J. Paradells, 'Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology,' *Sensors (Basel, Switzerland)*, vol. 12, pp. 11 734–53, Dec. 2012. DOI: `10.3390/s120911734`.

[30] K. Townsend, C. Cufí, Akiba and R. Davidson, *Getting Started with Bluetooth Low Energy*. Sebastopol, CA, 2014, ISBN: 1491949511.

Appendix A

Bluetooth Low Energy Overview

This is an excerpt from the specialization project, covering theory about Bluetooth Low Energy (BLE).

A.1 Bluetooth Low Energy

BLE is the low power version of Bluetooth. It is used in applications where energy consumption is important. BLE is a wireless communication protocol operating in the 2.4GHz Industrial, Scientific and Medical (ISM) band [29]. The ISM band is unlicensed and legal to use worldwide. The BLE protocol is widely used for Internet of Things applications as well as for beacons, computer peripherals, lighting, gaming peripherals and wearables.

A.1.1 Generic Access Profile

The Generic Access Profile (GAP) defines the general topology of the BLE network stack [29]. It controls connections and advertising in BLE. It is what makes the device visible and determines how two devices can interact with each other. GAP specifies four different roles; *central*, *peripheral*, *broadcaster* and *observer*.

The broadcaster role sends non-connectable advertising packets periodically to anyone willing to receive them. Broadcasting makes it possible to transmit data to multiple devices at the same time. The observer role, on the other hand, repeatedly scans for non-connectable advertising packets. A central device is a device that discovers and listens to other devices that are advertising. The central device can connect to a peripheral device. While a peripheral device is a device that advertises and accepts connections from central devices. A device can operate in multiple roles at the same time.

A.1.2 Connections

If a central device observes a peripheral device when scanning, a *connection request packet* from the central device can be sent. This is called initiating. This triggers the forming of a connection between the two devices if the request is accepted. The connection is *established* once the device receives a packet from the peer device. The central is controlling the parameters and the timing of events within the connection. A connection is simply exchange of data between a central and a peripheral at predefined intervals.

A *connection event* is when the central and the peripheral device exchange data by sending data packets to each other. The *connection events* lasts until neither of the two has more data to send.

Three of the most important parameters that defines the connection is *connection interval*, *slave latency* and *supervision timeout*. The connection interval defines the interval in which two BLE devices will wake up the radio and exchange data. Slave latency is the parameter which allows a peripheral device to skip *connection events*. The value of the parameter is the number of consecutive connection events it is allowed to skip before compromising the connection. The supervision timeout parameter is the maximum time between two received data packets before the connection is considered lost.

Connections are for transmitting data in both directions. Also, broadcasting only supports sending two advertising payloads. Connections, on the other hand, can periodically exchange a larger amount of data.

A.1.3 Advertising

When advertising, a device sends out packages of useful data for other devices to receive and process. There are 40 channels in BLE, and each of them is separated by 2MHz (center-to-center). Three of the channels are reserved for advertising, as shown in Figure A.1. Advertising is performed on the three channels reserved for advertising, one at a time. Advertising packets are used for two things. It is used for broadcasting data to applications which does not need a full connection establishment, and it is also used to discover peripherals and connect to them.

According to "*Getting started with Bluetooth Low Energy*" on can classify advertising packets according to three properties; *connectability*, *scannability* and *directability* [30].

Connectability:

- **Connectable**: A scanner can initiate a connection upon reception of an advertising packet.
- **Non-connectable**: A scanner can not initiate a connection upon reception of an advertising packet.

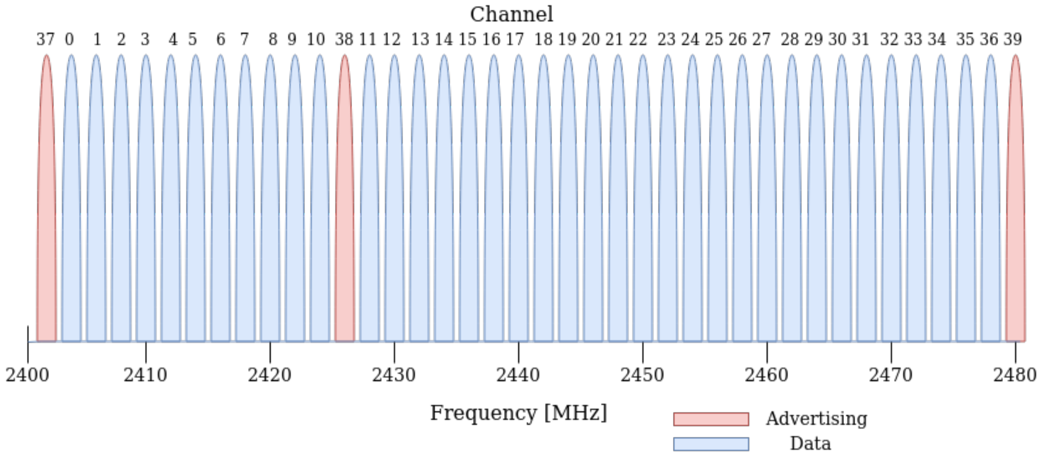


Figure A.1: The BLE channels.

Scannability:

- **Scannable**: A scanner can issue a scan request upon a reception of an advertising packet.
- **Non-scannable**: A scanner can not issue a scan request upon reception of an advertising packet.

Directability:

- **Directed**: A directed advertising packet contains the advertiser's and the scanners Bluetooth address in its payload, and no user data. This means it is connectable.
- **Undirected**: An undirected advertising packet can contain user data in its payload, and is not targeted to a specific scanner.

A.1.4 Scanning

When scanning, a device listens to the three advertising channels one at a time. For a scanning device to discover another device, the device has to scan the channel the other device is advertising on. This has to happen simultaneously. In order to increase the possibility for this to happen, or to make it happen quicker, the scanning and advertisement parameters can be adjusted. There exists two types of scanning procedures, *passive scanning* and *active scanning*. Using passive scanning, the advertiser is never aware of if any scanners actually received the packets. In other words, the scanner does not respond to advertising packets. While active scanning, on the other hand, responds to the advertising packets with a *scan response packet*. This makes it possible for the advertiser to double the effective payload it can transfer to the scanner.

A.1.5 Generic Attribute Profile

The Generic Attribute Profile (GATT) describes the details of how data is transferred once devices have a BLE connection [29]. It defines how two BLE devices transfer data using *services* and *characteristics* by using the generic data protocol called Attribute Protocol (ATT). ATT is used to store services, characteristics and related data using a lookup table with 16-bit IDs for each entry in the table. The IDs are Universally Unique Identifiers (UUIDs) which is used to describe and determine the services, characteristics or types that are available. GATT is used after a connection is established. There are two roles defined by GATT, *server* and *client*. The server is the device that exposes its data, while a client interfaces with the server and reads or controls the servers' behavior.

Attributes: The attributes are the data that is exposed by a server and defines the structure of this data. Services and characteristics are types of attributes

Services: The services is a grouping of one or more attributes. Services are meant to group together related attributes that satisfies the functionality of the server.

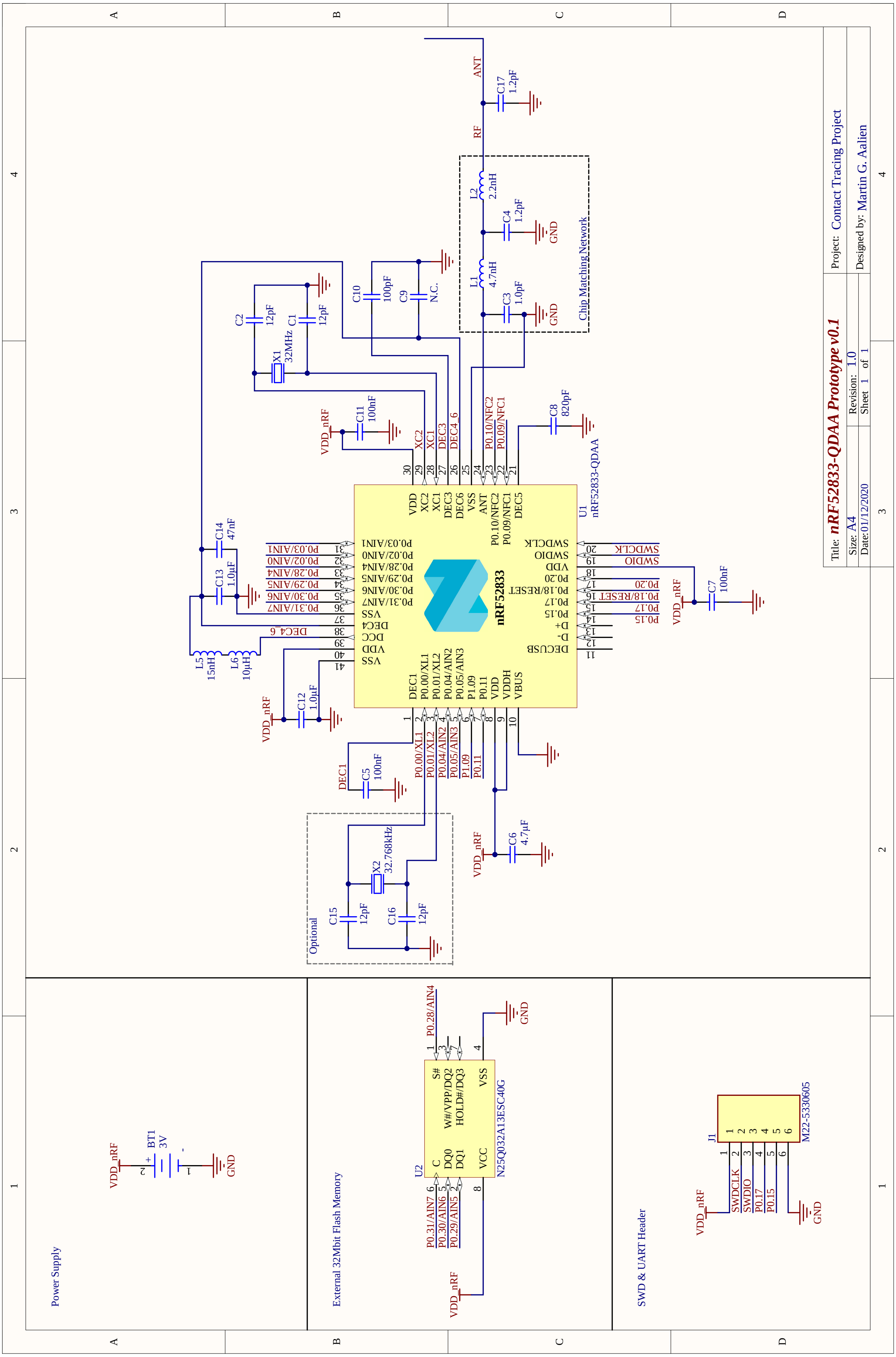
Characteristic: The characteristic is the part of a service that represents the data or information that the server wants to expose to the client.

