

Studiengang Systemtechnik

Vertiefungsrichtung Infotronics

Bachelorarbeit

Diplom 2021

Bastian Schwery

Dual Bluetooth 5 Stack on nRF53



Dozent
Medard Rieder



Experte
Jannic Schären



Datum der Abgabe des Schlussberichts
20.08.2021

Es handelt sich um den Originalbericht des/der Studierenden.
Er wurde nicht korrigiert und kann deshalb Ungenauigkeiten oder Fehler enthalten.

Filière / Studiengang SYND	Année académique / <i>Studienjahr</i> 2020/21	No TD / Nr. DA IT/2021/58
Mandant / Auftraggeber <input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>	Etudiant / Student Bastian Schwery	Lieu d'exécution / <i>Ausführungs</i> ort <input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>
	Professeur / Dozent Medard Rieder	
Travail confidentiel / <i>vertrauliche Arbeit</i> <input type="checkbox"/> oui / ja ¹ <input checked="" type="checkbox"/> non / nein	Expert / <i>Experte</i> (<i>données complètes</i>) Jannic Schären Moskitowatch AG, Mühlegasse 21, 3033 Wohlen bei Bern jschaeren@gmail.com	

<p>Titre / Titel Dual Bluetooth 5 Stack on NRF53</p> <p>Description / Beschreibung</p> <p>Nordic Semiconductor recently stated that further development of the traditional Bluetooth low Energy Stack, the so called Softdevice, will be discontinued and versions will be frozen. All new development will be done on ZEPHYR, a recent and very powerful RTOS. At the same time, Nordic has released a new Chip family, the NRF53xxx. This are dual core devices with one core being occupied with the Radio and one with the user software. The goal of this bachelor work is to develop a simple BT5 demonstrator using a NRF53 and Zephyr. The demonstrator is a light electrical vehicle (LEV) board computer: This means that it can be detected by a smartphone exposing a service similar to the ANT+ LEV service. But it must also be able to detect a number of sensors / actuators installed on the vehicle like cadence sensors, speed sensors and so on. This means that its Bluetooth stack has to be a dual stack and that it can play at the same time the role of a central and a peripheral Bluetooth device.</p> <p>Objectifs / Ziele</p> <ul style="list-style-type: none"> — Develop a Zephyr based central device able to use the CDC profile. — Develop a Zephyr based peripheral device exposing a service similar to the ANT+ LEV service. — Develop a simple Android application in order to monitor the LEV service — Merge the central and the peripheral device in order to obtain a hybrid device — Establish a technical documentation.
--

<p>Signature ou visa / Unterschrift oder Visum</p> <p>Responsable de l'orientation / filière <i>Leiter der Vertiefungsrichtung / Studiengang:</i></p>  <p>¹ Etudiant / Student : <u>Bastian</u></p>	<p>Délais / Termine</p> <p>Attribution du thème / <i>Ausgabe des Auftrags</i>: 10.05.2021</p> <p>Présentation intermédiaire / <i>Zwischenpräsentation</i> 07 – 08.06.2021</p> <p>Remise du rapport / <i>Abgabe des Schlussberichts</i>: 20.08.2021, 12:00</p> <p>Exposition / <i>Ausstellung der Diplomarbeiten</i>: 25 – 27.08.2021 (si autorisé / falls genehmigt)</p> <p>Défense orale / <i>Mündliche Verfechtung</i>: 30.08 – 09.09.2021</p>
--	---

¹ Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.



Applikation

Diplomarbeit | 2021 |

 Studiengang
Systemtechnik

Anwendungsbereich
Infotronics

Verantwortliche/r Dozent/in
Medard Rieder
medard.rieder@hevs.ch

Dual Bluetooth 5 Stack auf einem nRF53

 Diplomand Bastian Schwery

Ziel des Projekts

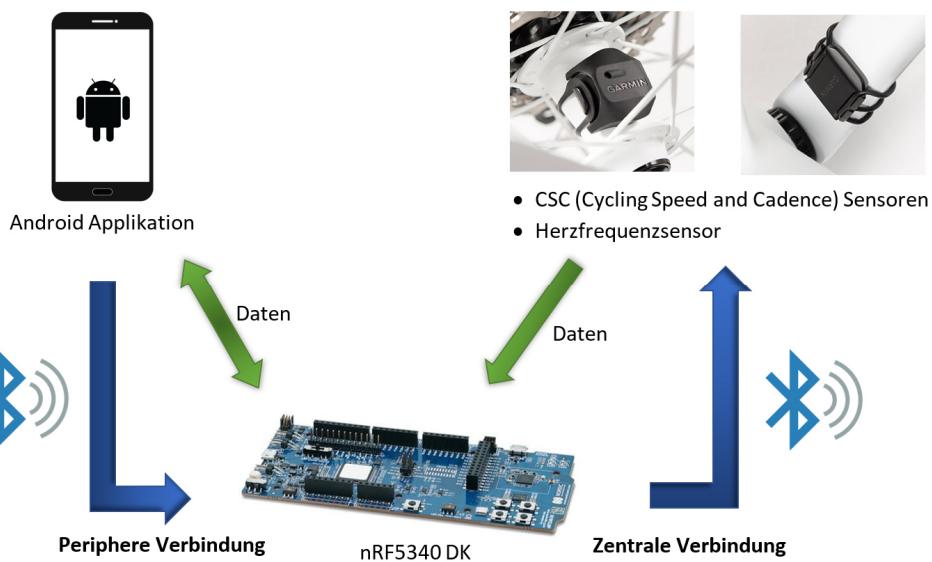
Das Ziel ist es, Informationen von Geschwindigkeit- und Trittfrequenzsensoren, welche an einem Fahrrad befestigt werden, über den nRF5340 an eine Android Applikation zu senden und diese dort anzuzeigen.

Methoden | Experimente | Resultate

Der nRF5340 ist das Kernelement der vorliegenden Arbeit. Mit ihm können alle notwendigen Bluetooth Verbindungen aufgebaut werden. Die Android Applikation, welche auch während der Bachelorarbeit entwickelt wurde, ist ein weiterer Hauptbestandteil. Mit ihr kann zuerst ausgewählt werden, mit welchem Board und zu welchen Sensoren eine Verbindung aufgebaut werden soll. Anschliessend werden diese Verbindungen hergestellt. Damit die Geschwindigkeit angezeigt werden kann, muss ein Raddurchmesser angegeben werden. Mit diesem wird auch die zurückgelegte Distanz berechnet.

Bei Bedarf kann zu Beginn ein Herzfrequenzsensor ausgewählt werden. Wird dieser korrekt getragen, sind die Schläge pro Minute ersichtlich.

Für den Nutzer der Applikation ist ausserdem der Batteriestand aller verbundenen Geräte ersichtlich.



Schema



Application

Travail de diplôme | 2021 |

Filière
Ingénieur en Systèmes Industriels

Domaine d'application
Infotronics

Professeur responsable
Medard Rieder
medard.rieder@hevs.ch

Dual Bluetooth 5 Stack sur un nRF53

Diplômant Bastian Schwery

Objectif du projet

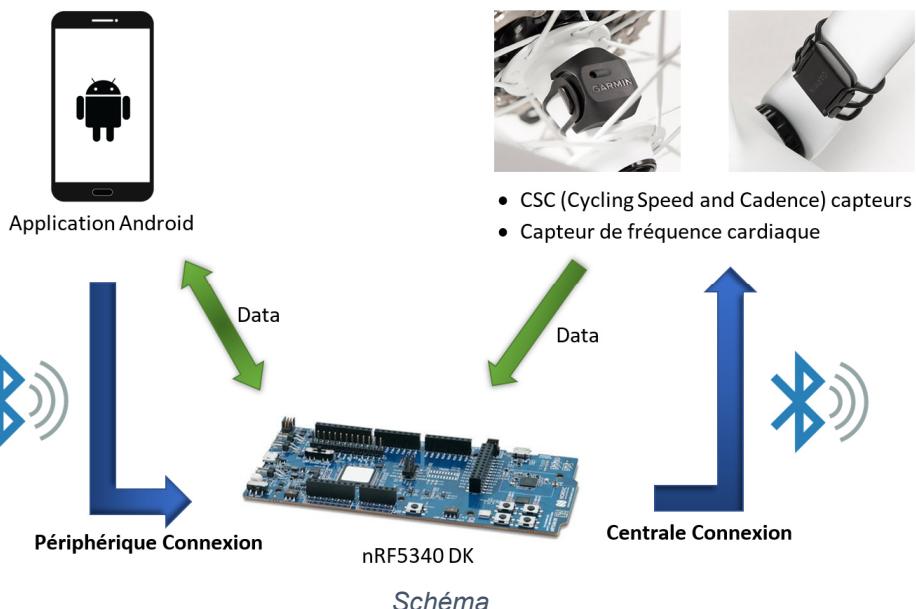
L'objectif est d'envoyer et d'afficher des informations provenant de capteurs de vitesse et de cadence de pédalage fixés sur un vélo à une application Android via le nRF5340.

Méthodes | Expériences | Résultats

Le nRF5340 est l'élément central de ce travail. Il peut être utilisé pour établir toutes les connexions Bluetooth nécessaires. Un autre composant principal est l'application Android, qui a également été développée pendant ce travail de bachelor. Elle permet tout d'abord de sélectionner la carte et les capteurs sur lesquels on souhaite établir une connexion. Ces connexions sont ensuite établies. Pour que la vitesse soit affichée, un diamètre de roue doit être spécifié. Elle est également utilisée pour calculer la distance parcourue.

Si nécessaire, un capteur de fréquence cardiaque peut être sélectionné au début. Si celui-ci est porté correctement, les battements par minute s'affichent.

L'utilisateur de l'application peut également voir l'état de la batterie de tous les appareils connectés.



Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während meiner Bachelorarbeit unterstützt haben.

Zuerst gebührt mein Dank Herrn Rieder Medard, der meine Bachelorarbeit betreut hat. Ebenfalls möchte ich mich bei Rudaz Patrice, Rossier Yoan und Clausen Michael, welche mir bei Fragen hilfreich zur Seite standen, bedanken.

Zudem richte ich auch ein Dankeschön an meine Freunde für das Korrekturlesen meiner Arbeit.

Abschliessend möchte ich mich bei meinen Eltern und Angehörigen bedanken, die mich auf diesem Weg unterstützen.

Inhaltsverzeichnis

Danksagung	I
Abbildungsverzeichnis	IV
Tabellenverzeichnis	IV
1 Einführung	1
1.1 Projektbeschreibung	1
1.2 Projektziel	1
1.3 Aufbau der Arbeit	2
2 Komponenten	3
2.1 nRF5340 Entwicklungs-Kit	3
2.2 Geschwindigkeits- und Trittfrequenzsensor	4
2.3 Herzfrequenzsensor	4
2.4 Verwendete Softwares	4
3 Installation von Zephyr auf Windows 10	5
4 Nordic SDK in Visual Studio Code	7
5 Klassendiagramm nRF5340 Teil	7
6 Bluetooth Low Energy (BLE)	8
6.1 GAP Schicht	8
6.2 GATT Schicht	9
6.3 BLE: Peripherie	10
6.4 BLE: Zentral	13
6.5 Batterie Service Client	22
6.6 Data Service (Server)	24
6.7 Verwendung des Data Service	27
6.8 Konfigurationsmöglichkeiten	28
7 Android Applikation	29
7.1 Voreinstellungen	30
7.2 Scan – und Verbindungsvorgang	31
7.3 Data Service (Klient)	32
7.4 Senden / Empfangen von Daten	33
7.5 Nachrichtenanzeige	34
7.6 Referenzen Applikation	35
8 Datenfluss	36
9 Tests	37
10 Schlussfolgerung	38
11 Selbständigkeitserklärung	39

12	Anhang.....	40
13	Quellenverzeichnis	52
14	Literaturverzeichnis	53

Abbildungsverzeichnis

<i>Abbildung 1: Schema zur Darstellung des Projektziels.....</i>	2
<i>Abbildung 2: nRF5340 DK.....</i>	3
<i>Abbildung 3: Geschwindigkeits- und Trittfrequenzsensor.....</i>	4
<i>Abbildung 4: Herzfrequenzmesser Polar</i>	4
<i>Abbildung 5: BLE Schichtenmodell</i>	8
<i>Abbildung 6: Erklärung GAP</i>	9
<i>Abbildung 7: Erklärung GATT</i>	9
<i>Abbildung 8: Nordic Thingy:52</i>	13
<i>Abbildung 9: Berechnung RPM.....</i>	20
<i>Abbildung 10: Berechnung Geschwindigkeit.....</i>	21
<i>Abbildung 11: Sequenzdiagramm Battery Manager.....</i>	23
<i>Abbildung 12: Splash Screen Activity</i>	29
<i>Abbildung 13: Scanner Activity.....</i>	29
<i>Abbildung 14: CSC Activity</i>	29
<i>Abbildung 15: Sequenzdiagramm Scan- und Verbindungsvorgang</i>	31
<i>Abbildung 16: Sequenzdiagramm Senden / Empfangen von Daten</i>	34
<i>Abbildung 17: Sequenzdiagramm Datenfluss</i>	36
<i>Abbildung 18: Test mit Thingy und der "nRF Connect" Applikation.....</i>	37

Tabellenverzeichnis

<i>Tabelle 1: Verwendete Software</i>	4
<i>Tabelle 2: Nachrichtencodes</i>	35

1 Einführung

1.1 Projektbeschreibung

Die Firma Nordic Semiconductor¹ [1] hat vor kurzem erklärt, dass die Weiterentwicklung des traditionellen Bluetooth Low Energy² Stacks, dem sogenannten «*Softdevice*», eingestellt wird und die Versionen eingefroren werden. Alle Neuentwicklungen werden auf Zephyr, einem aktuellen und sehr leistungsfähigen RTOS³, durchgeführt. Gleichzeitig hat Nordic eine neue Chip-Familie entwickelt, die nRF53xxx. Dabei handelt es sich um Dual-Core-Bausteine, wobei ein Kern mit dem Radio- und einer mit der Anwendersoftware belegt ist.

1.2 Projektziel

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines einfachen BT5-Demonstrators⁴ mit einem nRF5340 Entwicklungs-Kit und Zephyr. Als Vorführobjekt dient ein E-Bike der HES-SO Valais-Wallis. Der nRF5340 muss von einem Smartphone erkannt werden können, aber auch in der Lage sein, eine Reihe von Sensoren zu erkennen und den Datenaustausch mit diesen zu ermöglichen. Es handelt sich um Geschwindigkeits-, Trittfrequenz- und Herzfrequenzsensoren. Das bedeutet, dass er gleichzeitig die Rolle eines zentralen und eines periphereren Bluetooth-Geräts spielen muss und so als Gateway⁵ fungiert.

Im folgenden Schema (Abbildung 1) ist das zuvor beschriebene Projektziel ersichtlich. Die Hauptkomponente ist das nRF5340 Entwicklungs-Kit. Die Daten der Sensoren werden über das Kit an die Android Applikation weitergesendet.

¹ Norwegischer Halbleiterhersteller

² Stromsparende Version von Bluetooth

³ Real Time Operating System (Echtzeit Betriebssystem)

⁴ Bluetooth 5

⁵ Stellt eine Verbindung zwischen zwei unterschiedlichen Systemen her.

