Software platform for health monitoring embedded system with Bluetooth LE capabilities

AUTHOR JIM THEORELL
SUPERVISOR FREDRIK JONSSON
EXAMINER INGO SANDER
Abstract

With starting point in a project called the HealthyWay project, this report explores how one can create bluetooth low energy applications on a nRF51822 MCU. The nRF51822 is the heart of an embedded system designed to measure ECG signals, breathing and coughing in order to send this information through bluetooth to a smartphone. How to use bluetooth low energy is subject to research and the results are a main application as well as a collection of so services used for communication between bluetooth units. Additionally a library for communicating through SPI with a ADS1292R ECG chip is produced. Finally some ideas for future applications of the results are presented.
Referat

Contents

List of Figures

List of Tables

I Introduction 1

1 Background 2
  1.1 General ................................................. 2
  1.2 Healthy Way ............................................ 2
    1.2.1 Bluetooth Protocol .................................. 3
    1.2.2 Software architecture specification ..................... 3
    1.2.3 Smartphone Software implementation ...................... 4
    1.2.4 Wireless Device software ............................... 4

2 Defining the project 5
  2.1 Purpose ................................................. 5
  2.2 Problem ................................................. 5

II Hardware platform 7

3 Healthy Way project 9
  3.1 Patch .................................................. 9
  3.2 nRF51822 chip ............................................ 10
  3.3 ADS1292R ............................................... 11

4 Development platforms 12
  4.1 nRF51822 Development Kit ............................... 12
  4.2 ADS1292RECG-FE Analog Evaluation Module ................. 13
  4.3 Healthy Way prototype ................................... 14
  4.4 Software ............................................... 14
Abbreviations

HWP .......... Hardware platform
ADC .......... Analogue Digital Converter
MCU .......... Microcontroller unit
PCB .......... Printed Circuit Board
CPU .......... Central Processing Unit
SPS .......... Samples Per Second
SPI .......... Serial Peripheral Interface
BLE .......... Bluetooth Low Energy
I/O .......... Input/Output
CCCD .......... Client Characteristic Configuration Descriptor
UUID .......... Universally Unique IDentifier
API .......... Application Programming Interface
List of Figures

3.1 The Healthy Way project idea ......................................................... 9
3.2 The design of the Healthy Way patch ................................................. 10

4.1 nRF51822 Development Kit ......................................................... 12
4.2 Example for powering the Development Kit ........................................ 13
4.3 ADS1292R Evaluation Kit ............................................................. 13
4.4 Example for powering the Development Kit ........................................ 14

5.1 Simplified depiction of the BLE protocol ........................................... 17
5.2 An example how a characteristic may work ......................................... 19

6.1 Relation between Softdevice and Application ..................................... 20
6.2 Transfer with either Read or Notify ................................................... 24

8.1 Overview of the project ..................................................................... 29
8.2 Code flow ....................................................................................... 31
List of Tables

7.1 Most important SPI commands. ........................................ 26
7.2 Brief overview of the API functions ................................ 28

8.1 TimeStamp Service characteristic overview ...................... 30
8.2 ECG Service characteristic overview .............................. 30
8.3 Respiration Service characteristic overview ..................... 30
Part I

Introduction
Chapter 1

Background

This chapter introduces the reader to the background of the project and other closely related projects. These projects either serve as a basis for certain aspects of the thesis or will be used in conjunction with it.

1.1 General

With the introduction of the smartphone many things changed. Suddenly every other person now has a programmable device in their pocket. This enables developers to create a great variety of applications and accessories to be used in many different areas, such as entertainment, communication, social media, logistics, healthcare etc.

Furthermore, the market seems to have stabilized on a select few platforms such as Android and iOS [2]. This has reduced the costs of development through limiting the need for customised code over different devices.

Along with the standardization of software platforms, the need for standardized hardware modules emerged. Especially when it comes to different communication standards. One of them is Bluetooth. A short range radio communication protocol that is projected to be included in 96% of all smartphones by 2018 [3].

1.2 Healthy Way

"In the HealthyWay project we investigate how a smartphone can be used to collect information from printed wireless Electro Cardiograph (ECG) sensors and combine this with the accelerometer and position readings of the phone. The results can be displayed in real time, or saved for later analysis by medical personnel." [1, page 1]

The specifics about the HealthyWay hardware platform will be given during part Example for powering the Development Kit.
The HealthyWay project consists of the four parts explained in the following sections. However the purpose of this thesis is only to address the last, Wireless Device software.