Profiles: Profiles are a broader definition from services. They define the behavior of the client and server. Everything from services, characteristics, connections and security.

Appendix B

PoC Prototype Schematics

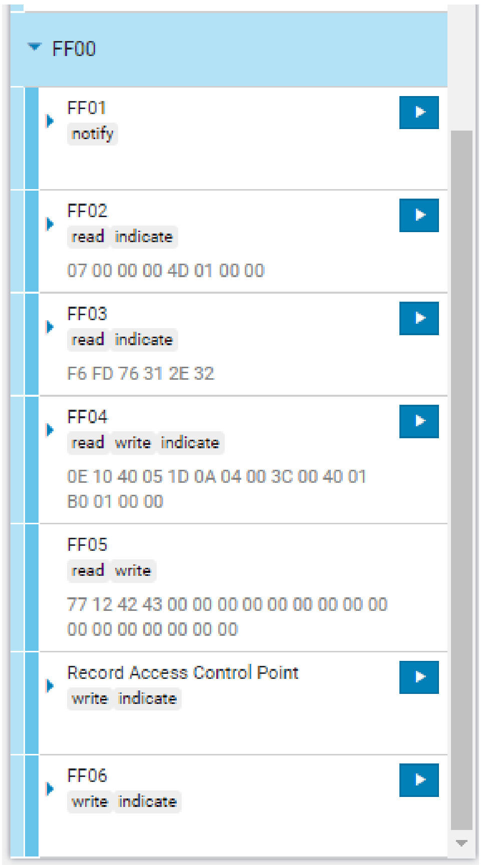
On the following page, the PoC prototype schematics is shown. The circuit diagram has been designed in Altium Designer.


Title: nRF52833-QDAA Prototype v0.1
Project: Contact Tracing Project
Designed by: Martin G. Aalien
Size: A4
Revision: 1.0
Date: 01/12/2020
Sheet 1 of 1
U1
nRF52833-QDAA
nRF52833
Chip Matching Network
ANT
RF
C17 1.2pF
L2 2.2nH
C4 1.2pF
L1 4.7nH
C3 1.0pF
GND
C10 100pF
C9 N.C.
C2 12pF
C1 12pF
X1 32MHz
C11 100nF
C8 820pF
VDD_nRF
XC2
XC1
DEC3
DEC4_6
P0.10/NFC2
P0.09/NFC1
C14 47nF
C13 1.0µF
L5 15nH
L6 10µH
C12 1.0µF
DEC1
C5 100nF
C6 4.7µF
C7 100nF
SWDCLK
SWDIO
P0.20
P0.18/RESET
P0.17
P0.15
P0.03/AIN1
P0.02/AIN0
P0.28/AIN4
P0.29/AIN5
P0.30/AIN6
P0.31/AIN7
P0.00/XL1
P0.01/XL2
P0.04/AIN2
P0.05/AIN3
P1.09
P0.11
VSS
DEC4
DCC
VDD
DECUSB
D+
D-
VDDH
VBUS
DEC5
DEC6
Optional
X2 32.768kHz
C15 12pF
C16 12pF
Power Supply
BT1 3V
External 32Mbit Flash Memory
U2
N25Q032A13ESC40G
S#
W#/VPP/DQ2
HOLD#/DQ3
C
DQ0
DQ1
VCC
SWD & UART Header
J1
M22-5330605
A
B
C
D
1
2
3
4

Appendix C

BLE Services

C.1 Wearable Exposure Notification Service



FF00

FF01
notify

FF02
read indicate
07 00 00 00 4D 01 00 00

FF03
read indicate
F6 FD 76 31 2E 32

FF04
read write indicate
0E 10 40 05 1D 0A 04 00 3C 00 40 01
B0 01 00 00

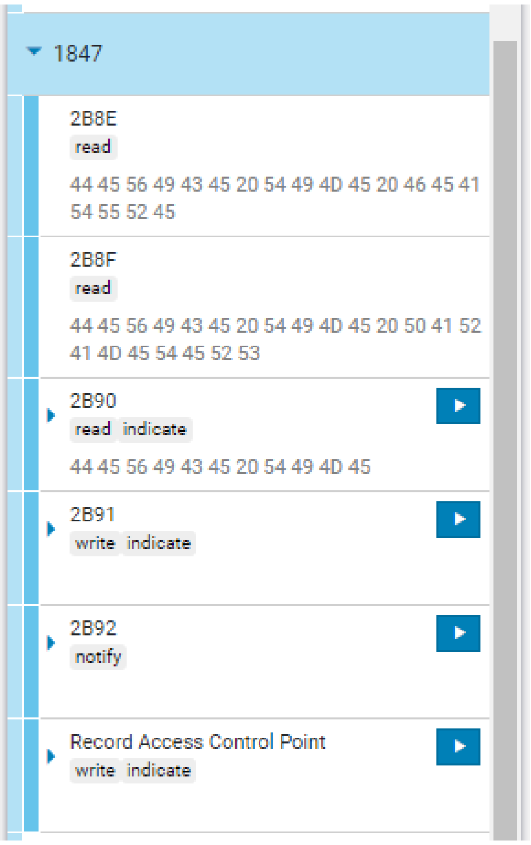
FF05
read write
77 12 42 43 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00

Record Access Control Point
write indicate

FF06
write indicate

Figure C.1: A screenshot from the nRF Connect application showing the WENS and its characteristics. As the WENS specification is still only preliminary, the UUIDs is not defined in the specifications. The UUIDs used for the characteristics is just temporary values used for testing.

C.2 Device Time Service



1847

2B8E
read
44 45 56 49 43 45 20 54 49 4D 45 20 46 45 41 54 55 52 45

2B8F
read
44 45 56 49 43 45 20 54 49 4D 45 20 50 41 52 41 4D 45 54 45 52 53

2B90
read indicate
44 45 56 49 43 45 20 54 49 4D 45

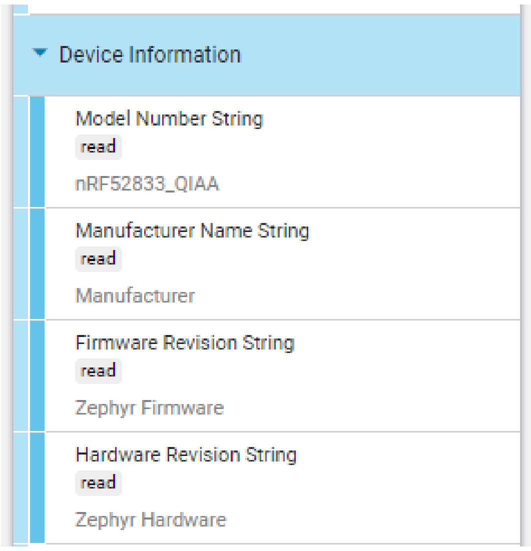
2B91
write indicate

2B92
notify

Record Access Control Point
write indicate

Figure C.2: A screenshot from the nRF Connect application showing the DTS and its characteristics. The service came in 2020, so the nRF Connect application does not recognize the UUIDs because it is quite new [11].

C.3 Device Information Service



Device Information

Model Number String
read
nRF52833_QIAA

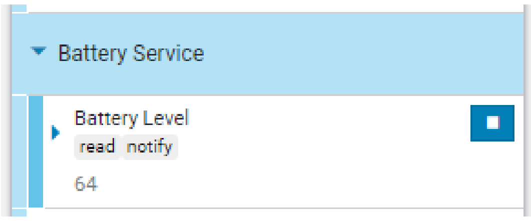
Manufacturer Name String
read
Manufacturer

Firmware Revision String
read
Zephyr Firmware

Hardware Revision String
read
Zephyr Hardware

Figure C.3: A screenshot from the nRF Connect application showing the DIS and its characteristics. The characteristics hold temporary values set by me, and can easily be updated.

C.4 Battery Service



Battery Service

Battery Level
read notify
64

Figure C.4: A screenshot from the nRF Connect application showing a temporary implementation of the BAS. The Battery Level characteristic holds the number of what percentage is left of the battery capacity (0x64=100%). This service is not mandatory according to the WENS specification, and is not discussed further [1].

Appendix D

Rotation of RPI, AEM and BLE Address

Code listing D.1: The C function for rotating the RPI, AEM and BLE address. It also chacks if the TEK has expired and updates the keys if it has.

```c
/**
 * @brief Work handler for changing the RPI, AEM and update advertise data.
 *
 * @param unused Not in use, but required.
 */
static void _rotate_rpi_handler(struct k_work *unused)
{
    // Stop advertising
    if (advertise_stop() < 0)
    {
        LOG_ERR("Failed to pause the advertising");
        return;
    }

    // Stop the timer
    k_timer_stop(&_rpi_rotation_timer);

    // Check if the Temporary Exposure Key has expired
    if (gaens_tek_expired() == 1)
    {
        if (gaens_update_keys() < 0)
        {
            LOG_ERR("Failed to update TEK");
            return;
        }
    }

    // Update RPI
    if (gaens_update_rpi() < 0)
    {
        LOG_ERR("Failed to update the RPI");
        return;
    }

    // Encrypt AEM
    uint8_t aem[AEM_LENGTH];
    if (gaens_encrypt_metadata(metadata, AEM_LENGTH, aem) < 0)
    {
        LOG_ERR("Failed to encrypt the metadata");
        return;
    }

    // Change advertise data
    if (advertise_change_gaens_service_data(current_rpi, RPI_LENGTH, aem,
                                            AEM_LENGTH) < 0)
    {
        LOG_ERR("Failed to change the gaens service data to advertise");
        return;
    }

    // Get random rotation interval between 10 and 20 minutes
    uint32_t random_time;
    if (_gaens_random_rotation_interval(&random_time) < 0)
    {
        LOG_ERR("Failed to fetch random rotation interval");
        return;
    }

    // Start a timer that will trigger in random_time
    k_timer_start(&_rpi_rotation_timer, K_SECONDS(random_time),
                  K_SECONDS(random_time));

    // Start advertising
    if (advertise_start() < 0)
    {
        LOG_ERR("Failed to resume advertising");
        return;
    }

    LOG_INF("Successfully updated RPI and AEM");
}
```