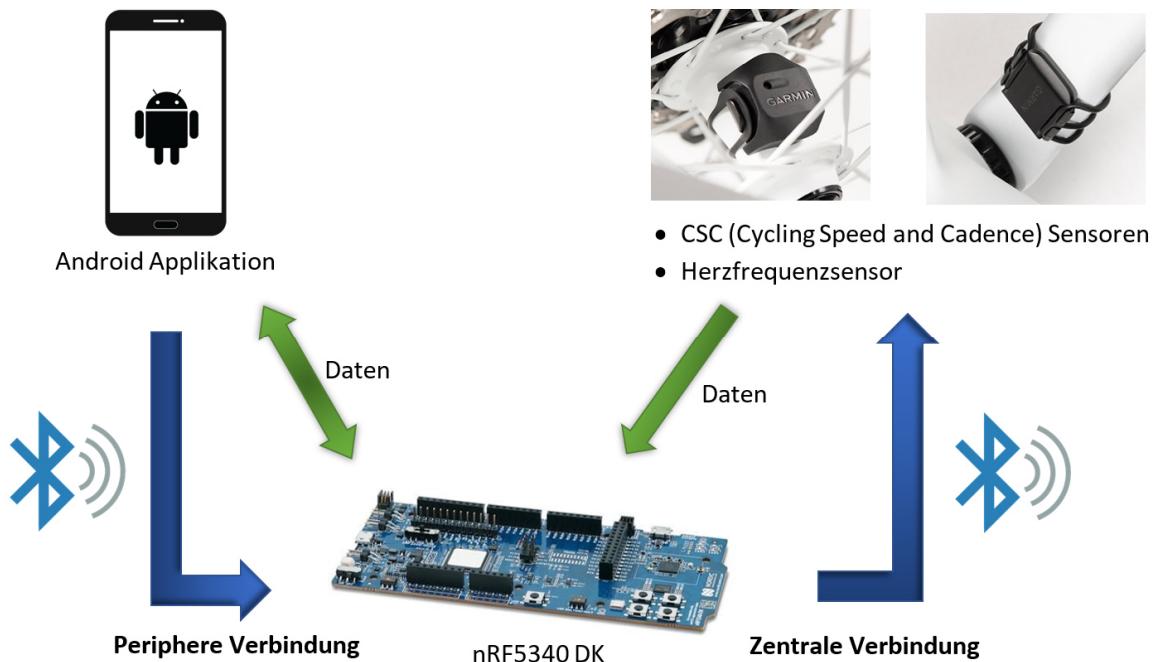


Abbildung 1: Schema zur Darstellung des Projektziels

Quelle: <https://play-lh.googleusercontent.com/WshzYj8MVD1cDz55UP3z0Hl3eqUiBlkez5GSYr0yxwGSbVTkdLILlb0m0SXkqX-tWC3>

Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK_SPL.jpg_3

Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK_SPL.jpg_3

1.3 Aufbau der Arbeit

Die folgenden Schritte werden näher erläutert:

- Komponenten des Schemas (Seite 2)
- Installation von Zephyr auf Windows 10 (Seite 5)
- Vorbereitungen für die Verwendung von Visual Studio Code (Seite 7)
- Ausführliche Erklärung, um eine periphere und zentrale Bluetooth Verbindung aufzubauen (Seite 8)
- Erklärung zur Verwendung des Batterie Services von Nordic (Seite 22)
- Implementierung eines Services zur Übermittelung von Daten (Seite 24)
- Programmierung der Applikation in Android Studio (Seite 29)

Das Vorgehen der vorliegenden Arbeit wird durch den Zeitplan, welcher sich im Anhang 1 befindet, kontrolliert.

2 Komponenten

Das Kapitel 2 hat zum Ziel, die einzelnen Komponenten des Projektes zu erläutern.

- nRF5340 Entwicklungs-Kit
- Geschwindigkeit- und Trittfrequenzsensoren
- Herzfrequenzsensor
- Verwendete Softwares

2.1 nRF5340 Entwicklungs-Kit

Das nRF5340 DK (Abbildung 2) ist das Entwicklungs-Kit für den nRF5340 System-on-Chip (SoC) und enthält alles, was für den Einstieg in die Entwicklung benötigt wird, auf einer einzigen Platine.

Im Folgenden werden nur die Punkte aufgelistet, welche für das Projekt dieser Bachelorarbeit eingesetzt wurden. Der nRF5340 SoC ist der Kern des Entwicklungskits. Er kombiniert einen Hochleistungs-Anwendungsprozessor mit einem voll programmierbaren Ultra-Low-Power-Netzwerkprozessor. Der nRF5340 unterstützt Bluetooth Low Energy. Das Entwicklungs-Kit wird via USB programmiert und mit Strom versorgt. Es gibt eine Batterie, welche bei Bedarf für die Stromversorgung verwendet werden kann. Auf dem Board sind vier programmierbare Taster und LEDs verbaut. [2]

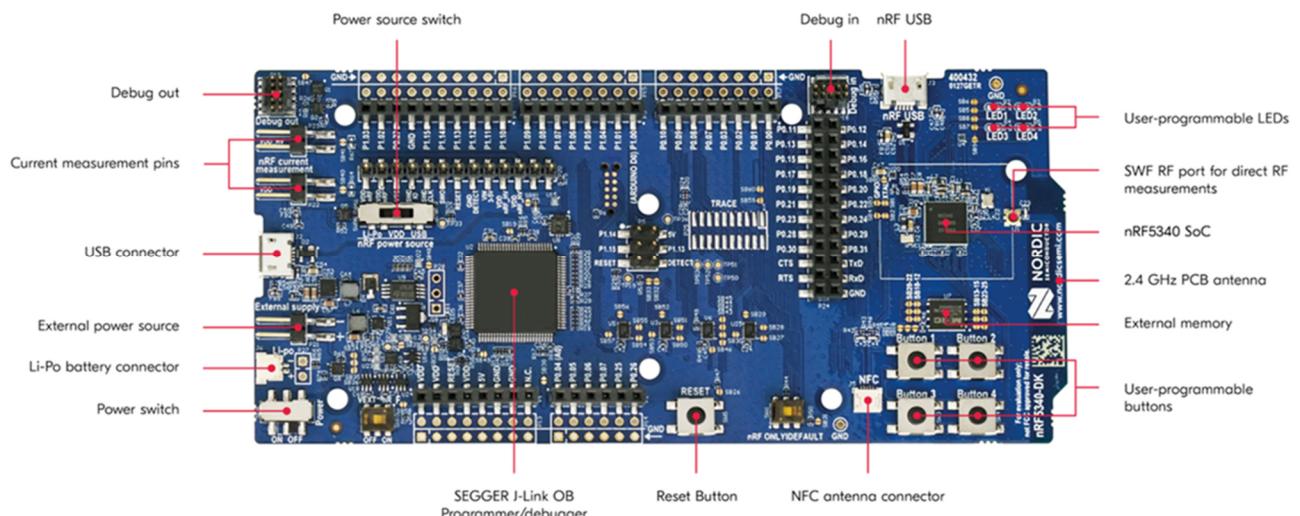


Abbildung 2: nRF5340 DK

Quelle: https://www.mouser.de/images/marketingid/2019/microsites/153282850/2020-12-21_15-07-43.png

2.2 Geschwindigkeits- und Trittfrequenzsensor

Die verwendeten CSC⁶ Sensoren (Abbildung 3) stammen von Garmin [3]. Garmin gehört zu den weltweit führenden Unternehmen im Bereich Navigation für die Automobilbranche, sowie für die Bereiche Luftfahrt, Marine und Outdoor / Sport.

Der Geschwindigkeitssensor wird auf eine der Radnaben⁷ montiert. Der Trittfrequenzsensor wird seitlich an der Tretkurbel befestigt. Die gemessenen Daten werden per Bluetooth an den nRF5340 gesendet.



Abbildung 3: Geschwindigkeits- und Trittfrequenzsensor

Quelle: https://static.garmincdn.com/en/products/010-12103-00/g/cadence_v03-eb1598f4-1e01-4e01-8691-8ddf6e108677.jpg 5

2.3 Herzfrequenzsensor

Der verwendete Herzfrequenzsensor stammt von der Firma Polar [4]. Polar ist seit 1977 führend im Bereich der Entwicklung von Herzfrequenzmessgeräten und technischen Innovationen [5]. Das Messgerät wird von der Person, die das Fahrrad fährt, um den Brustkorb getragen.



Abbildung 4: Herzfrequenzmesser Polar

Quelle: <https://media.alltricks.com/hd/15743025eaaa35aa3e5d9.44386646.jpg> 6

2.4 Verwendete Softwares

Name	Version
Windows	10 pro
Microsoft Office	365
Visual Studio Code	1.56.2
Android Studio	4.2.1

Tabelle 1: Verwendete Software

⁶ Cycling Speed and Cadence (Geschwindigkeit und Trittfrequenz beim Radfahren)

⁷ Zentrum eines Rades

3 Installation von Zephyr auf Windows 10

Im Folgenden ist eine Anleitung wie Zephyr auf Windows 10 installiert werden kann. Am Ende wird zum Testen ein Beispielcode auf das Entwicklungs-Kit geladen. Die Anleitung kann auf der Webseite [6] von Nordic gefunden werden.

Benötigte Anwendungen:

- CMD⁸: Bereits vorhanden
- Windows PowerShell: Bereits vorhanden
- Chocolatey⁹: Nicht vorhanden

Installation von Chocolatey

1. Windows PowerShell als Administrator öffnen
2. «Get-ExecutionPolicy» eingeben, wenn die Konsole «restricted» zurückgibt, muss Punkt 2.1 ausgeführt werden, ansonsten nicht.
 2.1. «Set-ExecutionPolicy Bypass -Scope Process» eingeben und die Anfrage mit «Y» bestätigen
3. Nun kann Chocolatey effektiv installiert werden. Dies geschieht mit folgendem Befehl:
`Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))`

Wenn keine Fehler erscheinen, ist Chocolatey installiert und kann verwendet werden.

Installation von Zephyr und anderen Tools

1. Windows PowerShell als Administrator öffnen
2. Folgende Befehle im Windows PowerShell eingeben:
 - 2.1. «choco feature enable -n allowGlobalConfirmation»
 - 2.2. «choco install cmake --installargs 'ADD_CMAKE_TO_PATH=System'»
 - 2.3. «choco install ninja gperf python git»
 - 2.4. Warten bis die Installation beendet ist, dies wird einen Moment dauern.
3. Windows PowerShell schliessen
4. Neuen Ordner im C: Verzeichnis erstellen: «c:\%HOMEPATH%\<zephyrfolder>» (zephyrfolder = ncs)
5. CMD als Administrator öffnen und folgende Befehle ausführen
 - 5.1. «pip3 install west»
 - 5.2. «cd %HOMEPATH%\<zephyrfolder>»
 - 5.3. «west init -m https://github.com/nrfconnect/sdk-nrf --mr v1.6.1»
 - 5.4. «west update»
 - 5.5. «west zephyr-export»
 - 5.6. «pip3 install -r %HOMEPATH%\<zephyrfolder>\zephyr\scripts\requirements.txt»
 - 5.7. «pip3 install -r nrf/scripts/requirements.txt»
 - 5.8. «pip3 install -r bootloader/mcuboot/scripts/requirements.txt»
 - 5.9. Warten bis die Installation beendet ist, dies wird einen Moment dauern.
 - 5.10. Im Verzeichnis «ncs» müssten jetzt unteranderem die Unterordner «zephyr» und «nrf» sein

Nun ist Zephyr und West installiert, fehlt noch die GNU Toolchain¹⁰.

⁸ Windows-Eingabeaufforderung

⁹ Software-Verwaltungslösung

¹⁰ GNU-Werkzeugkette, Programmierwerkzeug [29]

GNU Tools

1. Den folgenden Installer herunterladen:
<https://developer.arm.com/-/media/Files/downloads/gnu-rm/10-2020q4/gcc-arm-none-eabi-10-2020-q4-major-win32.exe?revision=9a4bce5a-7577-4b4f-910d-4585f55d35e8&la=en&hash=068C813EEFB68060B5FB40E6541BDE7159AFAA0>
2. Einen neuen Ordner im C: Verzeichnis: «c:\gnu_arm_embedded» erstellen.
3. «Downloads» Ordner öffnen.
4. Die vorhin heruntergeladen Datei «gcc-arm-none-eabi-10-2020-q4-major-win32.exe» ausführen.
5. Als Speicherort den zuvor erstellten Ordner auswählen «C:\gnu_arm_embedded», und das automatisch hinzugefügte Unterverzeichnis entfernen.
6. Am Ende das Feld zum Hinzufügen der UmgebungsvARIABLE aktivieren.
7. Warten, bis die Installation beendet ist.

JLink

1. Den folgenden Installer herunterladen:
https://www.nordicsemi.com/-/media/Software-and-other-downloads/Desktop-software/nRF-command-line-tools/sw/Versions-10-x-x/10-12-1/nRF-Command-Line-Tools_10_12_1_Installer_64.exe
2. «Downloads» Ordner öffnen
3. Die vorhin heruntergeladene Datei «nRF-Command-Line-Tools_10_12_1_Installer_64.exe» ausführen.
4. Alle Voreinstellungen beibehalten.

Umgebungsvariablen

1. Folgende Umgebungsvariablen in den Einstellungen definieren
 - 1.1. «GNUARMEMB_TOOLCHAIN_PATH=C:\gnu_arm_embedded»
 - 1.2. «ZEPHYR_TOOLCHAIN_VARIANT=gnuarmemb»

Test

Als Beispielprojekt wurde das «blinky» Projekt von Zephyr verwendet. Dieses befindet sich im Ordner «%HOMEPATH%\<zephyrfolder>\zephyr\samples\basic\blinky» oder auf Github [7]. Um das Projekt zu kompilieren¹¹ und auszuführen müssen folgende Befehle in der Konsole eingegeben werden.

1. «cd %HOMEPATH%\<zephyrfolder>\zephyr»
2. «zephyr-env.cmd»
3. «west build -p auto -b <boardname> samples\basic\blinky» (Hier wird das Board "nrf5340dk_nrf5340_cmuappns" verwendet)
4. «west flash»

Als Resultat blinkt die LED1 auf dem Development Kit im Sekundentakt.

¹¹ Quellcode in eine für den Computer verständliche Programmiersprache übersetzen.

4 Nordic SDK in Visual Studio Code

Im Folgenden wird erklärt, wie ein Projekt in Visual Studio Code kompiliert, auf das Board geladen und gedebuggt¹² wird. Das Projekt muss sich im zuvor erstellten Verzeichnis befinden. «%HOMEPATH%\<zephyr folder>\».

Für dieses Beispielprojekt wurde ein Workspace, namens «PerCen.code-workspace» eingerichtet. Eine Vorlage von Rieder Medard wurde bereitgestellt. Diese Vorlage wurde angepasst und befindet sich im Anhang Ordner «Anhänge -> nRF5340_code». Durch das Einfügen der Datei im Projektordner, kann durch ein Doppelklick Visual Studio Code geöffnet werden. Ausgehend in welchem Unterordner das Projekt erstellt wurde, müssen die Pfade für den «zephyr» Ordner und den des «nrf» Ordners angepasst werden. Außerdem müssen die Pfade für die «commands» entsprechend geändert werden.

Nun muss noch eine «CMakeList.txt» Datei erstellt werden. Diese Datei teilt dem Build-System mit, wo die anderen Anwendungsdateien zu finden sind, und verknüpft das Anwendungsverzeichnis mit dem CMake-Build-System von Zephyr. [8] Hier müssen einige Zeilen hinzugefügt werden, welche untenstehend zu sehen sind. Wird nicht das nRF5340 Development Kit verwendet, muss der Board Name angepasst werden. Eine Liste aller Namen ist auf der Webseite [9] von Nordic ersichtlich. Außerdem müssen alle Klassen, welche kompiliert werden sollen, hier aufgelistet werden.

```

cmake_minimum_required(VERSION 3.13.1)
set(BOARD nrf5340dk_nrf5340_cmuappns)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(peripheral_1bs)

# NORDIC SDK APP START
target_sources(app PRIVATE
    src/main.cpp src/DeviceManager.h src/DeviceManager.cpp src/Data.h
    src/Data.cpp src/DataService.h src/DataService.cpp src/BatteryManager.h
    src/BatteryManager.c
)
# NORDIC SDK APP END
zephyr_library_include_directories(.)

```

Jetzt sind alle Voreinstellungen abgeschlossen. Über den Menu Punkt «Terminal -> Run Task» kann die gewünschte Operation ausgewählt werden. Die Befehle hinter diesen Auswahlmöglichkeiten befinden sich in der «PerCen.code-workspace» Datei. Zuerst muss der Code kompiliert werden. Soll der Debug Modus gestartet werden, muss in der Menu Leiste «Run -> Start Debugging» gewählt werden.

5 Klassendiagramm nRF5340 Teil

Die Kernklasse ist die «DeviceManager» Klasse. Hier werden die Bluetooth Verbindungen aufgebaut und die notwendigen Charakteristiken eingeschrieben. Die Daten der Sensoren werden hier empfangen und anschliessend in der «DataCSC» Klasse gespeichert. Hier befinden sich die Berechnungen für die Geschwindigkeit und die Trittfrequenz. Um die Daten weiter an die Android Applikation zu senden, wurde ein Service implementiert. Dieser befindet sich in der «DataService» Klasse. Um den aktuellen Batteriestand der Sensoren zu kennen, wurde der «GATT Battery Service Client» (BAS) [10] von Nordic verwendet. Hierfür wurde eine Klasse «BatteryManager» erstellt. Zudem gibt es eine Klasse «main». Hier wird entschieden, ob der nRF5340 als zentrales und/oder als peripheres Gerät initialisiert wird. Im Anhang 2 befindet sich das Klassendiagramm. Im Folge dieses Berichts wird auf die einzelnen Klassen eingegangen.

¹² Methode um einfach und effizient Fehler im Code zu lokalisieren.

6 Bluetooth Low Energy (BLE)

Folgende Aufgaben werden im BLE Teil behandelt:

- Initialisierung des nRF5340 DK, sodass es als peripheres und zentrales Gerät arbeiten kann
- Werben (Advertising) des Boards, sodass sich eine Applikation (zentrales Gerät) mit ihm verbinden kann (Peripherer Teil)
- Scan nach den zu verbindenden Sensoren (Zentral Teil)
- Verbindungsaufbau mit den Sensoren (Zentral Teil)
- Datenaustausch: Sensoren → nRF5340 DK → Applikation

Ein Sequenzdiagramm, welches den Ablauf dieser Schritte zeigt, befindet sich im Anhang 3. Auf die einzelnen Schritte wird in den folgenden Unterkapitel eingegangen.