1.2.1 Bluetooth Protocol
The bluetooth version used in the HealthyWay project is 4.0 (Known as 'SMART' or 'LE'). It is a versatile protocol with different ways to achieve the desired solution. Therefore a detailed specification is required in order for both smartphone application and wireless device software implementation to be developed asynchronous. Furthermore this ensures that the full potential of bluetooth is being utilized as well as following industry standards.

This task has been done prior to this thesis and will be further discussed during chapter Bluetooth.

1.2.2 Software architecture specification
This task is for designing the user interface of the smartphone application as well as related file structures. The application should have the following functionalities [1, page 4-5]:

Welcome
The welcome screen shows an image, as well as the software revision number.

Scan and config
The Scan and Config screen shows available Bluetooth devices within range. When selecting one device, it displays a list of available services and allows the user to select which services to enable.

Measurement waveforms
The measurement screen display real-time waveforms for the data measured by the wireless node. Data to be displayed depends on the settings in the configuration screen. The measurement screen is only available if the user is connected to a device capable of generating waveform data. If the user is connected to several devices, more than one waveform screen may be available. A button on the measurement screen allows the user to save waveforms to Dropbox.

Adherence data
If the user is connected to a device logging user adherence, the adherence data screen will be displayed. The adherence screen displays type of medicine, how many pills have been taken and when was the last time a pill was taken.

Additional analysis screens
The software should be prepared for easy extension of additional analysis. Each additional screen should have access to measurements, and display post processed data.
This task will be produced by another group simultaneously with this thesis.

1.2.3 Smartphone Software implementation

Developing the smartphone application according to the "Software architecture specification" and the Bluetooth Protocol specification.

This part will be produced simultaneously with this thesis, by the same group that made the software architecture specification.

1.2.4 Wireless Device software

This part is the subject of this thesis and will be defined during the following chapter, Defining the project.
Chapter 2

Defining the project

2.1 Purpose

In this thesis the application of smartphones in the medical field is being researched. In particular how the use of a wearable device in conjunction with a smartphone application may be used to monitor a patient’s health. The purpose of this thesis is to produce the source code, including documentation, for the wearable sensor in the Healthy Way project.

2.2 Problem

The predefined hardware platform generates several tasks that the embedded software must tackle:

Sensors

The HWP contain two different sensors, the first for collecting Heart rate and respiration data and the second is an accelerometer for measuring movement. The first task of the software is to collect data from the peripherals on the HWP. The data type for the first sensor is ADC values in volts. Voltage is being used to measure both heart rate and respiration. The second sensor gives us accelerometer measurements for cough detection.

In order to collect data from these devices some form of library should be used in order to abstract the hardware calls of that peripheral device. Where these libraries do not exist, they have to be implemented.

Bluetooth

The MCU have a bluetooth radio transmitter embedded. The code to make this radio work will have to be created with help from the resources to do so provided with the MCU. It will probably be necessary to learn of lot about both the BLE protocol and the chip specific implementation of the BLE protocol.
Power

The HWP is powered by a battery. This will always mean that power usage should be limited. The different peripheral devices will need to be shut down and the MCU source code will need to be optimized for low power consumption.
Part II

Hardware platform
To be able to properly understand the things discussed in this part, it's preferable to have the source code close at hand. The source code may be acquired from the project github page: https://gits-14.sys.kth.se/fjon/HealthyWay or studied from the documentation: https://gits-14.sys.kth.se/pages/fjon/HealthyWay/index.html.
Chapter 3

Healthy Way project

Figure 3.1: The Healthy Way project idea.

Simply put, the Healthy Way platform is wearable patch attached to a person's chest. This patch will measure electrocardiographic and respiratory waveforms as well as use an accelerometer for detecting coughing. These measurements are then gathered by a MCU and sent by Bluetooth to a smartphone application. This is illustrated in fig 6.1

3.1 Patch

The patch is a long rounded shape attached with an adhesive from the bottom of the unit. This makes sure the three probes (blue in fig 3.2) on the bottom has proper contact with the skin. Inside the plastic casing there's two batteries and wires connecting battery, probes and circuitry. The casing containing the PCB is sturdy and is attached to the patch with magnets on the PCB. These also work as contact surface between patch and case.

The patch is designed as to be easy to produce. This is done by printing most of the unit. All but the capsule containing the circuitry and the circuit itself is printed
together. This includes bendable batteries, plastic casing and probes. One of the advantages of this is flexibility which makes the patch easier to attach to the body. Another is the cost. The idea is to have the cheap printed part expendable while the non-printed expensive part will be reused. Another argument for this is that the adhesive will wear out and contact with the body will generate filth.

The Healthy Way PCB have three major Integrated circuits: nRF51822, ads1292r and LSM330. The former two will be addressed in the following sections while the latter is not included in this thesis.