Appendix E

Scan Window Simulation

Code listing E.1: A Python script for determining the average number of advertising packets that will be received by the wearable with different scan windows and advertise intervals.

```python
import matplotlib.pyplot as plt
import numpy as np

SCAN_WINDOW = 300        # Scan window (ms)
ADV_INTERVAL_LOWER = 50  # Lower bound advertise interval (ms)
ADV_INTERVAL_UPPER = 500 # Upper bound advertise interval (ms)

result = []

# Simulate all the different advertise intervals from lower bound to upper bound
for adv_interval in range(ADV_INTERVAL_LOWER, ADV_INTERVAL_UPPER):
    # Initialize the total amount of packets found inside the scan
    # window to zero before every simulation
    total = 0

    # Create a list of all packets that possibly can fall inside
    # the scan window
    packets = np.arange(-SCAN_WINDOW, SCAN_WINDOW + 1, adv_interval)

    # Step forward in time until the pattern repeat itself
    for step in range(0, adv_interval):
        # Sum how many of the packets were inside the scan window
        total += sum(0 < x < SCAN_WINDOW for x in packets)

        # Move all packet 1ms forward in time
        packets = [n + 1 for n in packets]

    # Append the average number of packets that were inside the scan interval
    # to result for that advertise interval
    result.append(total / adv_interval)

# Plot graph from the lower bound advertise interval to the upper bound
plt.figure()
plt.title(f"Average Number of Packets Received,\nScan Window = {SCAN_WINDOW}ms")
plt.ylabel(f"Number of Packets")
plt.xlabel(f"Advertise Interval (ms)")
plt.plot(range(ADV_INTERVAL_LOWER,ADV_INTERVAL_UPPER), result)
plt.grid()
plt.savefig(f"avg_packets_received.png")
```

Appendix F

Average Current Consumption

F.1 Accelerometer

BLE SoC				
	Current (mA)	**Duration (ms)**	**Interval (ms)**	**Average Current (µA)**
Non-Connectable GAENS Advertising	3	3.1	205	26.4634
Connectable WENS Advertising	2.7	3.78	1000	5.9535
Scanning	6	300	60000	17.5000
Connections	2.02	180000	86400000	4.2083
Rotation of GAENS Keys and Identifier	3	5.5	900000	0.0183
Sleep	0.0024	84154343.41	86400000	2.3376
				56.4812
External Memory				
Write	20	0.8	60000	0.1556
Read	8	4000	86400000	0.2160
Erase	20	10000	86400000	2.3148
Standby	0.01	166000	86400000	0.0192
Power-Down	0.001	86218848	86400000	0.9979
				3.7035
			Total:	63.1847

Table F.1: The calculation of the average current of each operation isolated operation averaged over all time where the wearable is using an accelerometer to reduce the current consumption. The average current contribution from the accelerometer is 3µA, and it is assumed that the wearable is not in use for 10 hours a day on average.

F.2 Scan and Advertise Interval Adjustments

BLE SoC				
	Current (mA)	**Duration (ms)**	**Interval (ms)**	**Average Current (µA)**
Non-Connectable GAENS Advertising	3	3.1	270	34.4444
Connectable WENS Advertising	2.7	3.78	2000	5.1030
Scanning	6	300	100000	18.0000
Connections	2.02	180000	86400000	4.2083
Rotation of GAENS Keys and Identifier	3	5.5	900000	0.0183
Sleep	0.0024	84804976	86400000	2.3557
				64.1298
External Memory				
Write	20	0.8	60000	0.2667
Read	8	4000	86400000	0.3704
Erase	20	10000	86400000	2.3148
Standby	0.01	166000	86400000	0.0192
Power-Down	0.001	86218848	86400000	0.9979
				3.9690
			Total:	68.0988

Table F.2: The calculation of the average current of each operation isolated operation averaged over all time where the wearable is optimized by adjusting the advertise and scan intervals.

F.3 Transmit Power Reduction

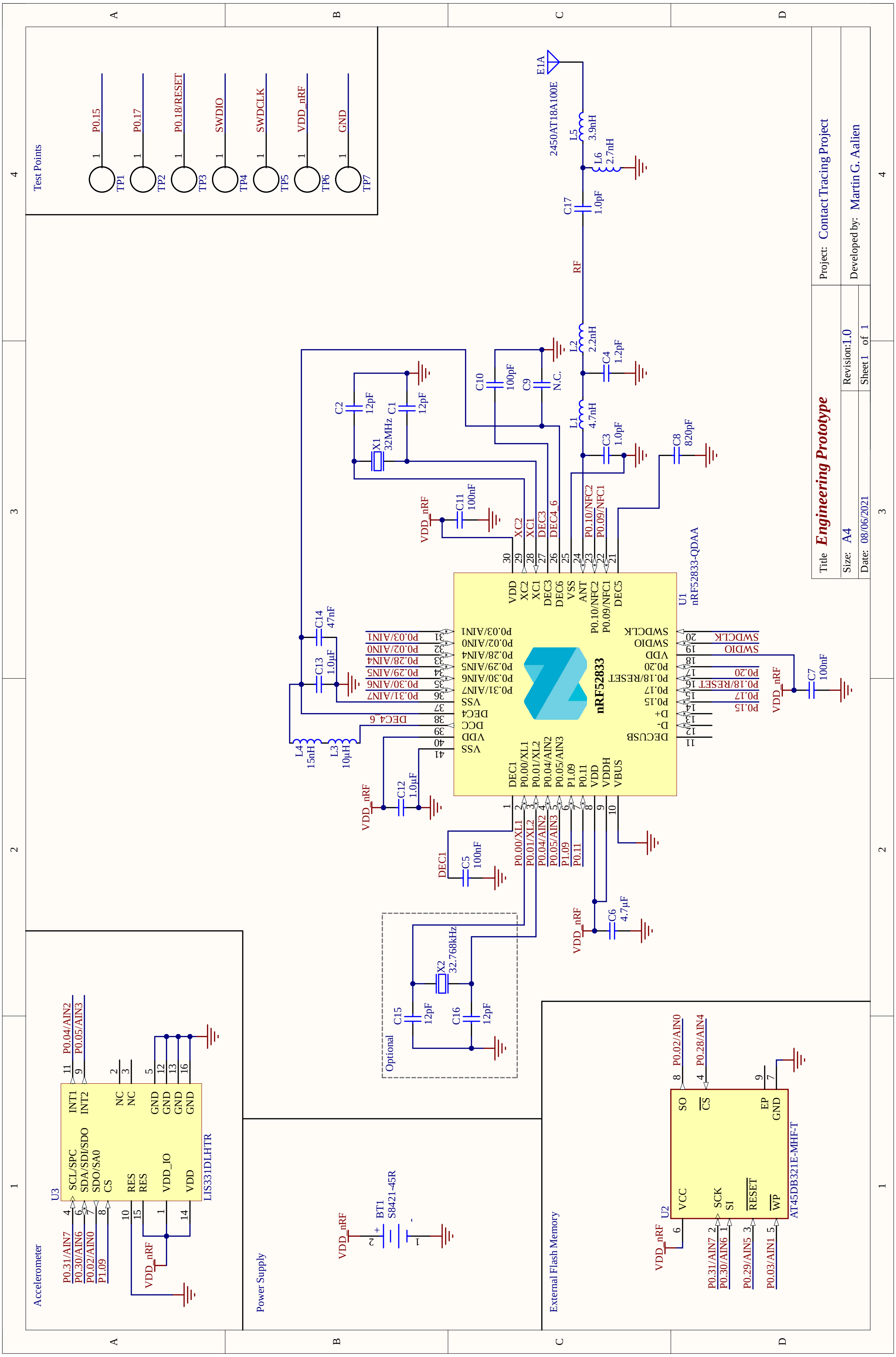
BLE SoC				
	Current (mA)	**Duration (ms)**	**Interval (ms)**	**Average Current (µA)**
Non-Connectable GAENS Advertising	2.3	3.1	205	34.7805
Connectable WENS Advertising	2.7	3.78	1000	10.2060
Scanning	6	300	60000	30.0000
Connections	2.02	180000	86400000	4.2083
Rotation of GAENS Keys and Identifier	3	5.5	900000	0.0183
Sleep	0.0024	84154343.41	86400000	2.3376
				81.5508
External Memory				
Write	20	0.8	60000	0.2667
Read	8	4000	86400000	0.3704
Erase	20	10000	86400000	2.3148
Standby	0.01	166000	86400000	0.0192
Power-Down	0.001	86218848	86400000	0.9979
				3.9690
			Total:	85.5197

Table F.3: The calculation of the average current of each operation isolated operation averaged over all time where the wearable is optimized by reducing the transmit power for GAENS advertising packets from 0dBm to -12dBm.