Bluetooth Low Energy (BLE) ist eine Funktechnologie, welche für den geringen Stromverbrauch optimiert ist. Sie ermöglicht die Übertragung von Daten von einem Gerät zu einem anderen, z. B. die Übertragung der Temperatur von einem Sensor an ein anderes Gerät, welches diese Daten sammelt. Die BLE-Architektur besteht aus mehreren Schichten (Abbildung 5). [11]

In den folgenden Abschnitten wird auf zwei dieser Schichten eingegangen.

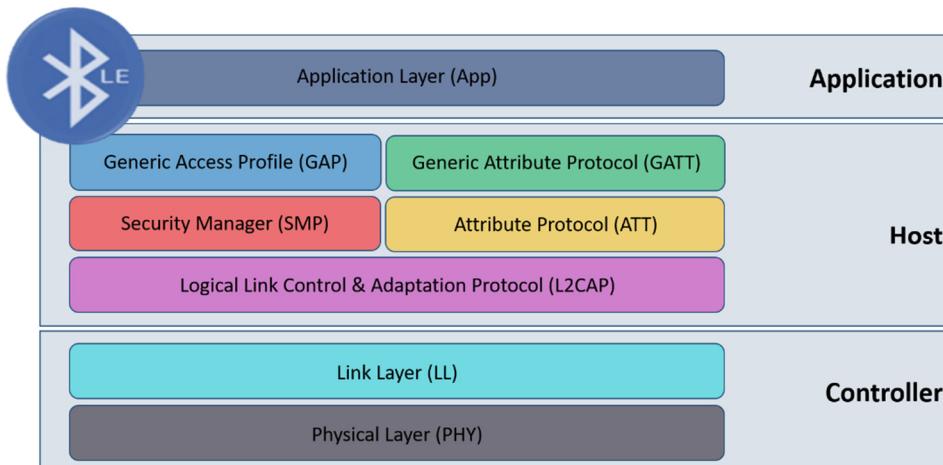


Abbildung 5: BLE Schichtenmodell

Quelle: https://miro.medium.com/max/1838/1*7R_hkwuk8v7gClsvXtoqPw.png 7

6.1 GAP Schicht

Die Bezeichnung GAP steht für:

«*Generic Access Profil*», was übersetzt «*Generisches Zugriffsprofil*» bedeutet. Es steuert die Verbindungen. GAP macht das Gerät für die Außenwelt sichtbar und bestimmt wie zwei Geräte miteinander interagieren können. (Abbildung 6) [12]

Die meisten BLE Geräte besitzen zwei Rollen, entweder eine zentrale oder periphere Rolle. Der nRF5340 kann beides zur selben Zeit. Das bedeutet, dass der Chip zuerst um sich wirbt (advertising), sodass sich eine Applikation mit dem Board verbinden kann. Anschliessend scannt der nRF5340 nach den gewünschten Sensoren und baut eine Verbindung zu diesen auf.

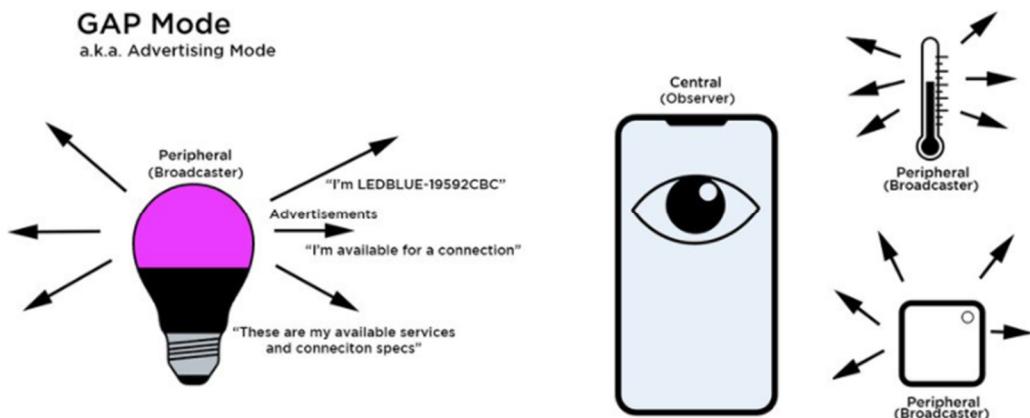


Abbildung 6: Erklärung GAP

Quelle: https://learn.adafruit.com/assets/86833_8

6.2 GATT Schicht

Die Bezeichnung GATT «*Generic Attribute Profile*», bedeutet «*Generisches Attributprofil*». Es bezeichnet die Art und Weise, wie zwei Bluetooth Low Energy Geräte Daten hin und her übertragen. [13]

Nach dem Verbindungsauflauf mit einem Bluetooth Low Energy Gerät, stellt dieses eine Auswahl von «*Generic Attribute Profile Services*» zur Verfügung. Jeder Service ist durch eine eindeutige Nummer (UUID¹³) gekennzeichnet und umfasst mehrere «*GATT Characteristics*». Jede Charakteristik¹⁴ ist wiederum durch eine UUID, mehreren Flags wie z.B. «*read*» oder «*write*» und durch ein Datenwert (Byte Array¹⁵) definiert. Dies ist in Abbildung 7 ersichtlich.

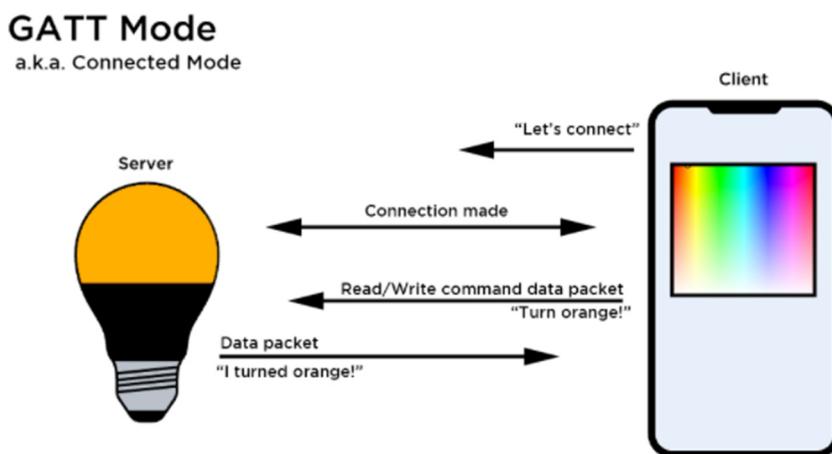


Abbildung 7: Erklärung GATT

Quelle: https://learn.adafruit.com/assets/86833_9

¹³ Universally Unique Identifier (Universell eindeutiger Bezeichner)

¹⁴ Eigenschaft

¹⁵ Zusammenfassung mehrerer Objekte eines bestimmten Datentyps

6.3 BLE: Peripherie

In diesem Abschnitt wird alles rund um den Verbindungsaufbau zwischen dem nRF5340 und einem zentralen Gerät (Hier ein Smartphone) genauer erläutert. Die notwendigen Informationen wurden vom «*peripheral_lbs*» [14] Beispiel von Nordic hinzugezogen. Dieser Code befindet sich in der «*DeviceManager*» Klasse. Bevor auf die einzelnen Schritte eingegangen werden kann, müssen einige Voreinstellungen getroffen werden. Wenn die «*prj.conf*» Datei noch nicht vorhanden ist, muss diese im Projektordner erstellt werden. Anschliessend werden folgende Konfigurationen eingestellt.

```
# General Bluetooth settings
CONFIG_NCS_SAMPLES_DEFAULTS=y
CONFIG_BT=y
CONFIG_SETTINGS=y
CONFIG_BT_SETTINGS=y
CONFIG_BT_PERIPHERAL=y
CONFIG_BT_CENTRAL=y
CONFIG_BT_GATT_DM=y
CONFIG_BT_GATT_CLIENT=y
CONFIG_BT_MAX_CONN=5
CONFIG_BT_DEVICE_NAME="Nordic nRF5340 DK"

# Enable the LBS service
CONFIG_BT_LBS=y
CONFIG_BT_LBS_POLL_BUTTON=y
CONFIG_DK_LIBRARY=y

# Enable the BAS service
CONFIG_BT_BAS_CLIENT=y

# Scan Settings
CONFIG_BT_SCAN=y
CONFIG_BT_SCAN_FILTER_ENABLE=y
CONFIG_BT_SCAN_UUID_CNT=2

# Console settings
CONFIG_RTT_CONSOLE=y
CONFIG_USE SEGGER_RTT=y
CONFIG_UART_CONSOLE=n
```

Diese Konfigurationen sind dringend notwendig.

Des Weiteren müssen einige Klassen eingebunden werden, ansonsten zeigt der Kompilierer an, dass er gewisse Funktionen und Definitionen nicht kennt. Diese «*Includes*» befinden sich alle in der «*DeviceManager.h*» Datei.

```
#include "DataCSC.h"
#include "DataService.h"
extern "C"
{
    #include "BatteryManager.h"
}
#include <bluetooth/services/lbs.h>
#include <dk_buttons_and_leds.h>
#include <settings/settings.h>
#include <bluetooth/scan.h>
```

Nun, da alle Konfigurationen eingestellt und die nötigen Klassen eingebunden sind, kann in einem ersten Schritt der nRF5340 als peripheres Gerät konfiguriert werden. Mit einer bereits existierenden Applikation von nRF, «*nRF Connect*» [15], kann dieser Teil getestet werden. Es werden nur die elementaren Teile zum Verbindungsauflaufbau erklärt.

1. Bluetooth aktivieren:

```
err = bt_enable(NULL);
```

Diese Funktion gibt den Wert '0' zurück, wenn die Aktivierung erfolgreich war. Als Parameter kann eine Funktion angegeben werden, welche aufgerufen wird, sobald dass die Aktivierung abgeschlossen ist.

2. Rückrufstruktur (callback) muss definiert werden:

```
struct bt_conn_cb conn_callbacks = {
    .connected = connected,
    .disconnected = disconnected,
};
```

Danach müssen noch die Funktionen «*connected*» und «*disconnected*» deklariert und implementiert werden. Dies geschieht in einem späteren Schritt.

Jetzt müssen die Rückruffunktionen noch eingeschrieben werden. Dies geschieht über die Funktion:

```
bt_conn_cb_register(&conn_callbacks);
```

Als Parameter wird die Adresse der Struktur mit den Rückruffunktionen übergeben.

3. Der nRF5340 muss beginnen um sich zu werben, dem sogenannten «*advertising*»:

```
int bt_le_adv_start(const struct bt_le_adv_param *param,
                    const struct bt_data *ad, size_t ad_len,
                    const struct bt_data *sd, size_t sd_len);
```

Diese Methode besitzt einige Parameter. Als erstes muss der Methode eine Struktur mit den notwendigen Informationen übergeben werden. Hier wurde ein bereits existierendes «*define*» verwendet, nämlich «*BT_LE_ADV_CONN*».

Die Strukturen «*ad*» und «*sd*» sind Daten Arrays, die in Werbepaketen (advertising), bzw. Scan- Antwortpaketen (scanning) verwendet werden. Diese werden wie folgt deklariert.

```

static const struct bt_data ad[] = {
    BT_DATA_BYT(0x00, (BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
    BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_NAME_LEN),
};

static const struct bt_data sd[] = {
    BT_DATA_BYT(0x00, (BT_DATA_UUID128_ALL, DATA_SERVICE_UUID)),
};

```

Die zwei verbleibenden Parameter beinhalten die Informationen, über die Grösse dieser Strukturen. Der gesamte Methodenaufruf sieht nun folgendermassen aus.

```

err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY_SIZE(ad),
                      sd, ARRAY_SIZE(sd));

```

Diese Funktion gibt den Wert '0' zurück, wenn der «*advertising*» Vorgang erfolgreich gestartet wurde, ansonsten einen negativen Fehlercode.

4. Sobald sich nun ein zentrales Gerät, hier ein Smartphone, mit dem nRF5340 verbindet, wird die Rückruffunktion «*connected*» aufgerufen. Der Prototyp dieser Funktion sieht wie folgt aus:

```
static void connected(struct bt_conn *conn, uint8_t err);
```

Der erste Parameter ist die Adresse der Verbindung, die aufgebaut wurde. Der zweite Parameter ist ein Fehlercode, wenn dieser Wert '0' beträgt, wurde die Verbindung erfolgreich aufgebaut.

Nun muss der Unterschied zwischen einer zentralen und periphereren Verbindung gemacht werden. Dies wird wie folgt erreicht:

```

bt_conn_info info;
int error = bt_conn_get_info(conn,&info);

```

Mittels der Funktion «*bt_conn_get_info*», welche als Parameter die Adressen der bestehenden Verbindung und die einer «*bt_conn_info*» Struktur nimmt, können diverse Informationen der Verbindung gewonnen werden. Die Funktion gibt den Wert '0' zurück, wenn der Vorgang erfolgreich war.

Mit folgender Abfrage kann getestet werden, ob es sich beim verbundenen Gerät um ein zentrales oder ein peripheres Gerät handelt.

```
if (info.role == BT_CONN_ROLE_SLAVE)
```

In dieser Abfrage können nun diverse Anzeigemöglichkeiten hinzugefügt werden, welche dem Nutzer zeigen, dass ein peripheres Gerät verbunden ist.

Im nächsten Schritt wird die Konfiguration als «*zentral*» hinzugefügt.

6.4 BLE: Zentral

In diesem Abschnitt wird auf den Verbindungsaufbau zwischen einem peripheren Gerät und dem nRF5340 eingegangen. Es werden zwei verschiedene Varianten vorgestellt. Bei der ersten Variante wurde als Testgerät ein Nordic Thingy:52 (Abbildung 8) verwendet. Die zweite Variante ist diejenige, welche mit den verwendeten Sensoren benutzt wird.

Variante mit Thingy:

«Das Nordic Thingy:52 ist ein kompaktes, energieoptimiertes Multi-Sensor-Entwicklung-Kit. Es ist eine einfach zu bedienende Entwicklungsplattform, die Entwicklern hilft, IoT-Prototypen und -Demos zu erstellen, ohne die Notwendigkeit, eigene Hardware zu bauen.» [16]

Es wird nur auf die elementaren Teile zum Verbindungsauflaufbau eingegangen. Die nötigen Informationen wurden vom «central» Beispiel von Zephyr hinzugezogen. Dieses befindet sich auf Github. [17]

Da bei der Initialisierung des peripherie Teils alle allgemeinen Initialisierungen ausgeführt wurden, sind diese hier nicht mehr notwendig und es kann direkt mit der Initialisierung des Scan-Vorgangs begonnen werden.



Abbildung 8: Nordic Thingy:52

Quelle:

https://www.mouser.ch/images/marketingid/2017/img/116326113_Nordic_Thingy52IoTSensorDevelopmentKit.png?v=052820.0204.10

- Um den Scan Vorgang zu starten, braucht es eine Struktur mit den nötigen Informationen:

```
struct bt_le_scan_param scanParam = {
    .type = BT_LE_SCAN_TYPE_ACTIVE,
    .options = BT_LE_SCAN_OPT_FILTER_DUPLICATE,
    .interval = BT_GAP_SCAN_FAST_INTERVAL,
    .window = BT_GAP_SCAN_FAST_WINDOW,
    .timeout = 0
};
```

Anstatt einer Struktur können auch die default Einstellungen gewählt werden.

Es wird eine Rückruffunktion benötigt. Sobald ein Gerät gefunden wird, wird diese Funktion aufgerufen.

```
static void deviceFound(
    const bt_addr_le_t *addr, int8_t rssi, uint8_t type,
    struct net_buf_simple *ad);
```

Die Parameter enthalten Informationen wie die Adresse, den RSSI Wert, den Typ, und einen Zeiger auf einen Puffer. Der RSSI Wert ist der Leistungspegel und gibt an, wie nah oder fern ein Gerät ist.

Nun kann ein Scan Vorgang gestartet werden:

```
err = bt_le_scan_start(&scanParam, deviceFound);
```

Die Funktion gibt den Wert '0' zurück, wenn der Scan Vorgang erfolgreich gestartet wurde.

2. Sobald ein Gerät in der Nähe gefunden wird, wird die Funktion «*deviceFound*» aufgerufen. Hier wird zuerst kontrolliert, ob zum gefundenen Gerät eine Verbindung aufgebaut werden kann.

```
if (type != BT_GAP_ADV_TYPE_ADV_IND &&
    type != BT_GAP_ADV_TYPE_ADV_DIRECT_IND) {
    return;
}
```

Ausserdem soll nur eine Verbindung aufgebaut werden, wenn sich das Gerät nahe beim Entwicklungs-Kit befindet.

```
if (rssi < -70) {
    return;
}
```

Je kleiner der RSSI Wert, desto näher befindet sich das Gerät. Die Einheit sind Dezibel. Ein Wert von -70db entspricht einer noch knapp guten Verbindung.