3.2 nRF51822 chip

The nRF51822 chip from Nordic Semiconductor is single chip solution for bluetooth communication. Its based on the ARM Cortex M0 CPU, with 256kB\textsuperscript{1} Flash, 16kB\textsuperscript{2} RAM and a 2.4GHz radio transceiver.

\textsuperscript{1}256kB for the nRF51822-AA version used in the Healthy Way project\textsuperscript{[4]}
\textsuperscript{2}16kB for the nRF51822-AA version used in the Healthy Way project\textsuperscript{[4]}

Figure 3.2: The design of the Healthy Way patch.
3.3 ADS1292R

The ADS1292R chip from Texas Instruments is a 2-channel 24-bit ADC for Biopotential Measurements. It has a built-in programmable gain amplifier, internal reference, and an onboard oscillator. This means that its can measure analogue voltage measured from the skin and convert it to digital values to be transmitted to another chip. The internal reference and onboard oscillator means that once configured it will work without assistance from the MCU. All we need to do is to set the appropriate registers in the ADS1292R and receive the digital voltage values.

It has 2-channels; One for measuring electrical activity from the heart, the other for measuring impedance through the lungs in order to determine whatever they are full or empty. The measurements may be done from 125 SPS to 8 kSPS and then amplified 1, 2, 3, 4, 6, 8, or 12 times.

Transfer to and form the device is done with SPI\textsuperscript{3}

\textsuperscript{3}SPI will be explained in section ADS1292R library
Chapter 4

Development platforms

Three different platforms have been used during software development:

4.1 nRF51822 Development Kit

A development kit from Nordic Semiconductor. It contains two different nRF51822 PCB:ς. One with integrated antenna and the other with external. For programming these, the included J-Link LITE CortexM from segger is being used. Lastly the kit contains a "nRF-dongle" that you connect to your PC in order to communicate with any BLE devices.

The development kit is supposed to be used with a "motherboard" but because this is extravagant in our case a 3.3V LDO (Low Drop Out) power regulator is used to power the chip, the logical circuit used is illustrated in fig 4.2. Because its convenient to use the +5 any PC supplies, the regulator is powered through a
severed USB cable. This regulator is then connected to the jumper K1 on the PCB. Please be advised the K1 jumper position: Between pin 2 and 3. The ground from the regulator is connected to any ground on the nRF51822 PCB, an example is illustrated in fig 4.2.

4.2 ADS1292RECG-FE Analog Evaluation Module

The ADS1292RECG-FE Analog Evaluation Module is mainly for experimenting with different settings for the probes. The program "ADS1x9x ECG-FE Evaluation Software"\(^1\) may be used to capture and visualize data. It also contains a graph-

\(^{1}\)May be acquired through the TI website: \text{http://www.ti.com/tool/ads1292ecg-fe}
ical interface for setting all the register. The accompanying user guide\(^2\) contains all necessary information to use the software. During testing of the Healthy Way smartphone application data captured from this software was used for testing purposes.

### 4.3 Healthy Way prototype

This PCB is very close to the actual Healthy Way wearable sensor prototype. The difference is the 6 pins on the side and 9 pins on the bottom as illustrated in fig 4.4. The 6 pins on the side is simply some of the vacant pins on the MCU for general debugging. The 9 pins on the bottom is the a standard interface for programming and debugging ARM processors with the SEGGER J-Link previously mentioned. Because of the 6 pin header there’s not enough space for the accelerometer which has been left out (Can be seen as the copper patch in the top left corner on the PCB).

### 4.4 Software

For developing nRF51822 application it’s recommended to use the KEIL MDK (Microcontroller Development Kit) from ARM. It is easy to integrate with the SDK that is needed for nRF51822 development. Additionally all examples provided by Nordic Semi contains KEIL Uvision (The IDE included in the MDK) project files which gives out-of-the-box functionality.

Accompanying nRF51822 DK is a variety of software applications. Two of them needs to be used inorder to set up a working development environment: Firstly "nRFgo Studio" is used to load the softdevice\(^3\) onto the MCU. The second is "Master Control Panel" which is use with the nRF-dongle to both send and receive BLE data.

---


\(^3\)This concept will be explained during the SoftDevice part.
This is very useful for debugging since you may see what the bluetooth packages contain.
Part III

Software implementation
Chapter 5

Bluetooth

To understand the software implementation it is necessary to have a basic understanding of the BLE protocol. The figure below illustrates a crude and simplified version of the protocol that should be sufficient to understand the Healthy Way source code.