Appendix G

Engineering Prototype Schematics

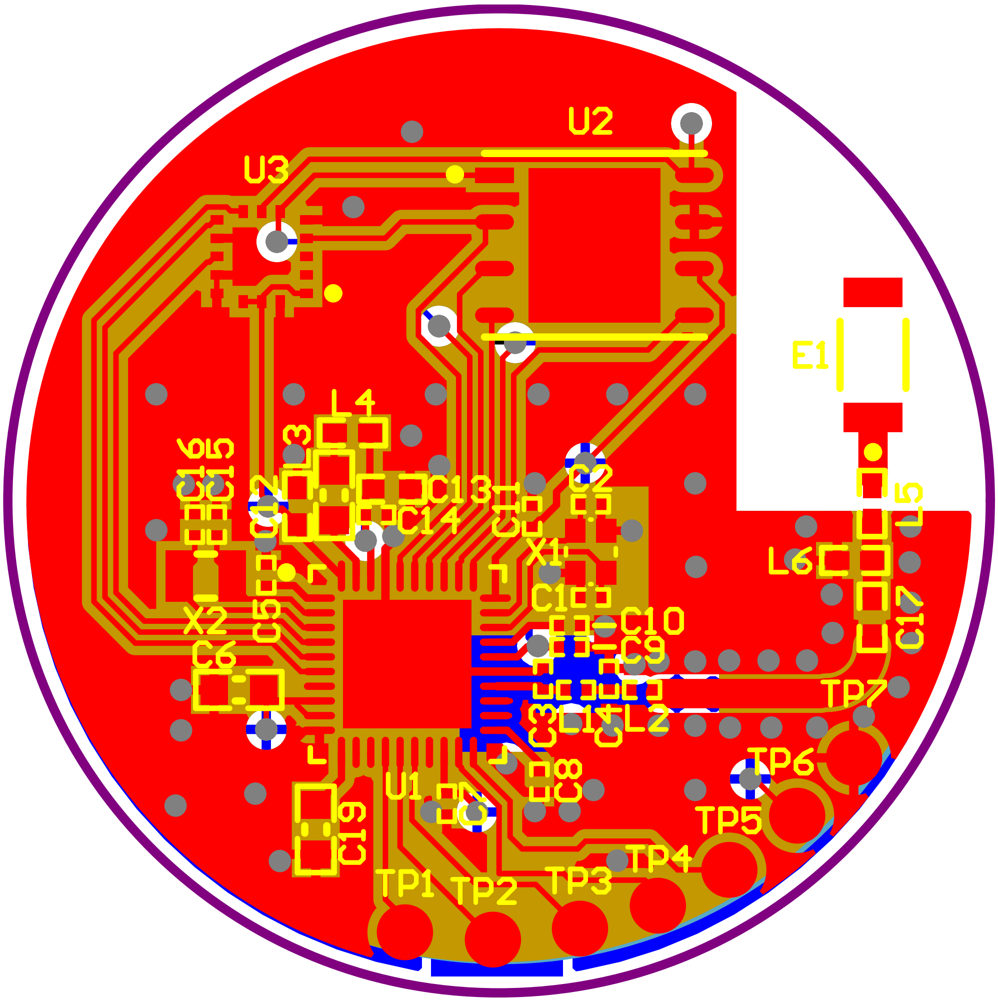
On the following page, the engineering prototype schematics is shown. The circuit diagram has been designed in Altium Designer.


Title Engineering Prototype
Project: Contact Tracing Project
Developed by: Martin G. Aalien
Size: A4
Revision:1.0
Date: 08/06/2021
Sheet 1 of 1
Test Points
TP1 P0.15
TP2 P0.17
TP3 P0.18/RESET
TP4 SWDIO
TP5 SWDCLK
TP6 VDD_nRF
TP7 GND
E1A
2450AT18A100E
L5 3.9nH
L6 2.7nH
C17 1.0pF
RF
L2 2.2nH
C4 1.2pF
L1 4.7nH
C3 1.0pF
C8 820pF
C10 100pF
C9 N.C.
C2 12pF
C1 12pF
X1 32MHz
VDD_nRF
C11 100nF
XC2
XC1
DEC3
DEC4_6
P0.10/NFC2
P0.09/NFC1
U1
nRF52833-QDAA
nRF52833
VDD XC2 XC1 DEC3 DEC6 VSS ANT P0.10/NFC2 P0.09/NFC1 DEC5
P0.03/AIN1 P0.02/AIN0 P0.28/AIN4 P0.29/AIN5 P0.30/AIN6 P0.31/AIN7 VSS DEC4 DCC VDD VSS
SWDCLK SWDIO VDD P0.20 P0.18/RESET P0.17 P0.15 D+ D- DECUSB
DEC1 P0.00/XL1 P0.01/XL2 P0.04/AIN2 P0.05/AIN3 P1.09 P0.11 VDD VDDH VBUS
C14 47nF
C13 1.0µF
DEC4_6
L4 15nH
L3 10µH
C12 1.0µF
C7 100nF
P0.20
P0.18/RESET
P0.17
P0.15
DEC1
C5 100nF
P0.00/XL1
P0.01/XL2
P0.04/AIN2
P0.05/AIN3
P1.09
P0.11
C6 4.7µF
Optional
X2 32.768kHz
C15 12pF
C16 12pF
Accelerometer
U3
LIS331DLHTR
INT1 INT2 NC NC GND GND GND GND
SCL/SPC SDA/SDI/SDO SDO/SA0 CS RES RES VDD_IO VDD
P0.04/AIN2
P0.05/AIN3
P0.31/AIN7
P0.30/AIN6
P0.02/AIN0
P1.09
Power Supply
BT1
S8421-45R
External Flash Memory
U2
AT45DB321E-MHF-T
SO CS EP GND VCC SCK SI RESET WP
P0.02/AIN0
P0.28/AIN4
P0.31/AIN7
P0.30/AIN6
P0.29/AIN5
P0.03/AIN1
A B C D
1 2 3 4