Wenn ein verbindbares, in der Nähe liegendes Gerät gefunden wurde, kann der Scan Vorgang abgebrochen werden.

```
if (bt_le_scan_stop()) {
    return;
}
```

Die Funktion gibt den Wert '0' zurück, wenn der Scan Vorgang erfolgreich abgebrochen wurde.

Anschliessend wird die Adresse des gefundenen Geräts in einen String¹⁶ umgewandelt. So kann diese verglichen werden. Dafür gibt es eine Funktion.

```
bt_addr_le_to_str(addr, addr_str, sizeof(addr_str));
```

Die Adresse im String Format befindet sich nun in der Variablen «*addr_str*», welche zuvor deklariert wurde. Nun kann die Adresse mit derjenigen des zu verbindenden Geräts verglichen werden. Hier ist es von Vorteil, wenn getestet wird, ob sich die gewünschte Adresse in «*addr_str*» befindet. Denn am Ende von «*addr_str*» befinden sich noch weitere Charaktere.

```
if(strstr(addr_str,addr_toConnect))
```

3. Da sichergestellt wurde, dass es sich um das korrekte Gerät handelt, kann eine Verbindung aufgebaut werden.

```
err = bt_conn_le_create(addr, BT_CONN_LE_CREATE_CONN,
                       BT_LE_CONN_PARAM_DEFAULT, &centralConnections
                       [nbrConnectionsCentral-1]);
```

Der erste Parameter ist die Adresse des zu verbindenden Geräts. Der zweite Parameter beinhaltet Informationen zum Verbindungsaufbau, hier wurden die Standardwerte verwendet. Der dritte Parameter ist der Verbindungsparameter, hier wurden auch die Standardwerte verwendet. Der vierte und letzte Parameter ist die Adresse eines Verbindungsobjektes. Dieses zeigt momentan auf nichts. Wenn die Verbindung aufgebaut ist, wird in dieser Variablen die Adresse der erzeugten Verbindung gespeichert sein. Die Funktion gibt den Wert '0' zurück, wenn die Verbindung erfolgreich aufgebaut wurde.

¹⁶ Zeichenkette

In der Rückrufmethode «connected» muss wie zuvor im Peripherie Teil, getestet werden, ob es sich um eine zentrale oder periphere Verbindung handelt. Dafür muss zuerst wieder die «bt_conn_get_info» Funktion aufgerufen werden. Eine genauere Erklärung befindet sich unter Punkt 4 im Kapitel «BLE: Peripherie». Im Fall, dass ein Fehler bei der Verbindung aufgetreten ist, sprich es befindet sich negativer Error Code im Parameter, wird der Scan Vorgang erneut gestartet und die Attribute wieder auf «NULL» initialisiert.

Jetzt besteht eine Verbindung zwischen dem nRF5340 und der Applikation und einem peripheren Gerät.

Variante mit den CSC (Cycling Speed and Cadence) / Herzfrequenz Sensoren:

Mit Hilfe des Cycling Speed and Cadence Service (CSCS), ein Service für die Geschwindigkeit und die Trittfrequenz beim Radfahren, ist es möglich, Daten von den Sensoren von Garmin via Bluetooth zu erhalten. Hierfür muss die notwendige Charakteristik benutzt werden, nämlich die «*CSC Measurement*», die Messcharakteristik. Es werden zwei Sensoren benötigt, einer für die Geschwindigkeit und einer für die Trittfrequenz.

Sobald dass alle Verbindungen zu den CSC Sensoren aufgebaut wurden, wird nach dem ausgewählten Herzfrequenzsensor gesucht und eine Verbindung zu diesem aufgebaut. Anschliessend wird auch hier eine Charakteristik verwendet, um die Daten zu empfangen, nämlich die «*HRS Measurement*» Charakteristik. In der folgenden Erklärungen wird nur auf den Verbindungsaufbau und das Abonnieren der Services der CSC Sensoren eingegangen. Für den Herzfrequenzsensor können die Erklärungen adaptiert werden.

Im vorderen Schritt wurde erklärt, wie eine zentrale Verbindung des nRF5340 mit einem Thingy:52 aufgebaut wird. Nun wird das Thingy mit den Sensoren von Garmin ersetzt. Die Adressen dieser Sensoren sind nicht statisch. Das bedeutet sie ändern sich, wenn die Batterie ausgetauscht wird. Daher ist es nicht möglich gleich vorzugehen wie bei der Verbindung mit einem Thingy. In diesem Fall wurde das Scanning Modul [18] von Nordic verwendet. Hier kann bereits vor dem Beginn des Scavorgangs, ein Filter gesetzt werden, sodass nur Sensoren mit einer «*CSC UUID*» erkannt werden. Im Folgenden wird erklärt, wie eine Verbindung mittels des Scanning Moduls zu den CSC Sensoren aufgebaut wird.

Die nötigen Informationen wurden vom «*lte_ble_gateway*» Beispiel von Nordic hinzugezogen. Dieses ist auf Github [19] verfügbar.

1. Es werden ein Scan Parameter und ein Scan Initialisierungs-Parameter benötigt. Diese werden mittels Strukturen wie folgt definiert. Der Scan Parameter ist derselbe wie zuvor bei der Verbindung mit einem Thingy.

```

struct bt_le_scan_param scanParam = {
    .type = BT_LE_SCAN_TYPE_ACTIVE,
    .options = BT_LE_SCAN_OPT_FILTER_DUPLICATE,
    .interval = BT_GAP_SCAN_FAST_INTERVAL,
    .window = BT_GAP_SCAN_FAST_WINDOW,
    .timeout = 0
};

struct bt_scan_init_param scanInit = {
    .scan_param = &scanParam,
    .connect_if_match = 0,
    .conn_param = BT_LE_CONN_PARAM_DEFAULT,
};

```

Mit der «*scanInit*» Struktur wird zuerst der «*scanParam*» als Referenz übergeben. Mit «*connect_if_match = 0*» wird erreicht, dass sich der nRF5340 nicht automatisch mit einem Gerät verbindet, wenn eines gefunden wird. Dies kann manuell erreicht werden. Als Verbindungsparameter wurden die Standardeinstellungen gewählt.

Nun wird das Initialisierungsmakro für das Scanning Modul verwendet. Der Prototyp sieht wie folgt aus:

```
BT_SCAN_CB_INIT(_name,match_fun,no_match_fun,error_fun,connecting_fun)
```

«*_name*» ist der Name der Rückruffunktion und kann frei gewählt werden. Nun folgen die Rückruffunktionen. Für den Fall, dass ein Gerät mit dem eingestellten Filter gefunden wurde, (*match_fun*). Wenn ein Gerät gefunden wurde, welches aber nicht auf die Filter zutrifft (*no_match_fun*). Im Fall, dass ein Fehler aufgetreten ist (*error_fun*). Die letzte Rückruffunktion ist die, wenn eine Verbindung aufgebaut wird (*connecting_func*). Es müssen nicht alle Funktion implementiert werden. In diesem Fall wurde die «*match_fun*» und «*error_fun*» implementiert. Das Initialisierungsmakro sieht wie folgt aus:

```
BT_SCAN_CB_INIT(scan_cb,scanFilterMatch,NULL,scanConnectionError,NULL);
```

Nun kann das Scanning Modul initialisiert werden:

```
bt_scan_init(&scanInit);
```

Als Parameter wird eine Referenz zur Initialisierungsstruktur übergeben. Als nächstes müssen die Rückruffunktionen noch eingeschrieben werden.

```
bt_scan_cb_register(&scan_cb);
```

Als nächstes wird der korrekte Filter eingestellt. Die verwendeten Parameter sind bereits in der «*scan.h*» Bibliothek definiert.

```
err = bt_scan_filter_add(BT_SCAN_FILTER_TYPE_UUID, BT_UUID_CSC);
```

Mit diesem Filter werden nur Geräte mit einer «*CSC UUID*» erkannt. Die Funktion gibt den Wert '0' zurück, wenn die Filter erfolgreich gesetzt wurden.

Als letzter Schritt müssen die Filter aktiviert werden. Dies erfolgt durch den Aufruf folgender Funktion:

```
err = bt_scan_filter_enable(BT_SCAN_UUID_FILTER, false);
```

Mit dem zweiten Parameter kann bestimmt werden, dass ein Gerät auf alle Filter passen muss (wenn mehrere definiert wurden). Oder, ob es ausreicht, wenn ein Filter zutrifft. In diesem Fall wird nur ein Filter verwendet, darum wird diese Einstellung nicht benötigt. Die Funktion gibt den Wert '0' zurück, wenn die Filter erfolgreich aktiviert wurden.

Mit dem Aufruf

```
err = bt_scan_start(BT_SCAN_TYPE_SCAN_ACTIVE);
```

wird der Scan Vorgang gestartet. Sobald ein Gerät, auf welches der Filter zutrifft, gefunden wird, wird die Funktion «*scanFilterMatch*» aufgerufen. Die Funktion gibt den Wert '0' zurück, wenn der Scan Vorgang erfolgreich gestartet wurde.

2. Wurde ein passender Sensor gefunden, wird in der «*scanFilterMatch*» Funktion zunächst der Scan Vorgang abgebrochen. Anschliessend wird eine Verbindung zu diesem Sensor aufgebaut. Die Funktion gibt den Wert '0' zurück, wenn die Verbindung erfolgreich aufgebaut wurde. Wenn ein bestimmter CSC Sensor gesucht wird, kann dieser mittels der Adresse verglichen werden. Dies wurde im vorderen Teil, «*Variante mit Thingy*», genauer erklärt.

```
bt_scan_stop();
err = bt_conn_le_create(device_info->recv_info->addr,
                        BT_CONN_LE_CREATE_CONN,
                        device_info->conn_param,
                        &centralConnections[nbrConnectionsCentral]);
```

3. Wenn die Verbindung aufgebaut wurde, wird die «*connected*» Funktion aufgerufen. Hier wird zwischen einem zentralen oder periphereren Verbindungsauftbau unterschieden.

```
bt_conn_info info;
int error = bt_conn_get_info(conn,&info);
if (info.role == BT_CONN_ROLE_MASTER)
```

Zuerst wird die Adresse der Verbindung gespeichert. Anschliessend folgt die Einschreibung des «*CSC Service*». Hierfür werden Rückruffunktionen benötigt. Diese werden in einer Struktur gespeichert und später als Parameter übergeben.

```
static struct bt_gatt_dm_cb discovery_cb_CSC =
{
    .completed = DeviceManager::discoveryCompletedCSC,
    .service_not_found = DeviceManager::discovery_service_not_found,
    .error_found = DeviceManager::discovery_error_found,
};
```

Nun kann die Suche nach dem Service gestartet werden. Als Parameter werden die Adresse der erstellten Verbindung, der zu suchende Service, die Adresse der Rückruffunktionsstruktur, sowie ein Kontext übergeben. Als Kontext kann «*NULL*» übergeben werden. Der Funktionsaufruf sieht wie folgt aus:

```
err = bt_gatt_dm_start(centralConnections[nbrConnectionsCentral-1],
                      BT_UUID_CSC, &discovery_cb, NULL);
```

Die Funktion gibt den Wert '0' zurück, wenn der Suchvorgang erfolgreich gestartet wurde. Wird der Service im verbundenen Gerät gefunden, wird die «*discoveryCompleted*» Funktion aufgerufen. Hier muss die «*CSC Measurement*» Charakteristik eingeschrieben werden. Der Prototyp sieht wie folgt aus.

```
static void discoveryCompletedCSC(struct bt_gatt_dm *dm, void *ctx)
```

4. Als erstes wird eine «*bt_gatt_subscribe_params*» Struktur definiert. Diese gibt an, welche Funktion aufgerufen wird, wenn Daten empfangen werden. Des Weiteren werden hier die Benachrichtigungen aktiviert. Diese Struktur sieht wie folgt aus. Die Rückruffunktion wird in einem späteren Schritt implementiert.

```
static struct bt_gatt_subscribe_params param = {
    .notify = onReceived,
    .value = BT_GATT_CCC_NOTIFY,
};
```

Nun werden zwei «*bt_gatt_dm_attr*» Attribute benötigt.

```
const struct bt_gatt_dm_attr *chrc;
const struct bt_gatt_dm_attr *desc;
```

Das erste Attribut, «*chrc*», wird benötigt, um die Charakteristik mittels der UUID zu bekommen.

```
chrc = bt_gatt_dm_char_by_uuid(dm, BT_UUID_CSC_MEASUREMENT);
```

Der Parameter «*dm*» ist bereits als Parameter in der Rückruffunktion enthalten und wird direkt weitergegeben. Die UUID befindet sich bereits in der «*scan.h*» Bibliothek.

Mit dem zweiten Attribut, «*desc*», wird der Deskriptor mittels der UUID erhalten.

```
desc = bt_gatt_dm_desc_by_uuid(dm, chrc, BT_UUID_CSC_MEASUREMENT);
```

Nun wird eine weitere Einstellung im zuvor definierten «*param*» vollzogen. Der «*value handler*» wird zugewiesen.

```
param.value_handle = desc->handle;
```

Jetzt wird noch ein Deskriptor benötigt. Der sogenannte «*Client Characteristic Configuration Descriptor*» (CCCD). Dies erfolgt ähnlich wie zuvor. Es muss nur die UUID geändert werden.

```
desc = bt_gatt_dm_desc_by_uuid(disc, chrc, BT_UUID_GATT_CCC);
```

Nun muss dieser auch wieder im «*bt_gatt_subscribe_params param*» gespeichert werden. Dieses Mal wird er dem «*ccc handler*» zugewiesen.

```
param.ccc_handle = desc->handle;
```

Mit der folgenden Funktion kann die Charakteristik abonniert werden. Als Parameter werden eine Referenz zur Verbindung, sowie der konfigurierte «*param*» übergeben. Die Funktion gibt den Wert '0' zurück, wenn die Einschreibung erfolgreich war.

```
err = bt_gatt_subscribe(bt_gatt_dm_conn_get(dm), &param);
```

Da nur jeweils eine Erkennungs-Prozedur zur selben Zeit gestartet werden kann, muss nun diese freigegeben werden. Dies geschieht durch folgenden Funktionsaufruf.

```
bt_gatt_dm_data_release(dm);
```

Nun ist es möglich Daten der CSC Sensoren zu erhalten.

Mit derselben Vorgehensweise kann beispielsweise auch die Taster Charakteristik eines Thingy's eingeschrieben werden. Es müssen lediglich die UUIDs geändert werden.

Im folgenden Schritt wird gezeigt, wie die Daten analysiert werden.

5. Die Datenanalyse befindet sich in der Rückruffunktion «*onReceived*», diese sieht wie folgt aus.

```
uint8_t onReceived(struct bt_conn *conn,
                  struct bt_gatt_subscribe_params *params,
                  const void *data, uint16_t length)
```

Als Parameter werden Referenzen auf die Verbindung, die Abonnenten-Parameter und die Daten, sowie die Länge der Daten erhalten. Der erste Wert des Datenzeigers beinhaltet den Typ des Sensors. Dieser gibt an, ob die erhaltenen Daten vom Geschwindigkeitssensor (Wert '0') oder vom Trittfrequenzsensor (Wert '1') stammen. Mit der folgenden Zeile kann der Typ aus den Daten gelesen werden.

```
type = ((uint8_t*)data)[0];
```

Der Wert an der Stelle 2, also «((uint8_t*) data) [1]» gibt die gesamten Umdrehungen, welche zum gesendeten Zeitpunkt gemessen wurden an. An der Stelle 6, also «((uint8_t*) data) [5]» befindet sich der letzte Ereigniszeitwert für den Geschwindigkeitssensor. An der Stelle 4, der letzte Ereigniszeitwert für den Trittfrequenzsensor. Mit diesen Daten können die aktuelle Fahrtgeschwindigkeit und die Trittfrequenz berechnet werden.

Information

«Der Zähler im Sensor, welcher die Umdrehungen zählt, ist vom Typ «*uint32*» und kann somit 4'294'967'296 Umdrehungen zählen. Bei einer Annahme eines Radumfanges von 2,1 Meter (66,8cm Durchmesser), so kann eine Distanz von 9'019'431 Kilometer gefahren werden, bis es einen Überlauf gäbe. Da der Sensor eine durchschnittliche Lebensdauer von 5 Jahren hat und ein Spitzenradfahrer bis zu 15'000 Kilometer im Jahr zurücklegt (75'000km in 5 Jahren), übertrifft dieser Wert die Erwartung deutlich.» [20]

6. Nun folgen Erklärungen zu den Berechnungen. Alle Werte und Berechnungen befinden in der Klasse «*Data*». Wichtig ist, dass die alten Werte gespeichert werden müssen. Um die Berechnungen durchführen zu können, werden die Differenzen benötigt. Die Formeln wurden dem Dokument «*CSC Spezifikation*» [20] auf Seite 14 entnommen. Ausserdem wurde eine vorjährige Diplomarbeit «*IoT bike gateway for Swiss Cycling*» [21] von Rossier Yoan zugezogen. Dieser befindet sich im Anhang Ordner «*Anhänge*»

In der Abbildung 9 und Abbildung 10 sind Diagramme für die Berechnung der Trittfrequenz und der Geschwindigkeit ersichtlich.