![Figure 5.1: Simplified depiction of the BLE protocol](image)

5.1 GAP

Generic Access Profile is responsible for all things needed in a basic connection between BLE devices regardless of what different functionality the device may perform. This profile exist in all BLE applications. GAP is responsible for advertising the device. This meaning that it sends a data package on certain channels during a set interval for other devices to find. The advertising package contains things like Device name and ID:s for certain services that we want the possible smartphone to know we have.
BLE have different roles called Central and Peripheral. In our case the smartphone is the central device and the patch peripheral. Ontop of this there’s a server and client model that’s interchangeable. Meaning its not set in connection what device is which. E.G. a smartphone may request data from the peripheral device but the peripheral device may also request data from the smartphone.

GAP is also responsible for setting and managing the security levels in the connection.

When the connection have been set up by GAP we need a set of rules to define how the devices speak to each other. This is where GATT comes in.

5.2 GATT

A Generic Attribute Profile is either a predefined profile\(^1\) or a custom, by the developer. The peripheral itself doesn’t know about profiles, because a profile is just a collection of Services. What kind of services there is in the peripheral is being discovered during connection.

A Service is a collection of ways to send and receive data grouped together in a way that makes it understandable. E.G. If a peripheral device have a time service. There might be two values available: Hour and minute. These both belong to the time service. Each service have its own set of what’s called characteristics. Each service may have one or more characteristic. A service is identified by something called UUID (Universally Unique IDentifier) which is a 128-bit binary number for custom services (16-bit for predefined services).

The different ways to send data listed under the service is called Characteristics. A Characteristic is member of a service with its own UUID based on the UUID of the service. An example of this is the UUID:s for the ECG service that may be found in the appendix: UUID. Each characteristic contains at least two attributes: Declaration and Value.

A Declaration contains three things:

- Value Handle
- UUID
- Properties

Value handle is simply to identify the Value attribute so that the application can find it.

Properties is the method of which the application may send and receive data. The property of the characteristic can be one of the following:

- Read

\(^1\)Predefined profiles are a collection of profiles defined by the Bluetooth Special Interest Group. More can be read here: [https://developer.bluetooth.org/TechnologyOverview/Pages/Profiles.aspx](https://developer.bluetooth.org/TechnologyOverview/Pages/Profiles.aspx)
• Write
• Write without response
• Notify
• Indicate
• Broadcast

These properties define different ways to transfer data between central and peripheral. Not all of them will be used in the Healthy Way application and will therefore not be explained. Notify however will be used to send ECG signals. Notify is a way of sending data to the central without waiting to get a notification. Every time the application updates the value for the characteristic a notification will be sent to the central. The peripheral don’t know and don’t care whatever the last value actually made it to the central. Now in the case of Notifications there’s another attribute in the characteristic: CCCD (Client Characteristic Configuration Descriptor). This attribute is for controlling if the peripheral will send notifications.

All these relations is easier to understand in an example. In fig 5.2b there’s an example with a Time characteristic: Suppose we have an embedded system with a clock. Once a second the application gets a new value from the clock. Every time it does it writes the value attribute with the new value 'TimeData'. The application now checks whatever the CCCD have been enabled by the central. If enabled the application send a package every second until the connection is broken or the central disables notifications through the CCCD.

![Diagram](Figure 5.2: An example how a characteristic may work.)
Chapter 6

Creating a BLE application

All the source code for this project is written in C. One thing that should be mentioned is the coding style used. Instead of the usual style of passing a lot of different variables to each function, something more like object oriented programming is used. For instance when initializing an arbitrary module, a data structure is set. Then when we call the function, we give a pointer to this structure as a parameter. This is the style used in implementation from Nordic Semi and have been applied to the rest of the project to.

6.1 SoftDevice

The Healthy Way application is using something called a SoftDevice. This is an abstraction layer provided by nRF51822 chip manufacturer to easier program the chip. The SoftDevice\(^1\) is loaded in to a protected section of the memory. From

\(^1\)In this project, Peripheral SoftDevice S110 is used.
the application point of view it works as the interface to control all bluetooth functionality. Apart from this, Nordic Semiconductor have a SDK\(^2\) to further abstract development.

How to load the SoftDevice onto the nRF51822 chip is best explained on page 35 of the Development Kit User Guide: [https://www.nordicsemi.com/eng/content/download/13239/213080/file/nRF51822_Developer_Kit_User_Guide%20v1.2.pdf](https://www.nordicsemi.com/eng/content/download/13239/213080/file/nRF51822_Developer_Kit_User_Guide%20v1.2.pdf)

6.2 Initialization process

There are several steps needed to create a BLE application with above mentioned resources. These steps will be explained below and then brought together in chapter Main application.