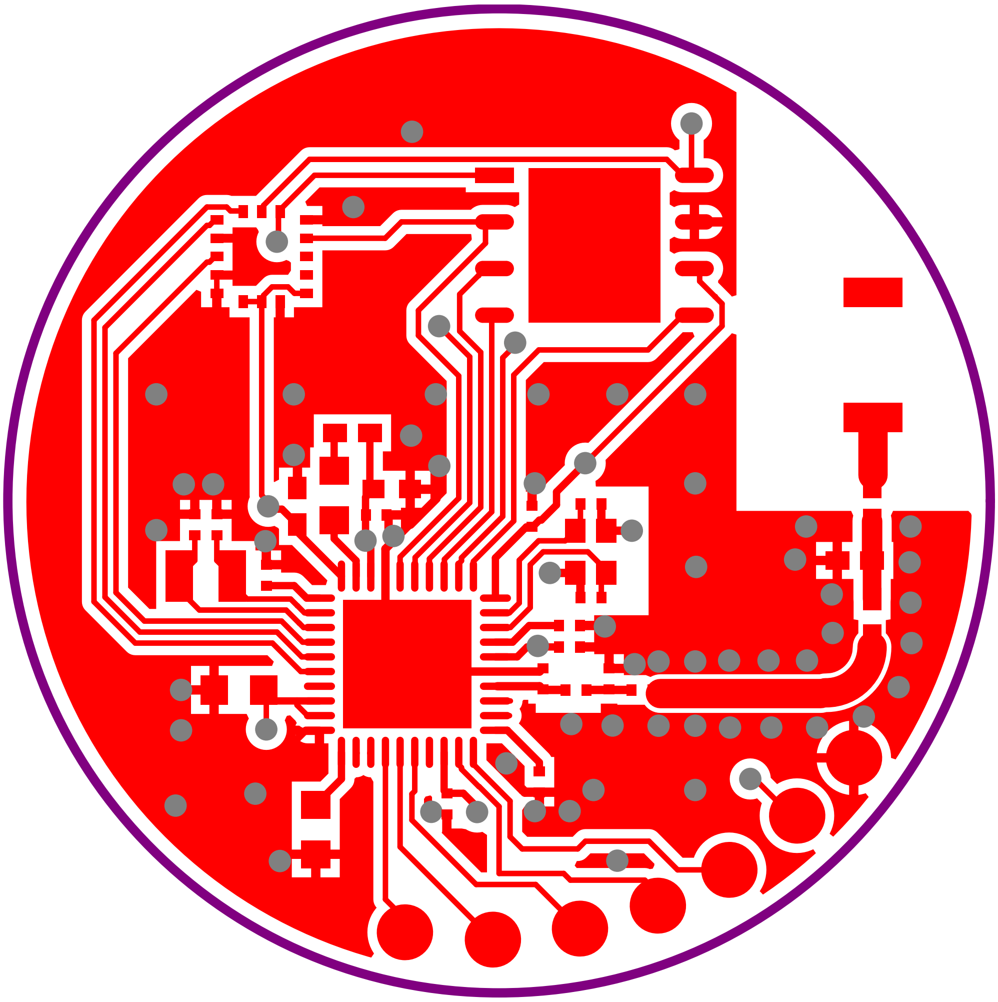
Appendix H

Layers

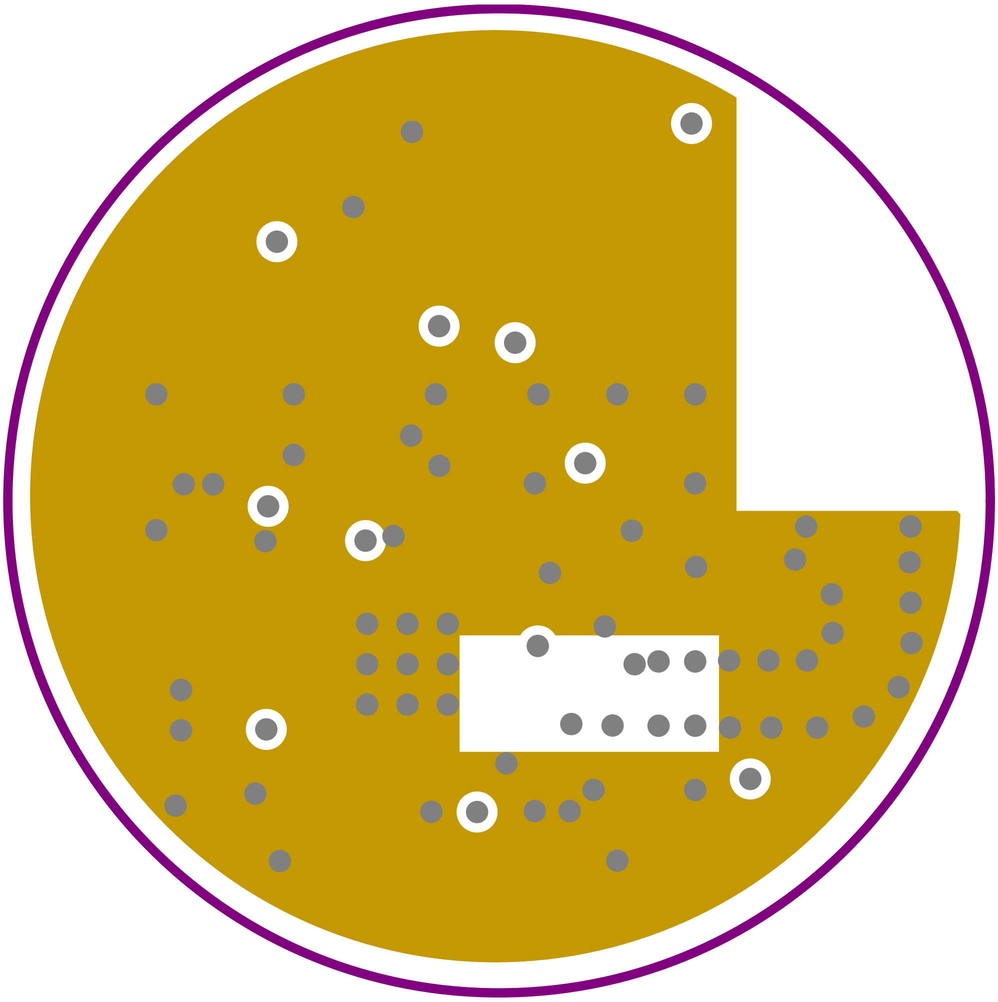
H.1 All Layers



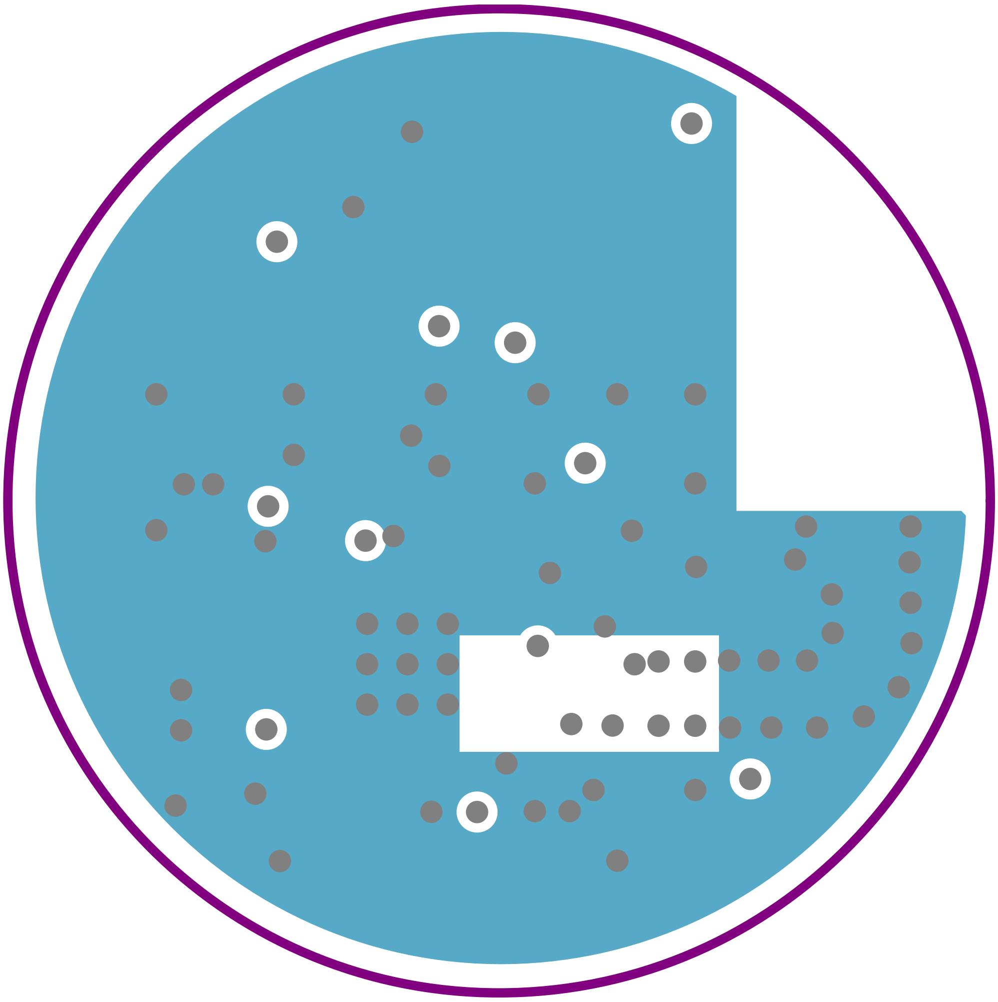
H.2 Top Layer



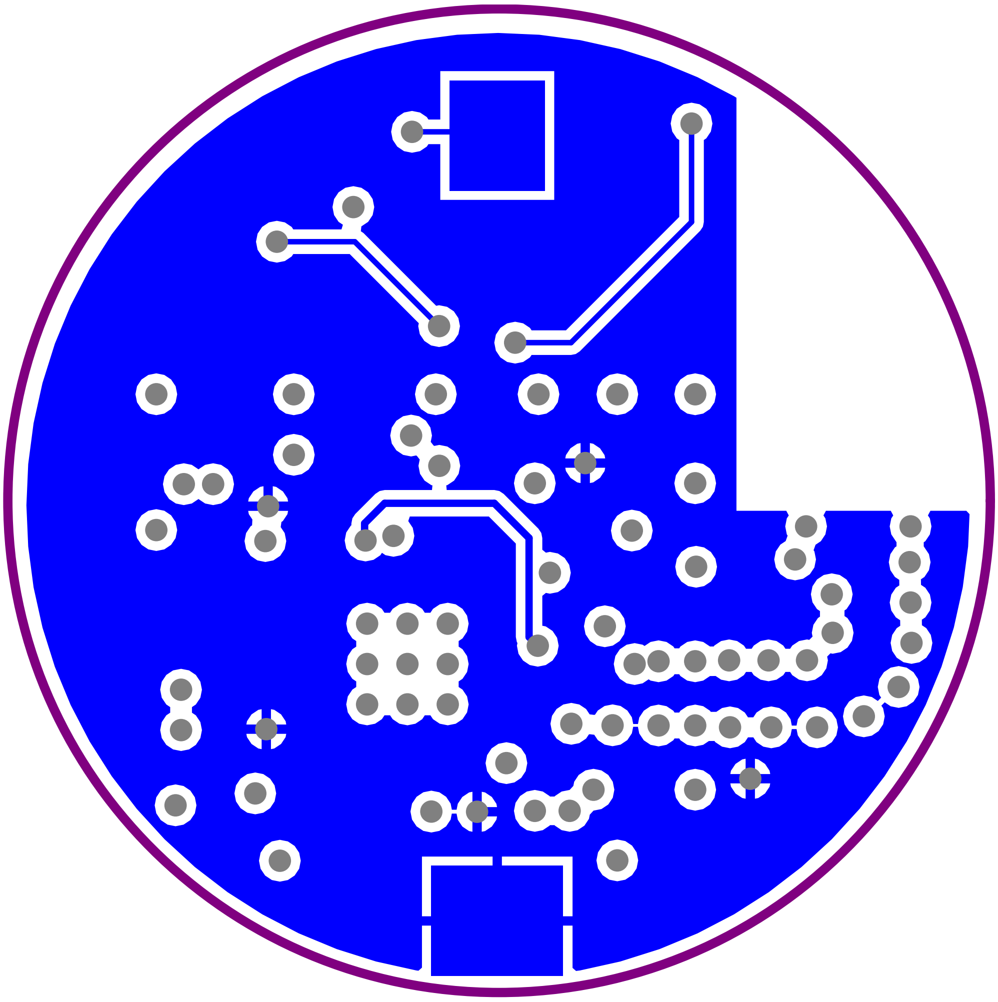
H.3 Mid-Layer 1



H.4 Mid-Layer 2



H.5 Bottom Layer



Appendix I

Code Documentation

The following is the documentation of the software solution based on GAENS and WENS, generated using Doxygen.

GAENS and WENS wearable implementation

v0.1

Generated by Doxygen 1.8.17

Chapter 1

GAENS and WENS wearable implementation

This is an wearable implementation of Google/Apple Exposure Notification System (GAENS) and Wearable Exposure Notification Service (WENS). The implementation is created for Nordic Semiconductor's nRF52833 SoC.

1.1 Install Zephyr

To be able to build this project you need to install the Zephyr devlopment environment. This can be done by following their [Getting Started Guide]("https://docs.zephyrproject.↩
org/latest/getting_started/index.html").

1.2 Comments

This sections includes some comments regarding the implementation.

1.2.1 Serial communication power consumption

Having enabled serial communications disables zephyr to put the device in sleep mode. In order to decrease the power consumption and enable sleep mode one has to change `CONFIG_SERIAL=y` to `CONFIG_SERIAL=n` in `prj.conf`. This overwrites `UART_CONSOLE`, `STDOUT_CONSOLE` and `LOG_BACKEND_UART`, which all will give a warning, but those can be disregarded.

1.2.2 undefined reference to ‘mbedtls_hkdf’

Zephyr uses an own config file for defining MBEDTLS configurations and HKDF, which is required by GAENS, is not included. To fix this error you need to add the following to `modules/crypto/mbedtls/configs/config-tls-generic.h` in your zephyr project:

```
#define MBEDTLS_HKDF_C
```

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 ens_identifier_t Struct Reference

Public Attributes

- uint16_t **uuid**
- char **version** [4]

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.h

4.2 ens_log_t Struct Reference

Public Attributes

- unsigned int **segmentation**: 2
- unsigned int **flags**: 6
- uint8_t ∗ **ens_payload**

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.h

4.3 ens_settings_t Struct Reference

Public Attributes

- uint8_t **data_retention**
- uint8_t **temp_key_length**
- uint16_t **max_key_duration**
- uint8_t **ens_adv_length**
- uint8_t **max_adv_duration**
- uint8_t **scan_on_time**
- uint16_t **scan_off_time**
- uint16_t **min_adv_interval**
- uint16_t **max_adv_interval**
- uint8_t **self_pause_resume**

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.h

4.4 features_t Struct Reference

Public Attributes

- unsigned int **multiple_bonds_supported**: 1
- unsigned int **self_pause_resume_supported**: 1
- unsigned int **self_generation_of_temp_keys**: 1
- unsigned int **rfu**: 13

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.h

4.5 temp_key_list_t Struct Reference

Public Attributes

- uint32_t **timestamp**
- uint8_t **temporary_key** [16]

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.c

4.6 wen_features_t Struct Reference

Public Attributes

- features_t **wen_features**
- uint16_t **storage_capacity**

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.h

4.7 wen_status_t Struct Reference

Public Attributes

- uint8_t **opcode**
- uint8_t **parameter** [18]

The documentation for this struct was generated from the following file:

- src/ble/services/wens/wens.h

Chapter 5

File Documentation

5.1 src/ble/advertise.h File Reference

Advertise library.

```
#include <bluetooth/bluetooth.h>
```

Functions

- int advertise_change_gaens_service_data (uint8_t ∗rpi, uint8_t rpi_length, uint8_t ∗aem, uint8_t aem_length)

  Function for changing the GAENS service data to advertise.
- int advertise_start ()

  Function for starting to advertise.
- int advertise_stop ()

  Function for stopping to advertise.