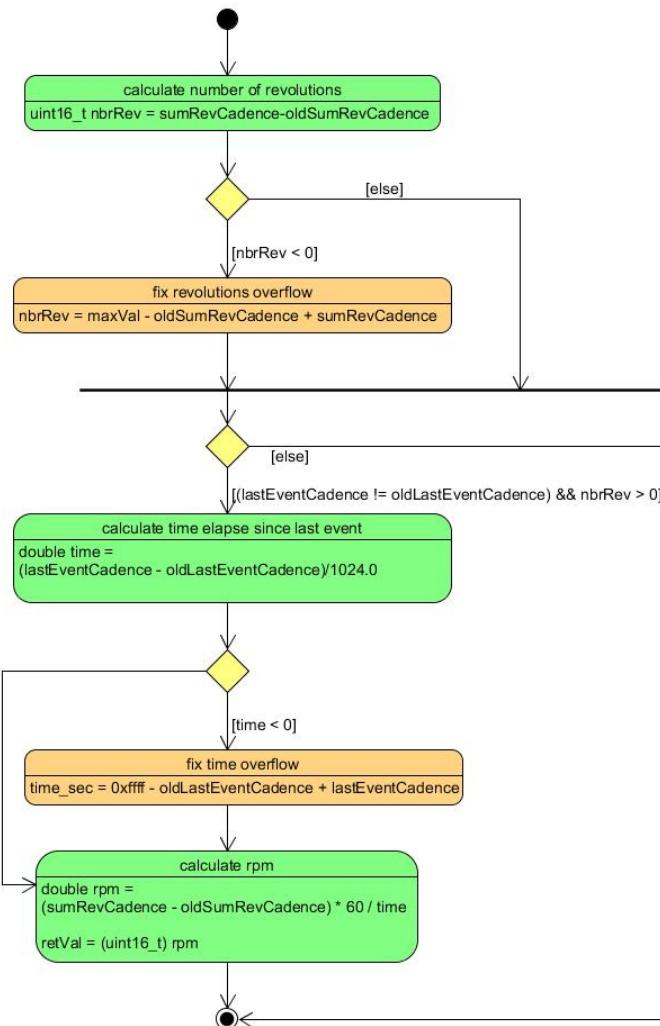
Trittfrequenz:

Abbildung 9: Berechnung RPM

Quelle: Eigene Darstellung 11

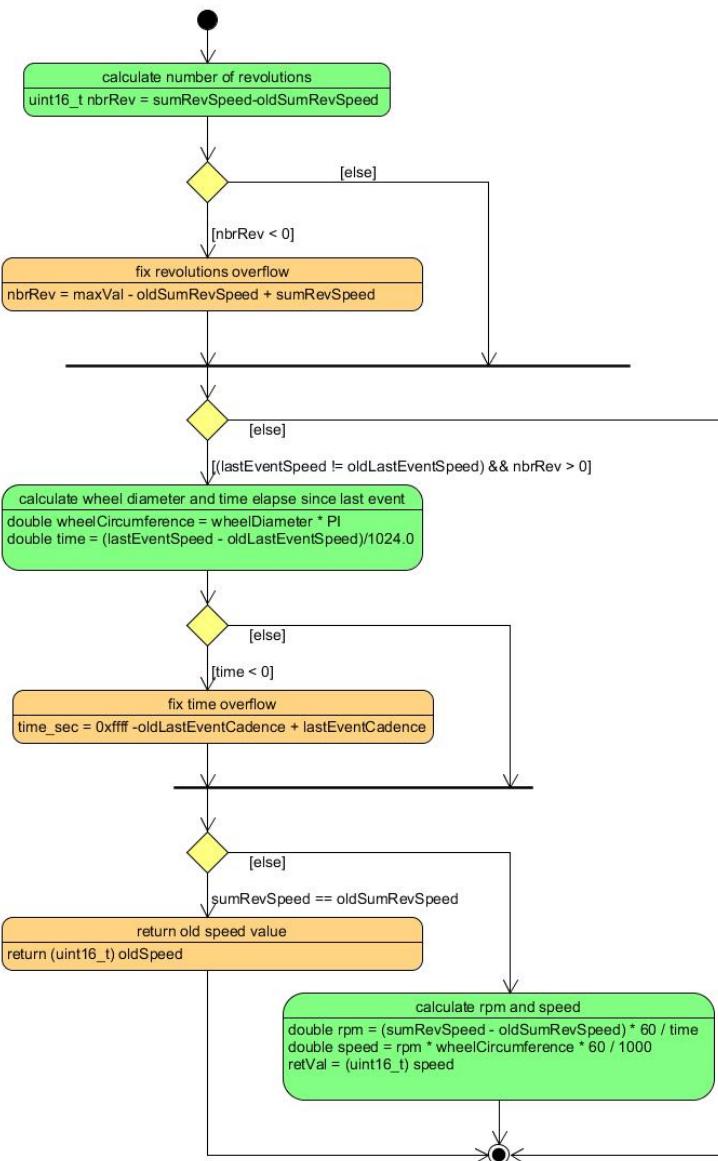
Geschwindigkeit:

Abbildung 10: Berechnung Geschwindigkeit

Quelle: Eigene Darstellung 12

Die Geschwindigkeit wird in der Einheit [km/h x 100] an den «DeviceManager» übergeben. So wird die Geschwindigkeit auf den Hundertstel genau in der Applikation angezeigt.

6.5 Batterie Service Client

In diesem Kapitel wird auf die Implementierung des GATT Batterie Service Client (BAS) von Nordic [10] eingegangen. Es wurde die Klasse «*BatteryManager*» erstellt. Wichtig zu erwähnen ist, dass dies eine .c Klasse ist. Dies ist notwendig, da es ansonsten Probleme bei der Kompilierung gibt. Es wird nur auf die Nutzung des Services für einen Sensor eingegangen. Zu Beginn muss nur folgende Datei eingeschlossen werden.

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

Um den Batterie Stand eines Gerätes zu erhalten, muss als erstes die Funktion «*initBatteryManager*» aufgerufen werden. Hier wird die Initialisierungsfunktion vom «*BAS Client*» aufgerufen. Dafür wird eine «*bt_bas_client*» Struktur benötigt.

```
static struct bt_bas_client bas_speed;
bt_bas_client_init(&bas_speed);
```

Anschliessend kann nach dem Batterie Service gesucht werden. Nicht alle Geräte unterstützen diese Funktion. Dies wird mittels der Funktion «*gatt_discover_battery_service(struct bt_conn *conn)*» erreicht. In dieser Funktion werden zuerst die Rückruffunktionen definiert.

```
static struct bt_gatt_dm_cb discovery_cb = {
    .completed = discovery_completed_cb,
    .service_not_found = discovery_service_not_found_cb,
    .error_found = discovery_error_found_cb,
};
```

Danach kann die Suche gestartet werden. Es wird dieselbe Funktion verwendet wie im Abschnitt «*BLE: Zentral*», wo nach dem CSC Service gesucht wurde. Hier wird die UUID des Batterie Services verwendet. Diese Funktion gibt den Wert '0' zurück, wenn der Discover Vorgang erfolgreich gestartet wurde, ansonsten einen negativen Fehlercode.

```
err = bt_gatt_dm_start(conn, BT_UUID_BAS, &discovery_cb, NULL);
```

Sobald der Service gefunden wurde, wird die Funktion «*discovery_completed_cb(struct bt_gatt_dm *dm, void *context)*» aufgerufen. Im Fall, dass der Service nicht gefunden wurde, oder ein Fehler aufgetreten ist, werden die anderen Rückruffunktionen aufgerufen.

Um das Verbindungsattribut, welches in der vorderen Funktion übergeben wurde, mit dem der Instanz des Modules zu verbinden muss der folgender Aufruf getätigter werden. Dies macht es möglich, mehrere Geräte mit dem Batterie Service zu verknüpfen. Die Funktion gibt den Wert '0' zurück, wenn die Vorgang erfolgreich war, ansonsten einen negativen Fehlercode.

```
err = bt_bas_handles_assign(dm, &bas_speed);
```

Dies war bereits alles. Es ist möglich, immer nach einem gewissen Zeitintervall, eine Benachrichtigung zu erhalten. Dies wird hier nicht verwendet. Es wird immer wieder der aktuelle Batteriestand abgefragt. Da nur jeweils eine Erkennungs-Prozedur zur selben Zeit gestartet werden kann, muss diese freigegeben werden. Dies geschieht durch folgenden Funktionsaufruf.

```
err = bt_gatt_dm_data_release(dm);
```

Mit dem folgenden Aufruf kann der aktuelle Batterie Stand abgefragt werden.

```
bt_bas_read_battery_level(&bas_speed, read_battery_level_cb_speed);
```

In der Rückruffunktion «*read_battery_level_cb_speed*» wird der Batterie Stand des Geschwindigkeitssensors als Parameter erhalten.

Im folgenden Sequenzdiagramm (Abbildung 11) ist dieser Ablauf nochmals erläutert.

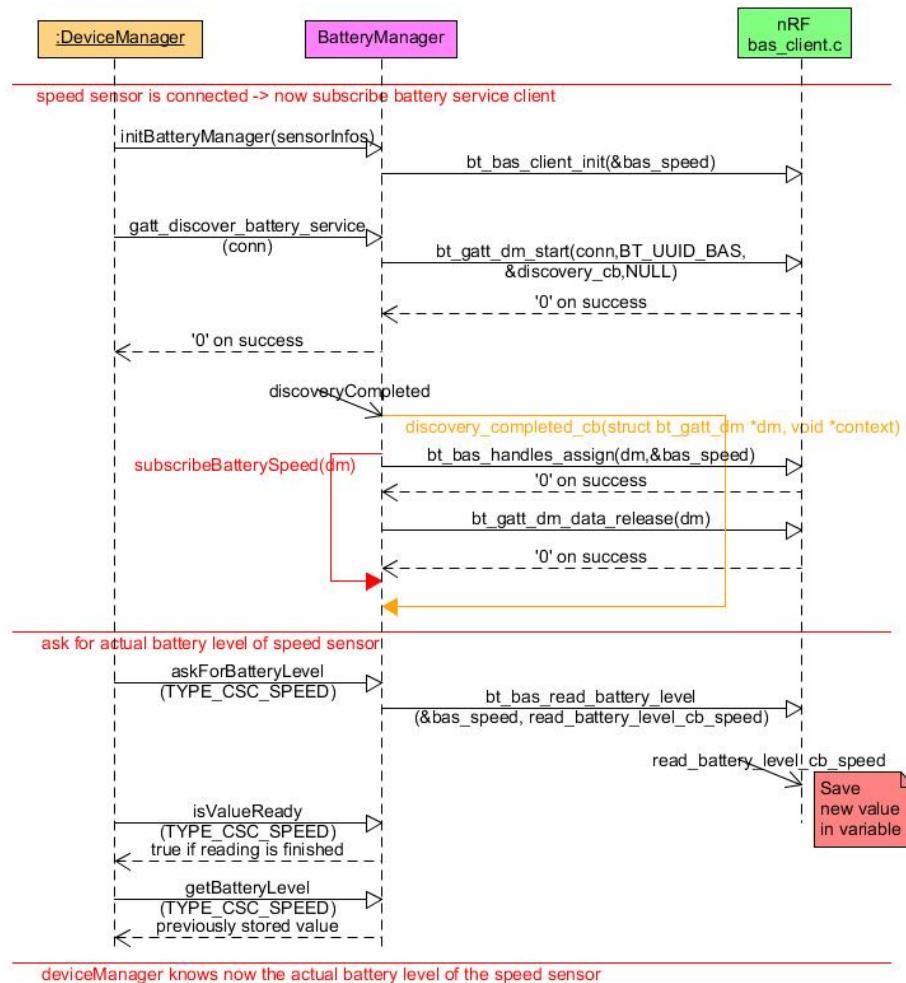


Abbildung 11: Sequenzdiagramm Battery Manager

Quelle: Eigene Darstellung 13

6.6 Data Service (Server)

In diesem Schritt werden die Daten an die Android Applikation gesendet. Um dies zu erreichen, musste ein neuer Service implementiert werden, ein RX-TX Service (Receive-Transmitt). «*Transmitt*», um die berechneten Werte zu schicken und «*receive*», um Daten von der Applikation zu erhalten. Der Nutzer kann in der Applikation den Durchmesser des Fahrrades eingeben. Dieser wird benötigt, um die Geschwindigkeit zu berechnen.

Um den Server Teil (nRF5340) des Services zu implementieren wurde ein Tutorial [22] von Nordic befolgt, welches nun genauer erläutert wird. Der Klient Teil (Android Applikation) wird im Kapitel 7 Android Applikation genauer erläutert.

Als erstes muss eine neue Klasse erstellt werden, in diesem Fall wurde der Name «*DataService*» vergeben. In der «*DataService.h*» Datei muss folgendes «*Include*» gemacht werden.

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

Nun wird eine Initialisierungsfunktion benötigt. Diese muss von der «*DeviceManager*» Klasse aufgerufen werden, bevor Daten gesendet oder empfangen werden können.

```
int data_service_init(void)
{
    int err = 0;

    memset(&data_rx, 0, MAX_TRANSMIT_SIZE);
    memset(&data_tx, 0, MAX_TRANSMIT_SIZE);

    return err;
}
```

Es werden mindestens zwei Puffer benötigt, Einen zum Senden und einen zum Empfangen der Daten.

```
#define MAX_TRANSMIT_SIZE 240
uint8_t data_rx[MAX_TRANSMIT_SIZE];
uint8_t data_tx[MAX_TRANSMIT_SIZE];
```

Für den Service wird eine 128-Bit UUID benötigt. Diese kann auf «*uuidonline*» [23] generiert werden. Die Definition sieht wie folgt aus.

```
#define DATA_SERVICE_UUID 0xd4, 0x86, 0x48, 0x24, 0x54, 0xB3, 0x43, 0xA1, \
0xBC, 0x20, 0x97, 0x8F, 0xC3, 0x76, 0xC2, 0x75
```

Ausserdem werden zwei UUIDs für die TX und RX Charakteristik benötigt. Die Definitionen befinden sich in der «*DataService.h*» Datei.

```
#define RX_CHARACTERISTIC_UUID 0xA6, 0xE8, 0xC4, 0x60, 0x7E, 0xAA, 0x41,
0x6B, \ 0x95, 0xD4, 0x9D, 0xCC, 0x08, 0x4F, 0xCF, 0x6A

#define TX_CHARACTERISTIC_UUID 0xED, 0xAA, 0x20, 0x11, 0x92, 0xE7, 0x43,
0x5A, \ 0xAA, 0xE9, 0x94, 0x43, 0x35, 0x6A, 0xD4, 0xD3
```

Nun müssen die UUIDs deklariert werden, die Deklaration befindet sich in der «*DataService.h*» Datei.

```
#define BT_UUID_DATA_SERVICE      BT_UUID_DECLARE_128(DATA_SERVICE_UUID)
#define BT_UUID_DATA_SERVICE_RX    BT_UUID_DECLARE_128(RX_CHARACTERISTIC_UUID)
#define BT_UUID_DATA_SERVICE_TX    BT_UUID_DECLARE_128(TX_CHARACTERISTIC_UUID)
```

Mit den folgenden Hilfsmakros werden der Service und die Charakteristiken im BLE-Host Stack¹⁷ registriert.

```
BT_GATT_SERVICE_DEFINE(data_service,
    BT_GATT_PRIMARY_SERVICE(BT_UUID_DATA_SERVICE),
    BT_GATT_CHARACTERISTIC(BT_UUID_DATA_SERVICE_RX,
        BT_GATT_CHRC_WRITE | BT_GATT_CHRC_WRITE_WITHOUT_RESP,
        BT_GATT_PERM_READ | BT_GATT_PERM_WRITE,
        NULL, on_receive, NULL),
    BT_GATT_CHARACTERISTIC(BT_UUID_DATA_SERVICE_TX,
        BT_GATT_CHRC_NOTIFY,
        BT_GATT_PERM_READ,
        NULL, NULL, NULL),
    BT_GATT_CCC(on_cccd_changed,
        BT_GATT_PERM_READ | BT_GATT_PERM_WRITE),
);
;
```

Als nächstes werden die Funktionen für das Senden und Empfangen von Daten geschrieben. Zuerst die für das Empfangen. Im Parameter «buf» befinden sich die Daten.

```
static ssize_t on_receive(struct bt_conn *conn,
    const struct bt_gatt_attr *attr,
    const void *buf,
    uint16_t len,
    uint16_t offset,
    uint8_t flags)
{
    const uint8_t * buffer = buf;

    printk("Received data, handle %d, conn %p, data: 0x", attr->handle,
        conn);
    for(uint8_t i = 0; i < len; i++){
        printk("%02X", buffer[i]);
    }
    printk("\n");

    return len;
}
```

Nun folgt die Funktion für das Senden von Daten. Ein 8-Bit Array kann gesendet werden. Außerdem wird kontrolliert, ob die Benachrichtigungen beim Klienten eingeschaltet sind.