The, by Nordic Semiconductor, recommended workflow is to use an existing file as a template for your application. This is because a lot of code is necessary to have the basic functionality of a BLE application. For reference the template file used in the project, ble_app_template, has about 600 lines. This file is located in

6.2.1 Initialize the SoftDevice

Besides starting the device with a selected clocksource, the bluetooth stack needs to be activated. The bluetooth stack is the software implementation of the Bluetooth protocol stack. Lastly in order to handle any requests from the central device we need to register a function that should be called every time we get an event from the stack.

In the Healthy Way source code this is the function ble_stack_init.

6.2.2 Initialize the scheduler

The SoftDevice have a scheduler that receives all the BLE events. This scheduler can then be called to distribute the events.

In the Healthy Way source code this is the function scheduler_init.

6.2.3 Initialize GAP

All parameters related to connection basics needs to be set up. These include things as: Device name, Appearance\(^3\), security parameters and connection timings.

In the Healthy Way source code this is the functions gap_params_init, conn_params_init and sec_params_init.

\(^2\)In this project, version 6.1.0 of the SDK is used. [https://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v6.x.x/doc/6.1.0/index.html](https://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v6.x.x/doc/6.1.0/index.html)

\(^3\)This is a set of predefined usercases. In our case its set to BLE_APPEARANCE_GENERIC_HEART_RATE_SENSOR. This is so that the central will have an idea of what an unknown device does.
6.2.4 Initialize Advertising

When searching for a peripheral on for example you smartphone, the advertising data is what will show up. The advertising package have a limit of 31 bytes. So we need to choose what information we want to send. Transmission power level and a flags, telling central devices what kind of advertising we are doing, are always sent. We can choose to have the full device name (set earlier as GAP parameter) or just a fraction of it to save space. Lastly we pass on a array of short service UUIDS. In the our case we sends all short UUIDS for non mandatory services so that the central will know what this specific Healthy Way device provides functionality wise. Without this, the central would have to connect to the device in order to determine if it have for example the accelerometer installed.

In the Healthy Way source code this is the function advertising_init.

6.2.5 Initialize Services

Every service have there own initialization function we simply need to call them while passing along the required settings. When calling the functions a pointer to a special data structure, defined by the service, needs to be passed along. After initialization this structure will contain all the information we need to operate on the service.

In the Healthy Way source code this is the function services_init.

6.2.6 Start advertising

When everything is set-up properly, advertising must start. Without starting advertising no central device will be able to see or connect to the peripheral. This is done by setting a few parameters regarding how long and how often we should broadcast are package.

In the Healthy Way source code this is the function advertising_start.

6.3 Creating a service

In the same sprit as the main file, the recommendation is to start from a pre-existing service and modify it according to need. The same reason still stand: There’s a lot of boilerplate code needed to have a basic service. Nordic semi recommends starting with the ble_bas files, Battery Service.

Services are preferably implemented to be easily reused for other projects. This is how the predefined services are implemented and therefore the Healthy Way services follows this principle as well. This includes having the services in separate files and following a certain pattern for how the main application uses the service:

- Initializing

\[\text{See appendix UUID ECG Service UUID marked with red for example.}\]
• Handling its events
• Providing the I/O (In our case the communication with the ECG chip)

To make this as general as possible, the services provide a simple API (Application Programming Interface) with the functions required to manage it.

### 6.3.1 Data structures

In most cases, two different data structures are needed. The first one is a struct containing variables for initializing the service, given that the service needs to set initial values. The second is a struct that holds all status information for the service. This includes the handles\(^5\) used to address the service so that the application will be able to distinguish which service should be called when receiving data to the stack.

If the service will receive any data from the central device it will also need to define a Service event handler type. This is a function in the main app that will be passed to the service. This function will be called whenever the service receives data from the stack.

### 6.3.2 Initializing the service

As mentioned in section Initialize Services, every service have their own initialization function. In this function several things must be done:

- Set UUIDS
- Add service to stack
- Add all of the characteristics

If the service needs a service event handler, the function pointer for this needs to be copied to the service struct.

For an example of this, look at the ble_srv_ecg_init function in the ble_srv_ecg.c file.

### 6.3.3 Adding characteristics

How adding a characteristic is done varies somewhat depending on the what type it is. That’s why this is the first thing we need to decide. In section GATT different types of characteristics where listed. If we take for instance the notify characteristic, we need to set read/write permission on the Value field and the CCCD field. If we instead look at the write without response type, only read/write permission on the Value field needs to be set since there is no CCCD.

Besides these permissions, we need to set the UUID for the characteristic, decide what datasize the value may contain and register it to the SoftDevice stack.