5.1.1 Detailed Description

Advertise library.

This is a library for controlling the BLE advertising.

5.1.2 Function Documentation

5.1.2.1 advertise_change_gaens_service_data()

```
int advertise_change_gaens_service_data (
            uint8_t * rpi,
            uint8_t rpi_length,
            uint8_t * aem,
            uint8_t aem_length )
```

Function for changing the GAENS service data to advertise.

Parameters

rpi	A pointer to the Rolling Proximity Identifier.
rpi_length	The length of the RPI.
aem	A pointer to the Associated Encrypted Metadata.
aem_length	The length of the AEM.

Returns

int Returns 0 on success, negative otherwise.

5.1.2.2 advertise_start()

```
int advertise_start ( )
```

Function for starting to advertise.

Returns

int Returns 0 on success, negative otherwise.

5.1.2.3 advertise_stop()

```
int advertise_stop ( )
```

Function for stopping to advertise.

Returns

int Returns 0 on success, negative otherwise.

5.2 src/ble/ble.h File Reference

Bluetooth Low Energy library.

Functions

- int ble_init (void)

  Function for initializing the Bluetooth Subsystem.

5.2.1 Detailed Description

Bluetooth Low Energy library.

This is a library for initializing the BLE communication of the device.

5.2.2 Function Documentation

5.2.2.1 ble_init()

```
int ble_init (
            void  )
```

Function for initializing the Bluetooth Subsystem.

Returns

int Returns 0 on success, negative otherwise

5.3 src/ble/connection.c File Reference

USB device core layer APIs and structures.

```
#include "connection.h"
#include "advertise.h"
#include "scan.h"
#include "services/wens/wens.h"
#include <stddef.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/conn.h>
#include <bluetooth/hci.h>
#include <logging/log.h>
#include <sys/util.h>
#include <zephyr/types.h>
```

Macros

- #define **LOG_MODULE_NAME** connection

Functions

- **LOG_MODULE_REGISTER** (connection)
- static void **_connected** (struct bt_conn ∗connected, uint8_t err)
- static void **_disconnected** (struct bt_conn ∗disconn, uint8_t reason)
- void connection_init ()

  Function for initializing the connection module.

Variables

- static struct bt_conn ∗ **conn**
- static struct bt_conn_cb **conn_callbacks**

5.3.1 Detailed Description

USB device core layer APIs and structures.

This file contains the USB device core layer APIs and structures.

5.3.2 Variable Documentation

5.3.2.1 conn_callbacks

```
struct bt_conn_cb conn_callbacks  [static]
```

Initial value:

```
= {
    .connected = _connected,
    .disconnected = _disconnected,
}
```

5.4 src/ble/connection.h File Reference

Connection handling module.

Functions

- void connection_init ()

  Function for initializing the connection module.

5.4.1 Detailed Description

Connection handling module.

This is a module for handling BLE connections.

5.5 src/ble/scan.h File Reference

Bluetooth Low Energy scan module.

```
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
```

Functions

- void scan_set_parameters (struct bt_le_scan_param parameters)

  Function for changing scan parameters.
- int scan_start ()

  Function for starting to scan.
- int scan_stop ()

  Function for stopping to scan.

5.5.1 Detailed Description

Bluetooth Low Energy scan module.

This is a module for controlling the BLE scanning.

5.5.2 Function Documentation

5.5.2.1 scan_set_parameters()

```
void scan_set_parameters (
             struct bt_le_scan_param parameters )
```

Function for changing scan parameters.

Parameters

parameters	Scan parameters

5.5.2.2 scan_start()

```
int scan_start ( )
```

Function for starting to scan.

Returns

int Returns 0 on success, negative otherwise

5.5.2.3 scan_stop()

```
int scan_stop ( )
```

Function for stopping to scan.

Returns

int Returns 0 on success, negative otherwise

5.6 src/ble/services/bs/bas.h File Reference

Battery Service library.

```
#include <zephyr/types.h>
```

Functions

- uint8_t bt_bas_get_battery_level (void)

  Read battery level value.
- int bt_bas_set_battery_level (uint8_t level)

  Update battery level value.

5.6.1 Detailed Description

Battery Service library.

This is a library for the BLE Battery Service.

5.6.2 Function Documentation

5.6.2.1 bt_bas_get_battery_level()

```
uint8_t bt_bas_get_battery_level (
            void )
```

Read battery level value.

Read the characteristic value of the battery level

Returns

The battery level in percent.

5.6.2.2 bt_bas_set_battery_level()

```
int bt_bas_set_battery_level (
            uint8_t level )
```

Update battery level value.

Update the characteristic value of the battery level This will send a GATT notification to all current subscribers.

Parameters

level	The battery level in percent.

Returns

Zero in case of success and error code in case of error.

5.7 src/ble/services/dis/dis.h File Reference

Device Information Service.

5.7.1 Detailed Description

Device Information Service.

5.8 src/ble/services/dts/dts.h File Reference

Device Time Service.

5.8.1 Detailed Description

Device Time Service.

5.9 src/ble/services/wens/wens.h File Reference

WENS library.

```
#include "stdint.h"
```

Classes

- struct features_t
- struct wen_features_t
- struct wen_status_t
- struct ens_settings_t
- struct ens_log_t
- struct ens_identifier_t

Functions

- int wens_features_indicate (wen_features_t data)

  Function for indicating the WEN Feature characteristic.
- int wens_get_ens_settings (ens_settings_t ∗settings)

  Function for retreiving the ENS Settings.
- int wens_ens_log_notify (ens_log_t record)

  Function for notifying ENS records on the ENS Log characteristic.
- int wens_ens_identifier_indicate (ens_identifier_t identifier)

  Function for indicating ENS identifier characteristic.
- int wens_ens_settings_indicate (ens_settings_t settings)

  Function for indicating ENS settings characteristic.
- int wens_racp_indicate (uint8_t data)

  Function for indicating WEN features characteristic.
- int wens_status_indicate (wen_status_t status)

  Function for indicating WEN status characteristic.

5.9.1 Detailed Description

WENS library.

This is a library for the Wearable Exposure Notification Service (WENS).

5.9.2 Function Documentation

5.9.2.1 wens_ens_identifier_indicate()

```
int wens_ens_identifier_indicate (
            ens_identifier_t identifier )
```

Function for indicating ENS identifier characteristic.

Parameters

identifier	ENS identifier.

Returns

int 0 in case of success or negative value in case of error.

5.9.2.2 wens_ens_log_notify()

```
int wens_ens_log_notify (
            ens_log_t record )
```

Function for notifying ENS records on the ENS Log characteristic.

Parameters

record	ENS record.

Returns

int 0 in case of success or negative value in case of error.

5.9.2.3 wens_ens_settings_indicate()

```
int wens_ens_settings_indicate (
            ens_settings_t settings )
```

Function for indicating ENS settings characteristic.

Parameters

settings	ENS settings.

Returns

int 0 in case of success or negative value in case of error.

5.9.2.4 wens_features_indicate()

```
int wens_features_indicate (
            wen_features_t data )
```

Function for indicating the WEN Feature characteristic.

Parameters

data	The data to indicate.

Returns

int Returns 0 on success, negative otherwise.

5.9.2.5 wens_get_ens_settings()

```
int wens_get_ens_settings (
            ens_settings_t * settings )
```

Function for retreiving the ENS Settings.

Parameters

settings	The ENS Settings.

Returns

int Returns 0 on success, negative otherwise.

5.9.2.6 wens_racp_indicate()

```
int wens_racp_indicate (
            uint8_t data )
```

Function for indicating WEN features characteristic.

Note

The data parameter is not supposed to be an uint8_t, but it is set to that temporarily until more of the RACP is implemented.

Parameters

data	RACP data.

Returns

int 0 in case of success or negative value in case of error.

5.9.2.7 wens_status_indicate()

```
int wens_status_indicate (
            wen_status_t status )
```

Function for indicating WEN status characteristic.

Parameters

status	WEN status.

Returns

int 0 in case of success or negative value in case of error.

5.10 src/ble/uuid.h File Reference

UUID library.

```
#include <bluetooth/uuid.h>
```

Macros

- #define **BT_UUID_GAENS_VAL** 0xFD6F
- #define **BT_UUID_GAENS** BT_UUID_DECLARE_16(BT_UUID_GAENS_VAL)
- #define **BT_UUID_WENS_VAL** 0xFF00
- #define **BT_UUID_WENS** BT_UUID_DECLARE_16(BT_UUID_WENS_VAL)
- #define **ENS_LOG_UUID** 0xFF01
- #define **BT_UUID_ENS_LOG** BT_UUID_DECLARE_16(ENS_LOG_UUID)
- #define **WEN_FEATURES_UUID** 0xFF02
- #define **BT_UUID_WEN_FEATURES** BT_UUID_DECLARE_16(WEN_FEATURES_UUID)
- #define **ENS_IDENTIFIER_UUID** 0xFF03
- #define **BT_UUID_ENS_IDENTIFIER** BT_UUID_DECLARE_16(ENS_IDENTIFIER_UUID)
- #define **ENS_SETTINGS_UUID** 0xFF04
- #define **BT_UUID_ENS_SETTINGS** BT_UUID_DECLARE_16(ENS_SETTINGS_UUID)
- #define **TEMPORARY_KEY_LIST_UUID** 0xFF05
- #define **BT_UUID_TEMPORARY_KEY_LIST** BT_UUID_DECLARE_16(TEMPORARY_KEY_LIST_UUID)
- #define **RACP_UUID** 0x2A52
- #define **BT_UUID_RACP** BT_UUID_DECLARE_16(RACP_UUID)
- #define **WEN_STATUS_UUID** 0xFF06
- #define **BT_UUID_WEN_STATUS** BT_UUID_DECLARE_16(WEN_STATUS_UUID)
- #define **BT_UUID_DTS_VALUE** 0x1847
- #define **BT_UUID_DTS** BT_UUID_DECLARE_16(BT_UUID_DTS_VALUE)
- #define **BT_UUID_DTS_FEATURE_VALUE** 0x2B8E
- #define **BT_UUID_DTS_FEATURE** BT_UUID_DECLARE_16(BT_UUID_DTS_FEATURE_VALUE)
- #define **BT_UUID_DTS_PARAMETERS_VALUE** 0x2B8F
- #define **BT_UUID_DTS_PARAMETERS** BT_UUID_DECLARE_16(BT_UUID_DTS_PARAMETERS_VAL↩UE)
- #define **BT_UUID_DTS_DEVICE_TIME_VALUE** 0x2B90
- #define **BT_UUID_DTS_DEVICE_TIME** BT_UUID_DECLARE_16(BT_UUID_DTS_DEVICE_TIME_VAL↩UE)
- #define **BT_UUID_DTS_CONTROL_POINT_VALUE** 0x2B91
- #define **BT_UUID_DTS_CONTROL_POINT** BT_UUID_DECLARE_16(BT_UUID_DTS_CONTROL_POIN↩T_VALUE)
- #define **BT_UUID_DTS_CHANGE_LOG_DATA_VALUE** 0x2B92
- #define **BT_UUID_DTS_CHANGE_LOG_DATA** BT_UUID_DECLARE_16(BT_UUID_DTS_CHANGE_LO↩G_DATA_VALUE)
- #define **BT_UUID_DTS_RACP_VALUE** 0x2A52
- #define **BT_UUID_DTS_RACP** BT_UUID_DECLARE_16(BT_UUID_DTS_RACP_VALUE)

5.10.1 Detailed Description

UUID library.