¹⁷ Schicht im BLE Modell

```

void data_service_send(struct bt_conn *conn, const uint8_t *data,
                      uint16_t len)
{
    const struct bt_gatt_attr *attr = &data_service.attrs[3];

    struct bt_gatt_notify_params params =
    {
        .uuid     = BT_UUID_DATA_SERVICE_TX,
        .attr     = attr,
        .data     = data,
        .len      = len,
        .func     = on_sent
    };

    if(bt_gatt_is_subscribed(conn, attr, BT_GATT_CCC_NOTIFY))
    {
        if(bt_gatt_notify_cb(conn, &params))
        {
            printk("Error, unable to send notification\n");
        }
    }
    else
    {
        printk("Warning, notification not enabled on the selected attribute\
               \n");
    }
}

```

In der Struktur «*params*» wird als letztes eine Funktion angegeben. Diese wird ausgeführt, sobald die Daten versandt sind und sieht wie folgt aus. Sie dient zur Kontrolle, ob die Daten versandt worden sind.

```

static void on_sent(struct bt_conn *conn, void *user_data)
{
    ARG_UNUSED(user_data);

    const bt_addr_le_t * addr = bt_conn_get_dst(conn);

    printk("Data sent to Address 0x %02X %02X %02X %02X %02X %02X \n",
           addr->a.val[0], addr->a.val[1], addr->a.val[2],
           addr->a.val[3], addr->a.val[4], addr->a.val[5]);
}

```

Als letztes gibt es noch die Funktion «*on_cccd_changed*». Diese wird aufgerufen, wenn das CCCD¹⁸ Register vom Klienten (Hier Android Applikation) geändert wurde.

¹⁸ Client Characteristic Configuration Descriptor (Client-Merkmal Konfigurationsdeskriptor)

```

void on_cccd_changed(const struct bt_gatt_attr *attr, uint16_t value)
{
    ARG_UNUSED(attr);
    switch(value)
    {
        case BT_GATT_CCC_NOTIFY:
            // Start sending stuff!
            printk("Notifications ON\n");
            break;

        case BT_GATT_CCC_INDICATE:
            // Start sending stuff via indications
            printk("Notifications ON with Indications\n");
            break;

        case 0:
            // Stop sending stuff
            printk("Notifications OFF\n");
            break;

        default:
            printk("Error, CCCD has been set to an invalid value\n");
    }
}

```

Nun ist der RX-TX Service auf der Server Seite fertig implementiert.

6.7 Verwendung des Data Service

Der Data Service wird für vier verschiedene Datensätze verwendet.

1. Allgemeine Informationen, welche in der Applikation angezeigt werden sollen. Hierfür wird im zusendenden Array nur eine Stelle verwendet. Es wird eine Nachrichtencode gesendet, mit dem anschliessend in der Applikation weiter gearbeitet wird. Eine Liste mit allen Nachrichtencodes befindet sich im Abschnitt 7.5, «Nachrichtenanzeige».
2. Um die Herzfrequenz zu übermitteln, werden zwei Stellen im zusendenden Array benötigt. Die erste Stelle gibt an, dass es sich bei den Daten um die Herzfrequenz handelt und die zweite Stelle hält den Wert der Herzfrequenz.
3. Die Geschwindigkeit und Trittfrequenz werden mit drei Stellen im zusendenden Array übermittelt. Die erste Stelle gibt an, um welchen Typ (Geschwindigkeit oder Trittfrequenz) es sich handelt. Bei der Geschwindigkeit werden an zweiter Stelle der 8 Bit Wert der Geschwindigkeit, welche vor dem Komma steht gespeichert. An dritter Stelle werden die 8 Bit nach dem Komma gespeichert. Handelt es sich um die Trittfrequenz, werden an zweiter Stelle die 8 niedrigewertigsten Bits der Trittfrequenz gespeichert. An dritter Stelle befinden sich die 8 höchswertigsten Bits.
4. Der Batteriestand wird auch mit drei Stellen im zusendenden Array übermittelt. Die erste Stelle gibt an, dass es sich um einen Batteriewert handelt. An zweiter Stelle befindet sich die Information, um welchen Sensor es sich handelt. An dritter Stelle befindet sich der Batteriestand in Prozent.

6.8 Konfigurationsmöglichkeiten

Der Benutzer hat verschiedene Möglichkeiten beim Verbindungsaufbau zu den Sensoren. Es ist möglich sich nur mit einem Sensor, zwei oder drei Sensoren zu verbinden. Dies müssen aber zwingend CSC oder Herzfrequenz Sensoren sein, da ansonsten der Service für den Datenaustausch von den Sensoren zum nRF5340 nicht gefunden wird.

Wird die Verbindung zwischen einem Sensor und dem nRF5340 unterbrochen, beginnt der nRF5340 direkt wieder mit dem Scan Vorgang und versucht erneut eine Verbindung zu diesem Sensor aufzubauen. Wird die Verbindung zwischen dem nRF5340 und der Applikation unterbrochen, beginnt der nRF5340 direkt wieder mit dem Werben und wird so in der Applikation wieder erkannt. Es kann wieder eine Verbindung aufgebaut werden.

7 Android Applikation

Die Android Applikation wurde in Android Studio [24] in der Programmiersprache Java entwickelt. Als Vorlage wurde eine bereits existierende Open Source Applikation von Nordic verwendet, die «*Android-nrf-Blinky*» Applikation. Diese Vorlage ist auf Github [25] verfügbar. Es wurde diese gewählt, da hier der ganze Bluetooth Teil, mit Scan Vorgang und Verbindungsvorgang bereits zur Verfügung stand. Es wurden die unnötigen Dienste für die Leds und Taster entfernt, und der neue Service (Data Service) implementiert. Bei dieser Implementation wurden die bereits existierenden Services von Nordic adaptiert.

Die Applikation hat verschiedene Seiten, die sogenannten «activities». Es gibt deren sieben, sie befinden sich im Ordner «res -> layout». Hier wird das Aussehen der verschiedenen Seiten der Applikation gestaltet. Beim Start der Applikation wird die «activity_splash_screen» (Abbildung 12) Seite aktiv. Dies wurde in der «AndroidManifest.xml» Datei festgelegt. Nach einer Sekunde wird die nächste Seite, «activity_scanner» (Abbildung 13), aktiv. Hier werden alle gefundenen Geräte aufgelistet. Oben rechts in der Ecke können zwei Filter eingestellt werden. Der erste ist der UUID Filter. Es werden nur Geräte, welche den «Data Service» benutzen, angezeigt, das wäre in diesem Fall das nRF5340 Board. Mit dem zweiten Filter werden nur Geräte, welche sich in der Nähe befinden angezeigt. Nun können das Board und die zu verbindenden Sensoren ausgewählt werden. Mit einem Klick auf den Taster «Connect» wird eine Verbindung zum nRF5340 aufgebaut. Dies ist über eine Ladeseite ersichtlich. Auf dem Bildschirm wird zuerst «Connecting» und anschliessend «Initializing» angezeigt. Danach wird die «activity_csc» (Abbildung 14) Seite aktiv. Hier sind die Geschwindigkeit, Trittfrequenz, die gefahrene Distanz und die Herzfrequenz ersichtlich. Bevor jedoch die Echtzeitdaten des Geschwindigkeitssensors angezeigt werden können, muss ein Raddurchmesser eingegeben werden. Dieser kann bei Bedarf zurückgesetzt werden. Die Echtzeitdaten des Trittfrequenzsensors, sowie die des Herzfrequenzsensors werden direkt in der Applikation angezeigt. Um eine Fehlfunktion aufgrund eines geringes Batteriestandes der Sensor vorzubeugen, wird der aktuelle Batteriestand in Prozent angezeigt.



Abbildung 12: Splash Screen Activity

Quelle: Eigene Darstellung 14



Abbildung 13: Scanner Activity

Quelle: Eigene Darstellung 15



Abbildung 14: CSC Activity

Quelle: Eigene Darstellung 16

Die restlichen vier Activities befinden sich im Anhang 4. Eine Anleitung zur Verwendung der Applikation befindet sich im Anhang 5.

7.1 Voreinstellungen

Es gibt einige Einstellungen, welche zu Beginn vorgenommen werden müssen. Alle diese Einstellungen befinden sich in der «AndroidManifest.xml» Datei. In der folgenden Grafik ist diese Datei zu sehen. Am rechten Rand befinden sich Nummern, diese dienen zu Erklärungszwecken.

Zunächst müssen einige Genehmigungen festgelegt werden. Die Applikation verwendet Bluetooth und ab dem Android Betriebssystem Marshmallow (Android 6.0) muss auch der Standort freigegeben werden. Dies wird über die Einstellungen im Punkt 1 erreicht. Im Punkt 2 wird das Logo der Applikation definiert. Mittels den Punkten 3, 4 und 5 werden die drei Hauptseiten (Activity) festgelegt. Ein weiteres Logo wird im Punkt 4 eingefügt. Dieses wird in der «activity_scanner» Seite verwendet. Im Punkt 6 wird die Datei festgelegt, in welcher der Service später implementiert wird.

```
<manifest package="no.nordicsemi.android.csc"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="no.nordicsemi.android.LOG"/>

    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="true"/>

    <application
        android:name="no.nordicsemi.android.csc.CSCApplication"
        android:allowBackup="true"
        android:fullBackupContent="true"
        android:icon="@drawable/bicycle2" ← 2
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        tools:ignore="GoogleAppIndexingWarning">
        <activity
            android:name="no.nordicsemi.android.csc.SplashScreenActivity"
            android:theme="@style/AppTheme.SplashScreen"
            android:noHistory="true"
            android:launchMode="singleTop">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name="no.nordicsemi.android.csc.ScannerActivity"
            android:icon="@drawable/bicycle_side"
            android:label="@string/feature_name"
            android:launchMode="singleTop">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category
                    android:name="no.nordicsemi.android.nrftoolbox.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name="no.nordicsemi.android.csc.CSCActivity"
            android:launchMode="singleTop"

            android:parentActivityName="no.nordicsemi.android.csc.ScannerActivity">
        </activity>
        <service android:name="no.nordicsemi.android.csc.profile.CSCManager"
            tools:ignore="Instantiatable,MissingClass" />
    </application>

</manifest>
```



7.2 Scan – und Verbindungs vorgang

In diesem Kapitel wird auf den Scancvorgang mit dem anschliessenden Verbindungs aufbau genauer eingegangen. Es werden nur die notwendigen Schritte erklärt.

Sobald die «ScannerActivity» aktiv ist, wird der Scan Vorgang gestartet. Im folgenden Sequenzdiagramm (Abbildung 15) ist der Ablauf ersichtlich.

Ein Klassendiagramm mit den wichtigsten Teilen befindet sich im Anhang 6.

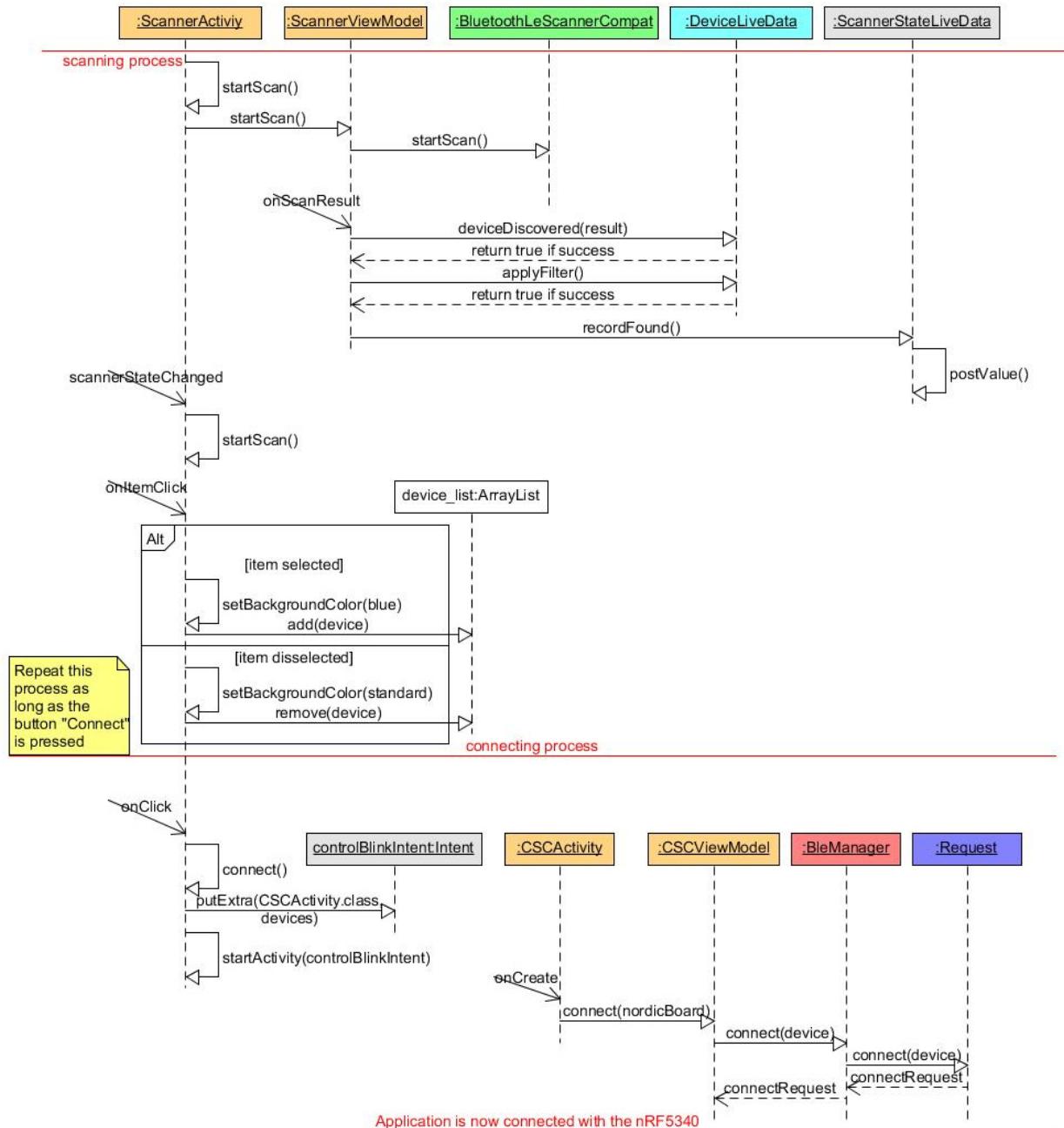


Abbildung 15: Sequenzdiagramm Scan- und Verbindungs vorgang

Quelle: Eigene Darstellung 17

7.3 Data Service (Klient)

In der «*CSCManager*» Klasse wurden zunächst dieselben Charakteristik UUIDs definiert wie zuvor im Kapitel «Data Service (Server)». Dies sieht wie folgt aus.

```
public final static UUID CSC_SERVICE =
UUID.fromString("75c276c3-8f97-20bc-a143-b354244886d4");
public final static UUID RX_CHARACTERISTIC_UUID =
UUID.fromString("6ACF4F08-CC9D-D495-6B41-AA7E60C4E8A6");
public final static UUID TX_CHARACTERISTIC_UUID =
UUID.fromString("D3D46A35-4394-E9AA-5A43-E7921120AAED");
```

Im Unterordner «*Profil->callback*» wurden die Klassen «*RXCallback*», «*RXDataCallback*», «*TXCallback*» und «*TXDataCallback*» hinzugefügt. Die Klassen «*RXCallback*» und «*TXCallback*» sind Interfaces, welche die Methode «*onCSCDataChanged*» zur Verfügung stellen. Diese wurden in der Klasse «*CSCManager*» implementiert. Werden neue Daten erhalten, wird zuerst die «*onDataReceived*» Methode der dementsprechenden Klasse aufgerufen. Hier werden anschliessend die Rückrufmethoden aufgerufen.