\(^5\)A handle is a unique number used to identify the service.
6.3.4 Transfer data from peripheral to central

In most cases, services will have to send data to the central device. This is done by the main app sending a value to an API function. The API function passes this value to the SoftDevice stack. Then depending on what kind of characteristic it belongs to different things happen. If for instance the characteristic type is read, nothing will be sent. Instead the central will read the value when it wishes, disregarding any values it might have missed. If instead the characteristic would have been a notify, the peripheral sends a package every time the value is changed as long as notifications is enabled with the CCCD. This is illustrated in fig Overview of the project.

For an example of a read characteristic, look at the function ble_tss_seconds_update in the ble_srv_tss.c file. For an example of a notify characteristic, look at the function ble_ecg_measurement_send in the ble_srv_ecg.c file.

6.3.5 Receiving data from central

In services where we need to receive data from the central, another API function needs to be implemented. This function will be called from the main app along with corresponding function for every other service, whenever a BLE event is received. Every such function checks if the handle on that event matches its own service handle. If it does, it means that the data should be passed along to the function we got from the main app during initialization. This will be illustrated during section Code flow.
Chapter 7

ADS1292R library

SPI, Serial Peripheral Interface, is common interface for serial communication between integrated circuits. Functionality for sending and receiving are built in to the nRF51822 chip and accessible though the SDK. The problem is that the combination between this MCU, the nRF51822, and the ADS1292R chip seems to have never been done before. Atleast not in an open source project. This means that an interface between the main app and the ADS1292R is needed.

7.1 Portability

Since the world lacks a ADS1292R library for the nRF51822 chip and one needs to be implemented for this project, it seems appropriate to make it as portable as possible. So to achieve this, an API have been created. This API includes:

- Definitions for addresses and settings
- Data structure for initializing
- Macro with default values for above mentioned structure for easier initializing
- Functions for communication/controlling with the device

7.2 ADS1292R SPI interface

Before diving in to how the API is implemented, a basic understanding of how to control the device is needed. How to communicate with the device is defined in the ADS1292R datasheet [5]. The device has 11 different commands for use. Those important in this case are:

It is crucial to realize that the device have two modes:

- Read Data Continuous ENABLED mode
- Read Data Continuous DISABLED mode
Table 7.1: Most important SPI commands.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>DESCRIPTION</th>
<th>FIRST BYTE</th>
<th>SECOND BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Start or restart (synchronize) conversions</td>
<td>0000 1000 (08h)</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>Stop conversion</td>
<td>0000 1010 (0Ah)</td>
<td></td>
</tr>
<tr>
<td>RDATAC</td>
<td>Enable Read Data Continuous mode. This mode is the default mode at power-up</td>
<td>0001 0000 (10h)</td>
<td></td>
</tr>
<tr>
<td>SDATAC</td>
<td>Stop Read Data Continuously mode</td>
<td>0001 0001 (11h)</td>
<td></td>
</tr>
<tr>
<td>RREG</td>
<td>Read n nnnn registers starting at address r rrrr</td>
<td>001r rrrr (2xh)</td>
<td>000n nnnn</td>
</tr>
<tr>
<td>WREG</td>
<td>Write n nnnn registers starting at address r rrrr</td>
<td>010r rrrr (4xh)</td>
<td>000n nnnn</td>
</tr>
</tbody>
</table>

While the device is in ENABLED mode, the only thing that will not be ignored is the SDATAC command.

Besides these commands the ADS1292R uses other means to communicate, pins. Firstly a reset pin that may be trigger to reset the device, secondly a optional start pin to use in the same manner as the START command. Lastly a DataReady (DRDY) pin. When the device have been sent the START command, the internal ADC will convert heart signals to digital values with a set rate. Whenever a conversion have been done and are ready to be transmitted, the DRDY pin will go low (0 Volts).

### 7.3 ADS1292R configuration

The ADS1292R is configured by writing its registers. Writing registers is done with the WREG command but this will be handled by the API.

For instance to disable channel 1. You would have to write to the register CH1SET. A list of all the register is available on page 39 of the datasheet [5].

### 7.4 API

#### 7.4.1 Datastructures

The API have one struct called ads1292r_config_t. It contains variables for setting MCU pin numbers for corresponding ADS1292R pins. Optional values for power
on delays that might exist. Lastly it have a variable for all different registers on the ADS1292R. This struct should be initiated with the ADS1292R_INIT_DEFAULT macro. This sets all pins to NULL (To disable them) and all registers to whatever value they will have after the device have been reset (The device will assume default register values every reset, either by command or loss of power). After initiation with the macro. Any values may be changed to fit the current situation.