This is a library for UUIDs that is not defined in Zephyr.

5.11 src/gaens/crypto.h File Reference

Cryptography module.

`#include <stdint.h>`

Macros

- #define TEK_ROLLING_PERIOD 144

  The rolling period for changing Temporary Exposure Keys in units of 10 minutes. 144 therefore represents represents 24 hours.
- #define TEK_LENGTH 16

  Length of a Temporary Exposure Key as specified in GAENS cryptography specifications (in bytes).
- #define RPIK_LENGTH 16

  Length of a Rolling Proximity Identifier Key as specified in GAENS cryptograhy specifications (in bytes).
- #define RPI_LENGTH 16

  Length of a Rolling Proximity Identifier as specified in GAENS cryptography specifications (in bytes).
- #define AEMK_LENGTH 16

  Length of an Associated Encrypted Metadata Key as specified in GAENS cryptography specifications (in bytes).

Functions

- int crypto_init (void)

  Initialize the crypto library. This initializes the keys used in the two AES encryptions for the RPI and AEM.
- int crypto_en_interval_number (uint32_t ∗output)

  Generate an Exposure Notification Interval Number. This number specifies a 10 minute window, meaning each time this number is incremented by 1, 10 minutes have passed.
- int crypto_tek (uint8_t ∗tek, uint8_t tek_len, uint32_t ∗tek_timestamp)

  Derive a Temporary Exposure Key and timestamp from which the key is valid. The key will be valid from the value specified in `tek_timestamp` and until a full `TEK_ROLLING_PERIOD` has occured.
- int crypto_rpik (const uint8_t ∗tek, const uint8_t tek_len, uint8_t ∗rpik, const uint8_t rpik_len)

  Derive a Rolling Proximity Identifier Key from a Temporary Exposure Key.
- int crypto_rpi (const uint8_t ∗rpik, uint8_t ∗rpi)

  Derive a Rolling Proximity Identifier from a Rolling Proximity Identifier Key.
- int crypto_rpi_decrypt (const uint8_t ∗rpik, const uint8_t ∗rpi, uint8_t ∗dec_rpi)

  Decrypt a Rolling Proximity Identifier.
- int crypto_aemk (const uint8_t ∗tek, const uint8_t tek_len, uint8_t ∗aemk, const uint8_t aemk_len)

  Derive Associated Encrypted Metadata Key from a Temporary Exposure Key.
- int crypto_aem (const uint8_t ∗aemk, uint8_t ∗rpi, const uint8_t ∗bt_metadata, const uint8_t bt_metadata_len, uint8_t ∗aem)

  Encrypt bluetooth metadata using an Associated Encrypted Metadata Key. Here the current Rolling Proximity Identifier is used as part of the encryption.
- int crypto_aem_decrypt (const uint8_t ∗aem, const uint8_t aem_len, const uint8_t ∗aemk, uint8_t ∗rpi, uint8_t ∗aem_dec)

  Decrypt associated encrypted metadata based on a given RPI and AEMK used for encryption.

5.11.1 Detailed Description

Cryptography module.

This is a module for handling GAENS cryptography.

5.11.2 Function Documentation

5.11.2.1 crypto_aem()

```
int crypto_aem (
            const uint8_t * aemk,
            uint8_t * rpi,
            const uint8_t * bt_metadata,
            const uint8_t bt_metadata_len,
            uint8_t * aem )
```

Encrypt bluetooth metadata using an Associated Encrypted Metadata Key. Here the current Rolling Proximity Identifier is used as part of the encryption.

Parameters

aemk	Pointer to Associated encrypted metadata key
rpi	Pointer to a copy of current Rolling proximity identifier. It is important that this is a copy of of current RPI as the encryption algorithm may change it. (Should be of length `RPI_LENGTH`)
bt_metadata	Pointer to bluetooth metadata to encrypt
bt_metadata_len	Length of Bluetooth metadata to encrypt
aem	Pointer to store the associated encrypted metadata in (should be of size `bt_metadata_len`)

Returns

int 0 on success, negative otherwise

5.11.2.2 crypto_aem_decrypt()

```
int crypto_aem_decrypt (
            const uint8_t * aem,
            const uint8_t aem_len,
            const uint8_t * aemk,
            uint8_t * rpi,
            uint8_t * aem_dec )
```

Decrypt associated encrypted metadata based on a given RPI and AEMK used for encryption.

Parameters

aem	Pointer to associated encrypted metadata to encrypt
aem_len	Length of `aem` (should be AEM_LENGTH)
aemk	Pointer to associated encrypted metadata key which was used when encrypting
rpi	Pointer to rolling proximity identifier used when encrypting
aem_dec	Pointer to store decrypted metadata in

Returns

int 0 on success, negative otherwise

5.11.2.3 crypto_aemk()

```
int crypto_aemk (
            const uint8_t * tek,
            const uint8_t tek_len,
            uint8_t * aemk,
            const uint8_t aemk_len )
```

Derive Associated Encrypted Metadata Key from a Temporary Exposure Key.

Parameters

tek	Pointer to temporary exposure key
tek_len	Length of `tek` (should be `TEK_LENGTH`)
aemk	Pointer to store associated encrypted metadata in
aemk_len	Length of output (should be `AEMK_LENGTH`)

Returns

int 0 on success, negative otherwise

5.11.2.4 crypto_en_interval_number()

```
int crypto_en_interval_number (
            uint32_t * output )
```

Generate an Exposure Notification Interval Number. This number specifies a 10 minute window, meaning each time this number is incremented by 1, 10 minutes have passed.

Parameters

output	Pointer to where to store the exposure notification interval number

Returns

int 0 on success

5.11.2.5 crypto_init()

```
int crypto_init (
            void  )
```

Initialize the crypto library. This initializes the keys used in the two AES encryptions for the RPI and AEM.

Returns

int 0 on success, negative otherwise

5.11.2.6 crypto_rpi()

```
int crypto_rpi (
            const uint8_t * rpik,
            uint8_t * rpi )
```

Derive a Rolling Proximity Identifier from a Rolling Proximity Identifier Key.

Parameters

rpik	Pointer to rolling proximity identifier key
rpi	Pointer to store rolling proximity identifier key in (must be of the same length as `rpik`)

Returns

int 0 on success, negative otherwise

5.11.2.7 crypto_rpi_decrypt()

```
int crypto_rpi_decrypt (
            const uint8_t * rpik,
            const uint8_t * rpi,
            uint8_t * dec_rpi )
```

Decrypt a Rolling Proximity Identifier.

Parameters

rpik	Pointer to rolling proximity identifier key used to encrypt the RPI
rpi	Pointer to rolling proximity identifier to decrypt
dec_rpi	Pointer to store decrypted RPI in (should be `RPI_LENGTH`)

Returns

int 0 on success, negative otherwise

5.11.2.8 crypto_rpik()

```
int crypto_rpik (
            const uint8_t * tek,
            const uint8_t tek_len,
            uint8_t * rpik,
            const uint8_t rpik_len )
```

Derive a Rolling Proximity Identifier Key from a Temporary Exposure Key.

Parameters

tek	Pointer to temporary exposure key
tek_len	Length of `tek` (should be `TEK_LENGTH`)
rpik	Pointer to store rolling proximity identifier key in
rpik_len	Length of output (should be `RPIK_LENGTH`)

Returns

int 0 on success, negative otherwise

5.11.2.9 crypto_tek()

```
int crypto_tek (
            uint8_t * tek,
            uint8_t tek_len,
            uint32_t * tek_timestamp )
```

Derive a Temporary Exposure Key and timestamp from which the key is valid. The key will be valid from the value specified in `tek_timestamp` and until a full `TEK_ROLLING_PERIOD` has occured.

Parameters

tek	Output to store the temporary exposure key
tek_len	Length of the output (should be `TEK_LENGTH`)
tek_timestamp	Timestamp from which `tek` is valid (in units of 10 minutes)

Returns

int 0 on success, negative otherwise

5.12 src/gaens/gaens.h File Reference

Google/Apple Exposure Notification System cryptography module.

```
#include "crypto.h"
#include <stdint.h>
```

Macros

- #define AEM_LENGTH 4

  Length (in bytes) of the associated encrypted metadata as specified in the GAENS Bluetooth specification.
- #define GAENS_SERVICE_DATA_LENGTH RPI_LENGTH + AEM_LENGTH

  Length (in bytes) of the service data to be sent in contact tracing advertising packets as specified in the GAENS Bluetooth specification.