Im «*CSCManager*» wurden alle notwendigen Einstellungen hinzugefügt, sodass der Service einwandfrei funktionieren kann. Diese befinden sich in den Funktionen «*initialize*» und «*isRequiredServiceSupported*» und sehen wie folgt aus.

```
protected void initialize() {
    setNotificationCallback(RX_characteristic).with(rxCallback);
    readCharacteristic(RX_characteristic).with(rxCallback).enqueue();
    readCharacteristic(TX_characteristic).with(txCallback).enqueue();
    enableNotifications(RX_characteristic).enqueue();
    setNotificationCallback(TX_characteristic).with(txCallback);
    enableNotifications(TX_characteristic).enqueue();
}
```

Hier wird eingestellt, ob eine Benachrichtigung erhalten wird, wenn sich ein Wert einer Charakteristik ändert.

```
public boolean isRequiredServiceSupported(@NonNull final BluetoothGatt gatt)
{
    final BluetoothGattService CSCService =
        gatt.getService(CSC_SERVICE);

    if (CSCService != null) {
        mBluetoothGatt = gatt;
        RX_characteristic =
            CSCService.getCharacteristic(RX_CHARACTERISTIC_UUID);
        TX_characteristic =
            CSCService.getCharacteristic(TX_CHARACTERISTIC_UUID);
    }

    boolean writeRequest = false;
    if (TX_characteristic != null) {
        final int rxProp = TX_characteristic.getProperties();
        writeRequest = (rxProp & PROPERTY_NOTIFY) > 0;
    }

    supported = RX_characteristic != null && TX_characteristic != null
        && writeRequest;
    return supported;
}
```

Hier wird der Service mit seinen Charakteristiken kontrolliert, ob diese unterstützt werden.

Wenn nun Daten empfangen werden, müssen diese ausgelesen werden. Dies geschieht in der «onCSCDataChanged» Funktion des TX-Callbacks.

```

switch (data[0]) {
    case TYPE_SPEED:
        double val = data[2].doubleValue();
        speed = data[1] + val / 100;
        speedValue.setValue(speed);
        break;
    case TYPE_CADENCE:
        rpmValue.setValue(data[1] + (data[2] << 8));
        break;
    case TYPE_HEARTRATE:
        heartRateValue.setValue(data[1]);
    case TYPE_BATTERY:
        switch (data[1]) {
            case 1:
                batteryLevelSpeed.setValue(data[2]);
                break;
            case 2:
                batteryLevelCadence.setValue(data[2]);
                break;
            case 3:
                batteryLevelHeartRate.setValue(data[2]);
                break;
            default:
                break;
        }
    default:
        log(Log.INFO, "Unknown type");
        break;
}

```

Die Weiterverarbeitung der Daten wird im nächsten Kapitel, Senden / Empfangen von Daten, behandelt.

7.4 Senden / Empfangen von Daten

Es gibt drei Klassen die für das Senden, Empfangen und Anzeigen von Daten zuständig sind. Die Klasse «*CSCActivity*» ist für die Darstellung der Daten zuständig. Der «*CSCManager*» ist die Klasse, welche die Daten erhält, bzw. abschickt. Die «*CSCModelView*» Klasse dient als Zwischenklasse. Ein einfaches Szenario soll dies erklären. Sobald, dass alle Verbindungen aufgebaut sind, muss der Raddurchmesser angegeben werden. Sobald die Taste zum Bestätigen gedrückt wurde, wird die Methode «*sendDiameter*» des «*CSCViewModels*» aufgerufen. Diese wiederum ruft die Methode «*sendDiameter*» des «*CSCManagers*» auf. Hier werden schlussendlich die Daten mit dem Aufruf «*writeCharacteristic*» gesendet. Diese Richtung funktioniert gut, also *CSCActivity* \Rightarrow *CSCViewModel* \Rightarrow *CSCManager*. Beim Empfangen von Daten kann dieser Vorgang nicht andersrum angewendet werden. In der «*CSCActivity*» wird ein «*<>LiveData> Double*» vom «*CSCManager*» observiert. Sobald neue Daten in dieser Variable gespeichert werden, wird die «*CSCActivity*» benachrichtigt und kann so die neuen Daten in der Applikation anzeigen. Dieses Szenario wird im nachfolgenden Sequenzdiagramm (Abbildung 16) nochmals genauer aufgezeigt.

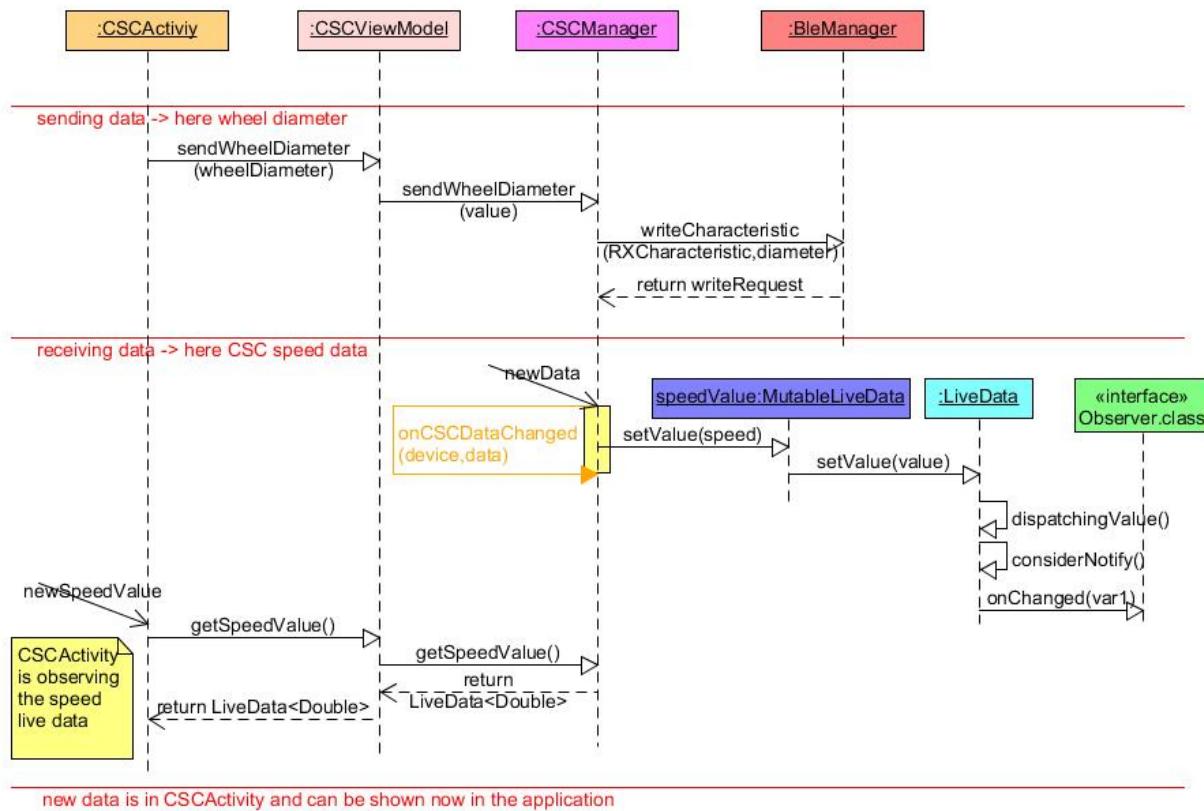


Abbildung 16: Sequenzdiagramm Senden / Empfangen von Daten

Quelle: Eigene Darstellung 18

7.5 Nachrichtenanzeige

Während der Ausführung der Applikation werden immer wieder Kurzmitteilungen am unteren Rand der Applikation angezeigt. Sogenannte «Toasts». Die Information welche Nachricht angezeigt werden soll, wird vom nRF5340 zur Applikation mittels des Data Services gesendet. In der folgenden Tabelle sind alle Nachrichtencodes mit ihrer Bedeutung aufgelistet.

N°	Bedeutung
10	Der Service wurde nicht gefunden
11	Verbindungsabbruch zum Geschwindigkeitssensor
12	Verbindungsabbruch zum Trittfrequenzsensor
13	Verbindungsabbruch zum Herzfrequenzsensor
14	Ein Geschwindigkeitssensor wurde ausgewählt und die Verbindung zu diesem wurde aufgebaut
15	Ein Trittfrequenzsensor wurde ausgewählt und die Verbindung zu diesem wurde aufgebaut
16	Ein Herzfrequenzsensor wurde ausgewählt und die Verbindung zu diesem wurde aufgebaut
17	Zum ersten Sensor (Geschwindigkeitssensor) von mehreren wurde eine Verbindung aufgebaut
18	Es wurden ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt und zum Ersten (Trittfrequenzsensor) wurde eine Verbindung aufgebaut
19	Es wurden ein Trittfrequenz- und ein Geschwindigkeitssensor ausgewählt und beide Verbindungen wurden aufgebaut (Aufforderung zur Eingabe eines Raddurchmessers)
20	Es wurden ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt und beide Verbindungen wurden aufgebaut (Keine Aufforderung zur Eingabe eines Raddurchmessers)
21	Es wurden drei Sensoren ausgewählt und zum Zweiten (Trittfrequenzsensor) wurde eine Verbindung aufgebaut
22	Es wurden ein Geschwindigkeits- und ein Herzfrequenzsensor ausgewählt und beide Verbindungen wurden aufgebaut (Aufforderung zur Eingabe eines Raddurchmessers)

23	Es wurden drei Sensoren ausgewählt und alle Verbindungen wurden aufgebaut. (Aufforderung zur Eingabe eines Raddurchmessers)
24	Die Verbindung zu einem Herzfrequenzsensor wurde wieder neu aufgebaut
25	Der Durchmesserwert muss eine Zahl sein
26	Der Durchmesserwert ist zu gross (Grösser als 255 Inch, 647.7cm)
27	Der Durchmesserwert ist zu klein (Kleiner als 1 Inch, 2.54cm)

Tabelle 2: Nachrichtencodes

7.6 Referenzen Applikation

Applikationslogo:

<https://www.flaticon.com/download/icon/landing/92649?format=png&size=512>

activity_splash_screen:

HES-SO Logo:

https://media-exp3.licdn.com/dms/image/C560BAQGeCvxg5z6RVQ/company-logo_200_200/0/1548143775894?e=2159024400&v=beta&t=dONOI4_oqB9vbGZZoLo8tRLMrxLhPil7GcczDsFmSK0

Valais Excellence Logo:

https://www.valais-excellence.ch/media/image/0/label_valaisexcellence_cmjn.jpg

activity_scanner:

Item Schaltung:

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQonFqstVJNhFO_Auor99-8YZ-fsB-1BLCGIg&usqp=CAU

Fahrrad von der Seite:

<https://cdn5.vectorstock.com/i/1000x1000/97/59/bicycle-rider-icon-vector-21679759.jpg>

activity_csc:

Symbol Geschwindigkeit, Trittfrequenz und Distanz:

Android Studio Bibliothek

8 Datenfluss

In diesem Kapitel wird der gesamte Datenfluss vom Start der Applikation bis zum Datenaustausch mit den Sensoren aufgezeigt. In der Abbildung 17 ist ein generelles Sequenzdiagramm zu sehen, welches den Datenfluss darstellt.

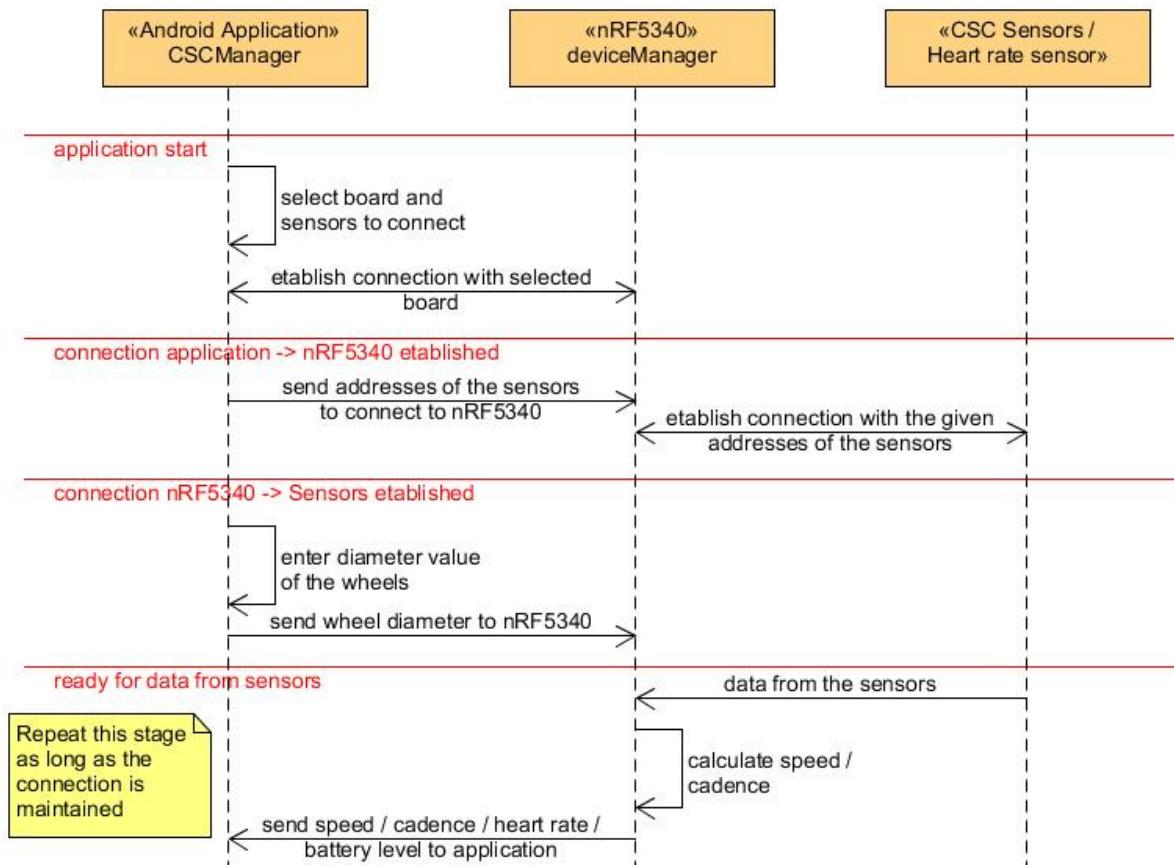


Abbildung 17: Sequenzdiagramm Datenfluss

Quelle: Eigene Darstellung 19

9 Tests

Um die zentrale und periphere Verbindung zu testen wurde zu Beginn als peripheres Gerät ein Nordic Thingy:52 [16] verwendet. Da noch keine Applikation programmiert war, wurde die bereits bestehende Applikation von Nordic Semiconductors, «*nRF Connect for Mobile*» verwendet. Die Applikation ist kostenlos im Google Play Store [15] oder im App Store [26] erhältlich.

Der Datenfluss wird mittels eines Tastendrucks auf dem Thingy:52 symbolisiert. In der Abbildung 18 ist die nRF Connect Applikation zu sehen, nachdem beide Verbindungen aufgebaut und die Taster Charakteristik eingeschrieben wurde. Wenn der Taster auf dem Thingy:52 betätigt wird, ist dies im roten Kasten durch einen Wechsel von «*Button released*» zu «*Button pressed*» ersichtlich. Dies zeigt, dass die Gateway Funktion, Thingy \Rightarrow nRF5340 \Rightarrow Applikation, funktioniert.

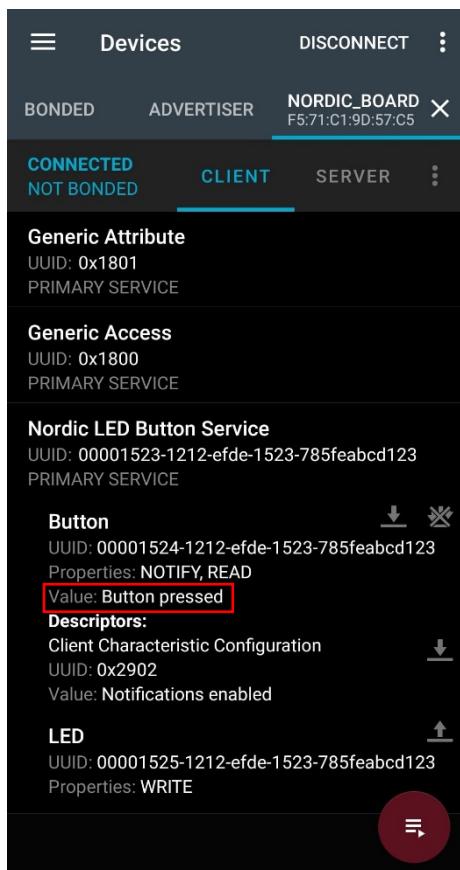


Abbildung 18: Test mit Thingy und der “nRF Connect” Applikation

Quelle: Eigene Darstellung 20

Eine komplette Liste mit allen Tests für die Applikation mit dem nRF5340 befindet sich im Anhang 7.