Another structure working with the exact same principle is the spi_master_config_t defined by the SDK’s SPI Master module. This structure contains every setting for starting one of the two SPI modules on the nRF51822. Even though this struct is not defined in this API, we still have a macro for initiation.

7.4.2 Initiation

The API function for initiation is ads1292r_init. It needs four parameters to work. Firstly both of the data structures mentioned above. Secondly it needs to know which of the two SPI modules. Lastly it needs a pointer to a function that should be called every time ADC data have been received.

With this provided the function does:

- Configures all pins
- Configures the SPI module
- Goes though a start up routine
- Sends SDATAC command
- Writes all registers that is not set to default values
- Configures interrupt on the DRDY pin

---

1 This routine is defined on page 62 of the datasheet [5].
### 7.4.3 API functions

<table>
<thead>
<tr>
<th>API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ads1292r_reset</td>
<td>Function for resetting the ads1292r</td>
</tr>
<tr>
<td>ads1292r_send_command</td>
<td>Function for sending commands to the ads1292r.</td>
</tr>
<tr>
<td>ads1292r_write_register</td>
<td>Function for writing one or more registers in the ads1292r.</td>
</tr>
<tr>
<td>ads1292r_read_register</td>
<td>Function for reading one or more registers in the ads1292r.</td>
</tr>
<tr>
<td>ads1292r_start_conversions</td>
<td>Function for starting ADC conversions.</td>
</tr>
<tr>
<td>ads1292r_stop_conversions</td>
<td>Function for stopping ADC conversions.</td>
</tr>
</tbody>
</table>

Table 7.2: Brief overview of the API functions. Details can be found in the source code documentation \(^2\).

### 7.4.4 Gathering data

As mentioned in Initiation section, the MCU pin connected to the DRDY pin on the ADS1292R have been set up generate interrupts. This means that every time the DRDY pin goes low, the MCU will know it and trigger a hardware interrupt. This interrupt calls an internal library function that:

- Clears the interrupt flags so that it might be triggered again
- Reads data from the device\(^3\)
- Sends the received value to the function passed to `ads1292r_init`

\(^2\)https://gits-14.sys.kth.se/pages/fjon/HealthyWay/group__ads1292r__spi__lib.html

\(^3\)6 bytes with structure: 24 status bits followed by a 24 bit ADC value (either ECG or respiration)
Chapter 8

Main application

Figure 8.1: Overview of the project

Now let's take a look at the different services provided by the project before exploring the code flow.

8.1 Services

8.1.1 TimeStamp Service

This service provides the central with a timestamp of how many seconds passed since the device started. More about why this service was implemented will be discussed during Future Work.

1https://gits-14.sys.kth.se/pages/fjon/HealthyWay/group__ble__srv__tss.html
Table 8.1: TimeStamp Service characteristic overview. Details can be found in the source code documentation\(^1\).

### 8.1.2 ECG Service

The ECG service enables the user to get a continuous stream of ECG measurements. It has a characteristic to enable conversions, another the send them and lastly one to give the central device information on which sampling rate and reference voltage is being used while capturing data. While sending data, the first 2 bytes is a numeric value used to ensure the bluetooth packages arrives and does so in the right order.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECG measurement</td>
<td>Notify</td>
</tr>
<tr>
<td>Sampling Rate &amp; Reference Voltage</td>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
<td>Write without Response</td>
</tr>
</tbody>
</table>

Table 8.2: ECG Service characteristic overview. Details can be found in the source code documentation\(^2\).

### 8.1.3 Respiration Service

The Respiration service is implemented in exactly the same way as the ECG service with one exception: It requests respiration instead of ECG data.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Respiration measurement</td>
<td>Notify</td>
</tr>
<tr>
<td>Sampling Rate &amp; Reference Voltage</td>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
<td>Write without Response</td>
</tr>
</tbody>
</table>

Table 8.3: Respiration Service characteristic overview. Details can be found in the source code documentation\(^3\).

### 8.2 Code flow

To illustrate the code flow, an example will be used. Lets follow what happens from device start until the smartphone receives ECG data:

\(^2\)https://gits-14.sys.kth.se/pages/fjon/HealthyWay/group__ble__srv__ecg.html  
\(^3\)https://gits-14.sys.kth.se/pages/fjon/HealthyWay/group__ble__srv__resp.html
1. Initializing

2. Advertise

3. Smartphone connects and discover services (Done on SoftDevice level).

4. User enables ECG measurements. The smartphone application enables notifications and sends data to the "Write" characteristic.

5. Write event is received by the SoftDevice.

6. An event dispatch function sends the event to all services. But only the ECG service will acknowledge it and send it to the ECG write handler passed to the service during initialization.