Functions

- int gaens_init (void)

  Initialize GAENS module. Must be run before attempting to use any of the other functions in this module.
- int gaens_get_rpi (uint8_t ∗rpi)

  Get the current rolling proximity identifier (RPI).
- int gaens_get_rpi_decrypted (uint8_t ∗dec_rpi)

  Get the current rolling proximity identifier decrypted.
- int gaens_get_tek (uint8_t ∗tek, uint32_t ∗tek_timestamp)

  Get the current temporary exposure key (TEK) and the timestamp from which it is valid.
- int gaens_update_rpi (void)

  Derive a new rolling proximity identifier (RPI) and store this for future use. The new RPI can be obtained by calling the function `gaens_get_rpi`.
- int gaens_update_keys (void)

  Updates the temporary exposure key (TEK), rolling proximity identifier key (RPIK), and associated encrypted metadata key (AEMK). The current TEK and timestamp can be obtained by calling the function `gaens_get_tek`, while the current RPIK and AEMK are internal to this module and cannot be extracted.
- int gaens_encrypt_metadata (const uint8_t ∗metadata, const uint8_t metadata_len, uint8_t ∗aem)

  Encrypt metadata to be stored in exposure notification advertisement packets.
- int gaens_decrypt_metadata (const uint8_t ∗aem, const uint8_t aem_len, uint8_t ∗decrypted_aem)

  Decrypt metadata encrypted with the current RPI and AEMK.
- int gaens_ble_addr_expired (void)

  Check if 10 minutes have passed since the last time the Bluetooth address was changed, meaning the address must be changed.
- int gaens_tek_expired (void)

  Check if temporary exposure key (TEK) has expired, i.e. 24 hours have passed since the last TEK key was generated.

5.12.1 Detailed Description

Google/Apple Exposure Notification System cryptography module.

This is a module for handling GAENS cryptography.

5.12.2 Function Documentation

5.12.2.1 gaens_ble_addr_expired()

```
int gaens_ble_addr_expired (
            void )
```

Check if 10 minutes have passed since the last time the Bluetooth address was changed, meaning the address must be changed.

Note

When this function returns 1, the current RPI will not be valid and thus the function `gaens_update_rpi` must be called in order to create a new valid RPI.

Returns

int 0 if less than 10 minutes have passed, 1 if 10 minutes or more have passed, negative on error

5.12.2.2 gaens_decrypt_metadata()

```
int gaens_decrypt_metadata (
            const uint8_t * aem,
            const uint8_t aem_len,
            uint8_t * decrypted_aem )
```

Decrypt metadata encrypted with the current RPI and AEMK.

Parameters

aem	Pointer to associated encrypted metadata to decrypt
aem_len	Length of `aem` (should be AEM_LENGTH)
decrypted_aem	Pointer to store decrypted output in

Returns

int 0 on success, negative otherwise

5.12.2.3 gaens_encrypt_metadata()

```
int gaens_encrypt_metadata (
            const uint8_t * metadata,
            const uint8_t metadata_len,
            uint8_t * aem )
```

Encrypt metadata to be stored in exposure notification advertisement packets.

Parameters

metadata	Pointer to metadata to encrypt
metadata_len	Length of metadata to encrypt (should be `AEM_LENGTH`)
aem	Pointer to store the associated encrypted metadata in (should be of length `AEM_LENGTH`)

Returns

int 0 on success, negative otherwise

5.12.2.4 gaens_get_rpi()

```
int gaens_get_rpi (
            uint8_t * rpi )
```

Get the current rolling proximity identifier (RPI).

Parameters

rpi	Pointer to store the current RPI in. Must be of length RPI_LENGTH

Returns

int 0 on success, negative otherwise

5.12.2.5 gaens_get_rpi_decrypted()

```
int gaens_get_rpi_decrypted (
            uint8_t * dec_rpi )
```

Get the current rolling proximity identifier decrypted.

Parameters

dec_rpi	Pointer to store the current decrypted RPI in. Must be of length RPI_LENGTH

Returns

int 0 on success, negative otherwise

5.12.2.6 gaens_get_tek()

```
int gaens_get_tek (
            uint8_t * tek,
            uint32_t * tek_timestamp )
```

Get the current temporary exposure key (TEK) and the timestamp from which it is valid.

Parameters

tek	Pointer to store the current TEK in. Must be of length RPI_LENGTH
tek_timestamp	Timestamp from which current TEK is valid (in units of 10 minutes)

Returns

int 0 on success, negative otherwise

5.12.2.7 gaens_init()

```
int gaens_init (
            void )
```

Initialize GAENS module. Must be run before attempting to use any of the other functions in this module.

Note

This function should be called before starting advertising, in order to initialize the GAENS service data to advertise.

Returns

int 0 on success, negative otherwise

5.12.2.8 gaens_tek_expired()

```
int gaens_tek_expired (
            void )
```

Check if temporary exposure key (TEK) has expired, i.e. 24 hours have passed since the last TEK key was generated.

Note

When this function returns 1, the current TEK, RPIK and AEMK are no longer valid and thus the function `gaens_update_keys` must be called in order to update these keys.

Returns

int 0 if TEK has not expired, 1 if TEK has expired, negative on error

5.12.2.9 gaens_update_keys()

```
int gaens_update_keys (
            void )
```

Updates the temporary exposure key (TEK), rolling proximity identifier key (RPIK), and associated encrypted metadata key (AEMK). The current TEK and timestamp can be obtained by calling the function `gaens_get_tek`, while the current RPIK and AEMK are internal to this module and cannot be extracted.

Note

This function should be called every time the function `gaens_tek_expired` returns 1, which happens once every 24 hours.

Returns

int 0 on success, negative otherwise

5.12.2.10 gaens_update_rpi()

```
int gaens_update_rpi (
            void )
```

Derive a new rolling proximity identifier (RPI) and store this for future use. The new RPI can be obtained by calling the function `gaens_get_rpi`.

Note

This function should be called every time the function `gaens_ble_addr_expired` returns 1, which happens once every 10 minutes.

Returns

int 0 on success, negative otherwise

5.13 src/main.c File Reference

The main module.

```
#include "ble/ble.h"
#include "records/extmem.h"
#include <logging/log.h>
#include <zephyr.h>
```

Macros

- #define **LOG_MODULE_NAME** main

Functions

- **LOG_MODULE_REGISTER** (main)
- void **main** (void)

5.13.1 Detailed Description

The main module.

5.14 src/records/extmem.h File Reference

External Memory module.

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>
```

Macros

- #define **EXTMEM_SUBSECTOR_SIZE** 4096
- #define **EXTMEM_SECTOR_SIZE** 65536
- #define **EXTMEM_CHIP_SIZE** 4194304

Functions

- int extmem_init (void)

  Function for initializing the external memory.
- int extmem_read (uint32_t offset, uint8_t buf[ ], size_t len)

  Function for reading from the external memory.
- int extmem_write (uint32_t offset, const void ∗data, size_t len)

  Function for writing from the external memory.
- int extmem_erase (uint32_t offset, size_t size)

  Function for erasing data on the external memory chip.

5.14.1 Detailed Description

External Memory module.

This is a module for communicating with the external NOR flash memory.

5.14.2 Function Documentation

5.14.2.1 extmem_erase()

```
int extmem_erase (
            uint32_t offset,
            size_t size )
```

Function for erasing data on the external memory chip.

The size of the area to e erased has to be a multiple of the EXTMEM_SUBSECTOR_SIZE or EXTMEM_SECTOR_SIZE. To erase the whole memory use EXTMEM_CHIP_SIZE.

Parameters

offset	Erase area starting offset.
size	Size of area to be erased.

Returns

int Returns 0 on success, negative otherwise.

5.14.2.2 extmem_init()

```
int extmem_init (
             void  )
```

Function for initializing the external memory.

Returns

int Returns 0 on success, negative otherwise.

5.14.2.3 extmem_read()

```
int extmem_read (
             uint32_t offset,
             uint8_t buf[],
             size_t len )
```

Function for reading from the external memory.

Parameters

offset	Offset (byte aligned) to read.
buf	Offset (byte aligned) to read.
len	Number of bytes to read.

Returns

int Returns 0 on success, negative otherwise.

5.14.2.4 extmem_write()

```
int extmem_write (
             uint32_t offset,
             const void * data,
             size_t len )
```

Function for writing from the external memory.

Parameters

offset	Starting offset for the write.
data	Data to write.
len	Number of bytes to write.

Returns

int Returns 0 on success, negative otherwise.

5.15 src/records/storage.h File Reference

Storage module.

```
#include <stddef.h>
#include <stdint.h>
```

Macros

- #define **SIZE_OF_ONE_ENTRY** 34

Functions

- int storage_write_entry (int timestamp, const uint8_t gaens_service_data[ ], uint8_t rssi)

  Function for writing an ENS log entry to the external memory.
- int storage_read (uint32_t offset, uint8_t buf[ ], size_t len)

  Function for reading from the external memory.
- int storage_delete_all (void)

  Function for erasing the whole external memory.

5.15.1 Detailed Description

Storage module.

This is a module for adding, reading and manipulating Exposure Notification records.

5.15.2 Function Documentation

5.15.2.1 storage_delete_all()

```
int storage_delete_all (
             void  )
```

Function for erasing the whole external memory.

Returns

int Returns 0 on success, negative otherwise.

5.15.2.2 storage_read()

```
int storage_read (
             uint32_t offset,
             uint8_t buf[],
             size_t len )
```

Function for reading from the external memory.

Parameters

offset	Offset (byte aligned) to read.
buf	Buffer that will be filled with the data that is read.
len	Number of bytes to read.

Returns

int Returns 0 on success, negative otherwise.

5.15.2.3 storage_write_entry()

```
int storage_write_entry (
             int timestamp,
             const uint8_t gaens_service_data[],
             uint8_t rssi )
```

Function for writing an ENS log entry to the external memory.

Parameters

timestamp	The timestamp of the received advertisement packet.
gaens_service_data	The rolling proximity identifier and associated encrypted metadata from the received advertisement packet.
rssi	The RSSI from the received advertisement packet.

Returns

int Returns 0 on success, negative otherwise.

5.16 src/time/time.h File Reference

Time module.

```
#include <stdint.h>
```

Functions

- int set_current_time (uint32_t current_time)

  Set the current time.
- int get_current_time (uint32_t *time)

  Get the current time.

5.16.1 Detailed Description

Time module.

This is a module for setting and getting the current time.

5.16.2 Function Documentation

5.16.2.1 get_current_time()

```
int get_current_time (
            uint32_t * time )
```

Get the current time.

Parameters

time	Pointer to store the time in.

Returns

int 0 on success, negative otherwise

5.16.2.2 set_current_time()

```
int set_current_time (
            uint32_t current_time )
```

Set the current time.

Parameters

current_time	Time you want to set.

Returns

int 0 on success, negative otherwise

Index

Martin Gulbrandsen Aalien

Development and Optimization of a Contact Tracing Wearable

NTNU
Norwegian University of
Science and Technology