7. The write handler uses the ADS1292R library to send a START command to the ADS1292R chip.

8. From this point on, whenever a new ADC values is available, the DRDY pin will trigger the ADS1292R library to read data via SPI. When data is read, its passed to the function in the main app given to the library during initialization. This function will store the values in a buffer. And when the buffer is full update the call the service API function for updating the ECG measurement characteristic. Since this characteristic is of type notify, the SoftDevice will immediately send to the smartphone.

Figure 8.2: Code flow. Grey arrow indicate sequential parts.
Part IV

Conclusion
8.3 Goals

The purpose of this thesis was mainly to produce source code to enable use of the HealthyWay hardware platform. This have been achieved with the exception of a few things: The accelerometer, battery measurement, optimizing the code for low power consumption.

Along the course of the thesis, the focus was changed somewhat to emphasize creating general and portable solutions to enable modularity in the project. This gives two advantages: Its easier to understand the source code and simple to extend or subtract from the project.

This will hopefully lead to many adaptations of the platform in several different scenarios.

Another goal with the thesis was to provide documentation for the source code. This have been done completely. Every module created have been fully documented, variables and functions alike. This documentation can be found on the project github page: https://gits-14.sys.kth.se/pages/fjon/HealthyWay/index.html. Unfortunately this page is only accessible to members of the project. This is a subject for change.

8.4 Obstacles

During the work a few minor problems appeared with the PCB, namely bad connections. Related to this was getting the SPI communication to work a hassle. The reason for this can be attributed to the main obstacle during the thesis: Lack of documentation. The Nordic Semiconductor SDK which have to be used in all communication over BLE and SPI have only documentation generated from the source code. Which means that if the few words commented about a function is not enough to understand its use or context, you are on your own. The only place left is to turn to their Stackoverflow clone or study example code for answers.

8.5 Future Work

The obvious things first, the accelerometer needs to have its own service and library to work. The battery measurement needs implemented and power managements needs work.

However the interesting work ahead may not necessary be done on the embedded system: Data analysis. Only the imagination limits what can be done with this data.

One of the ideas that have been formed during the thesis came after a discussion with a medical student from KI (Karolinska Institutet):

If someone have an infection. One indication may be that their breathing exceeds 30 breaths per minute. Now the problem is that it’s hard to measure breathing because a person aware of their how they breath will subconsciously change its pace. If instead a person were to wear the HealthyWay patch. The patch could
wake with certain intervals and check the heart rate. If the heart rate is low and stable it indicates that the person is in a resting state. Then that is a perfect time to get an accurate measurement of the breathing frequency.

Another more relevant suggestion comes from employees at AstraZeneca: Sometimes it’s necessary to check the heart rate of laboratory animals with certain intervals. But hooking an animal up to a ECG machine is apparently not that convenient. If the animal instead were to wear a harness with the HealthyWay patch, this could bring comfort to the animals as well as saving time for the personnel.
# Appendix A

## UUID

<table>
<thead>
<tr>
<th>Type</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application base UUID:</td>
<td>4E1B****-81AA-11E2-9E96-0800200C9A66</td>
</tr>
<tr>
<td>ECG Service UUID:</td>
<td>4E1B94B0-81AA-11E2-9E96-0800200C9A66</td>
</tr>
<tr>
<td>ECG Measurement Characteristic UUID:</td>
<td>4E1B94B1-81AA-11E2-9E96-0800200C9A66</td>
</tr>
<tr>
<td>ECG Sampling Rate and Reference Voltage UUID:</td>
<td>4E1B94B2-81AA-11E2-9E96-0800200C9A66</td>
</tr>
<tr>
<td>ECG Write Characteristic UUID:</td>
<td>4E1B94B3-81AA-11E2-9E96-0800200C9A66</td>
</tr>
</tbody>
</table>
Bibliography


[2] International Data Corporation (IDC),

[3] IHS Technology, BLUETOOTH MARKET ANALYSIS,

[4] Nordic Semiconductor, nRF51 Series Compatibility Matrix,

[5] Texas Instrument, ADS1292R Datasheet,

[6] Nordic Semiconductor, Creating Bluetooth® Low Energy Applications Using nRF51822 V1.1,

2014-10.20.

[8] Nordic Semiconductor, Introduction to Bluetooth Smart,
https://www.youtube.com/watch?v=BZw0rQ6zkzE&index=2&list=PLx_tBuQ_KSdEe_8V10S1wfuYdNRAKx, early 2014.

[9] Nordic Semiconductor, nRF51 SDK 6.1.0 documentation,

[10] Nordic Semiconductor, nRF51 SDK 4.3.0 documentation,