10 Schlussfolgerung

Während dem Projekt wurde ein BT5-Demonstrator mit einem nRF5340 Entwicklungs-Kit und Zephyr entwickelt.

Dabei wurden folgende Ziele erreicht:

- ✓ Applikation «*HES-SO BIKE*» wurde erstellt
- ✓ Verbindungsaufbau: Applikation \Rightarrow nRF5340
- ✓ Verbindungsaufbau: nRF5340 \Rightarrow CSC Sensoren (max. 2)
- ✓ Verbindungsaufbau: nRF5340 \Rightarrow Herzfrequenz Sensor (max. 1)
- ✓ Verbindungsaufbau: nRF5340 \Rightarrow CSC Sensoren (max. 2) und einem Herzfrequenz Sensor
- ✓ Datenaustausch Applikation \Rightarrow nRF5340
- ✓ Eingabe / Rücksetzung des Raddurchmessers in der Applikation
- ✓ Datenaustausch Sensoren \Rightarrow nRF5340
- ✓ Berechnung und Weiterleitung der Daten an die Applikation
- ✓ Anzeige aller Echtzeitdaten in der Applikation
- ✓ Anzeige des Batteriestandes aller verbundenen Sensoren
- ✓ Anzeige von Kurznachrichten in der Applikation
- ✓ Verbindungsabbruch und Wiederherstellung der Verbindung zu Sensoren
- ✓ Verbindungsabbruch und Wiederherstellung der Verbindung zur Applikation

Folgende Fehler treten auf:

- ✗ Wird auf der Scan Seite ein Gerät ausgewählt erscheint das betätigte Feld in der Farbe hellblau. Wird nun dieses Feld durch scrollen aus dem Sichtbereich befördert, wird wieder die Anfangsfarbe angezeigt.
- ✗ In vereinzelten Fällen wird nach dem Verbindungsaufbau zu einem Sensor der Service für den Datenaustausch (Daten vom Sensor zum nRF5340) nicht gefunden. In diesem Fall wird die Verbindung getrennt und erneut aufgebaut.
- ✗ In seltenen Fällen gibt es nach dem Verbindungsaufbau einen «*Timeout*» Fehler, was zu einer Trennung der Verbindung führt. In diesem Fall wird nach dem getrennten Sensor gesucht und erneut eine Verbindung aufgebaut.
- ✗ Wenn die Beiden zuvor erwähnten Fehler öfters hintereinander auftreten, hängt sich der nRF5340 auf und das Board muss neu gestartet werden. In der Applikation muss nur auf die Scan Seite zurücknavigiert werden.

Die ersten drei Fehlerfälle haben keinen Einfluss auf die fehlerfreie Anwendung des Projektes. Nur der letzte Fall hat einen Neustart des nRF5340 zur Folge. Aus den Tests im Anhang 7 geht hervor, dass der letzte Fehlerfall nur in 3 von 50 Fällen bei zwei oder drei verbundenen Sensoren aufgetreten ist. Daher kann behauptet werden, dass die Arbeit in den meisten Fällen funktioniert.

Aussicht

Die Gateway Funktion des nRF5340 mit Zephyr wird in Zukunft in weiteren Projekten wiederverwendet. Ausserdem wird die Bluetooth Programmierung im Unterricht von Herrn Rieder integriert.

Es ist möglich eine automatische Gangschaltung für ein Fahrrad herzustellen. In der Applikation kann eine Soll-Trittfrequenz eingeben werden. Diese wird anschliessend mit der gemessenen Trittfrequenz verglichen und je nach dem wird ein Gang hinauf, hinunter oder gar nicht geschalten. Diese Anwendung ist für Profisportler interessant.

11 Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.



Datum



Bastian Schwery

12 Anhang

Anhang 1 – Zeitplan.....	41
Anhang 2 – Klassendiagramm nRF5340	42
Anhang 3 – Sequenzdiagramm nRF5340.....	43
Anhang 4 – Activitys Applikation.....	44
Anhang 5 – Anleitung Applikation.....	44
Anhang 6 – Klassendiagramm Applikation.....	46
Anhang 7 – Tests Applikation, nRF5340	47

Im Abgabeordner befinden sich:

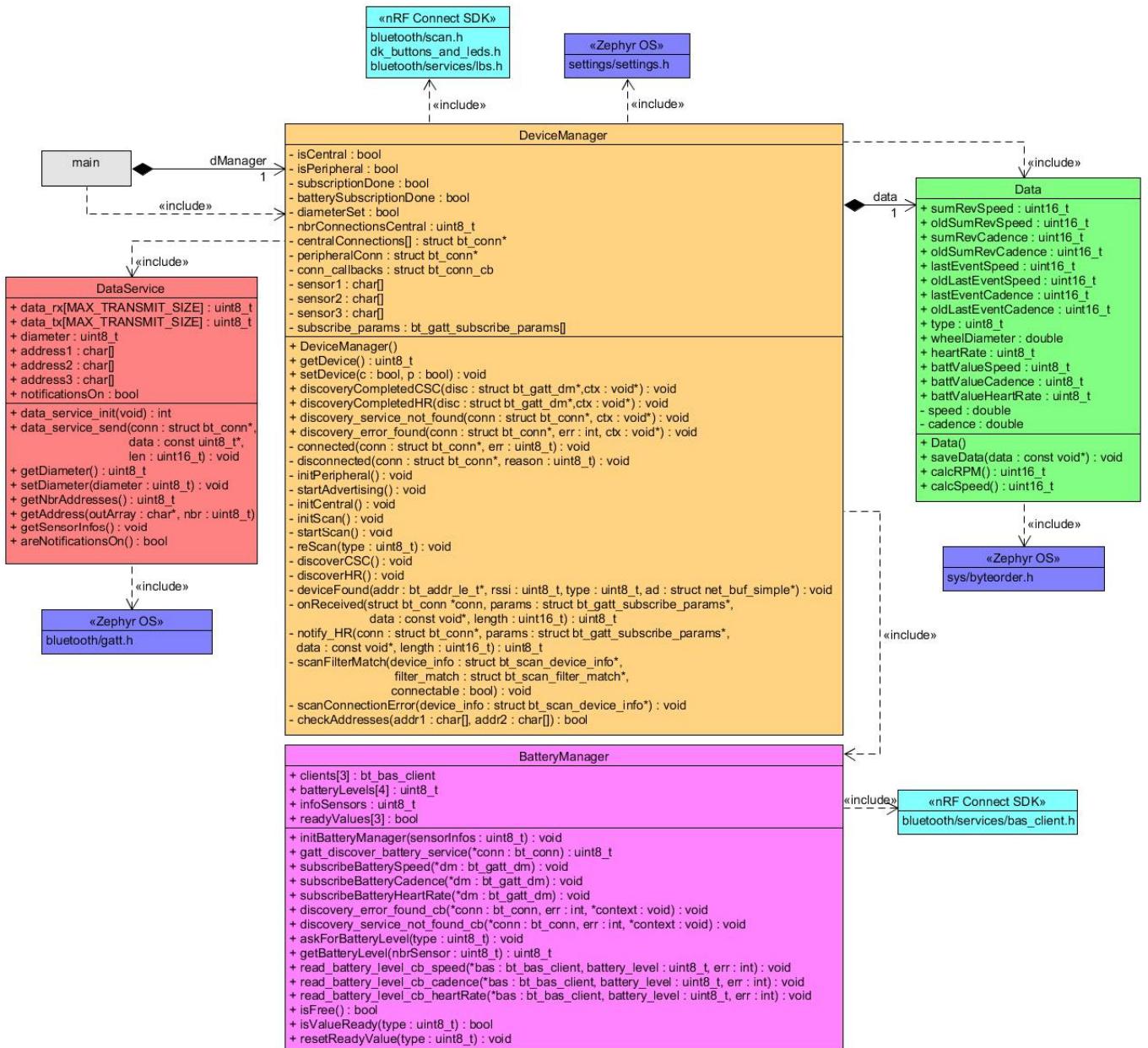
- Bericht
- Code Applikation
- Code nRF5340
- Datenblätter / Dokumente

Der gesamte Code und der Bericht sind ausserdem auf der GitHub Ablage (<https://github.com/bastianschwery/bachelor-thesis>) des Verfassers auffindbar.

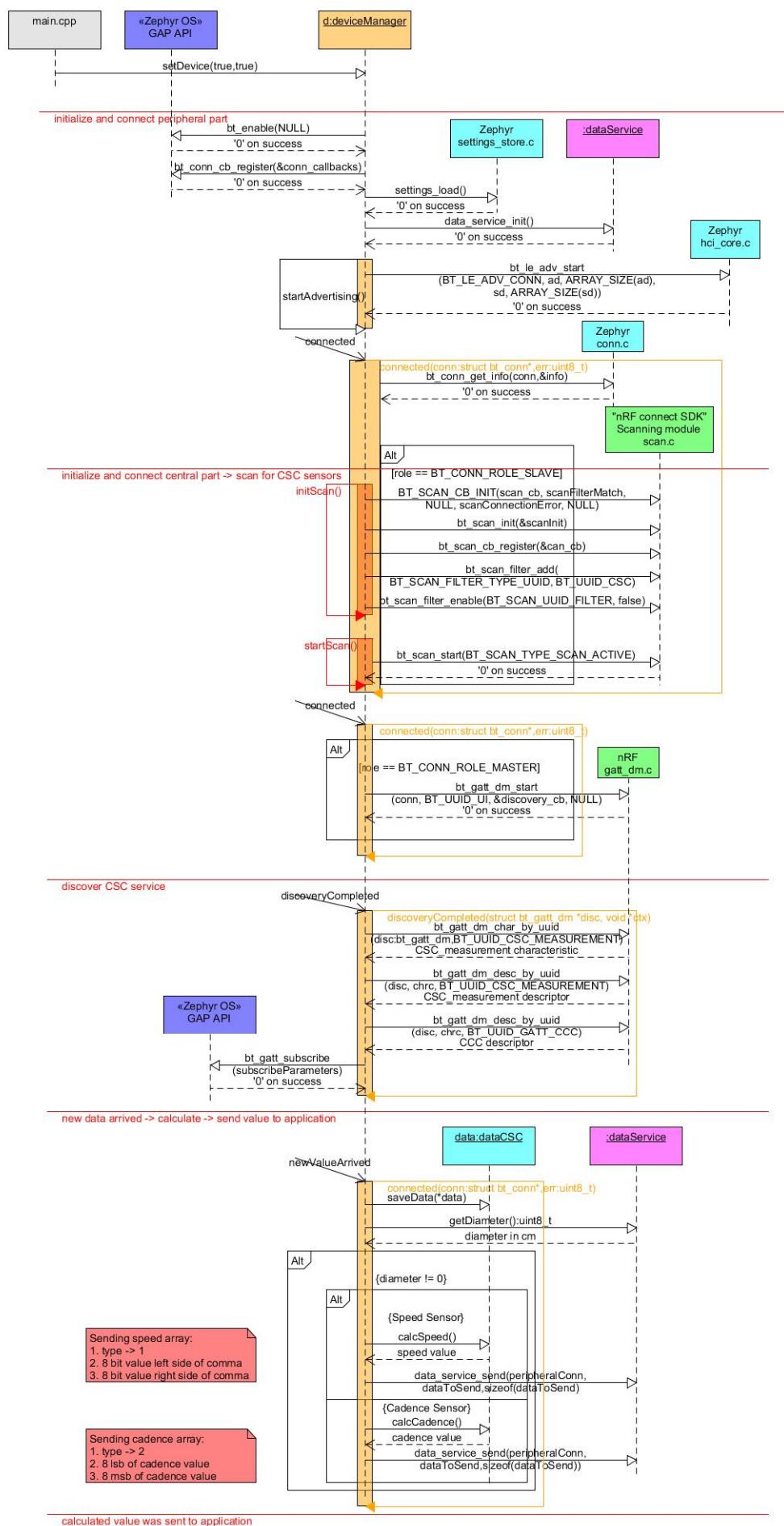
Anhang 1 – Zeitplan

Tätigkeit	Zeitplan																	
	SW 1	SW 2	SW 3	SW 4	SW 5	SW 6	SW 7	SW 8	SW 9	SW 10	SW 11	SW 12	SW 13	SW 14	SW 15	SW 16	SW 17	
Projektinformationen sammeln	Soil	10.05.2021	17.05.2021	24.05.2021	31.05.2021	07.06.2021	14.06.2021	21.06.2021	28.06.2021	05.07.2021	12.07.2021	19.07.2021	26.07.2021	02.08.2021	09.08.2021	16.08.2021	23.08.2021	30.08.2021
Clion Konfiguration	Soil	1st																
Visual Studio Code Konfiguration	Soil		1st															
Programmierung Peripheral	Soil			1st														
Programmierung Zentral	Soil				1st													
Kombination Peripheral & Zentral	Soil					1st												
Taster Charakteristik von Thingy einschreiben	Soil						1st											
CSC Service (Server Seite) implementieren & Charakteristik einschreiben	Soil							1st										
Android Applikation schreiben und testen	Soil								1st									
CSC Service (Klient Seite) implementieren	Soil									1st								
Datenfluss von CSC Sensor zu Applikation	Soil										1st							
Batterie Service implementieren	Soil											1st						
Herzfrequenz Service implementieren	Soil												1st					
Verbesserungen BLE / Applikation	Soil													1st				
Abschliessende Tests	Soil														1st			
Bericht	Soil															1st		
Präsentation	Soil																	1st
Mündliche Verteidigung 31.08.2021 13.30 Uhr																		
Ausstellung der Diplomarbeiten 25.08.2021 - 27.08.2021																		
Abgabe Bericht 20.08.2021 12:00																		

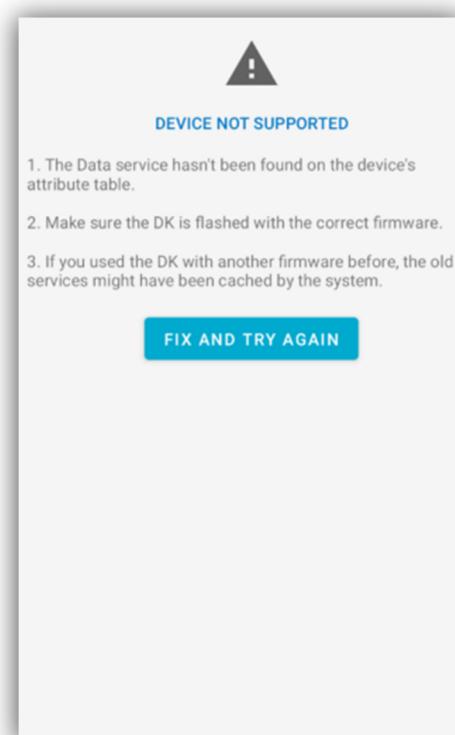
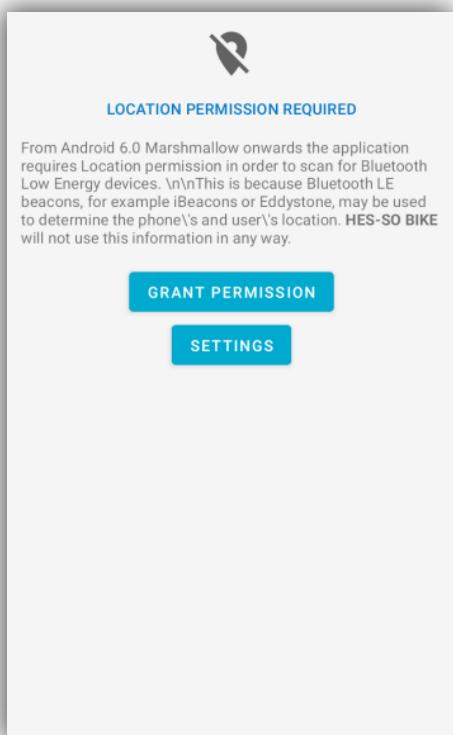
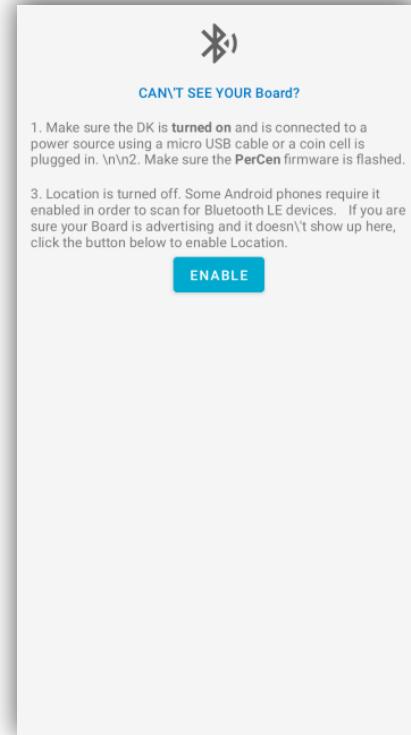
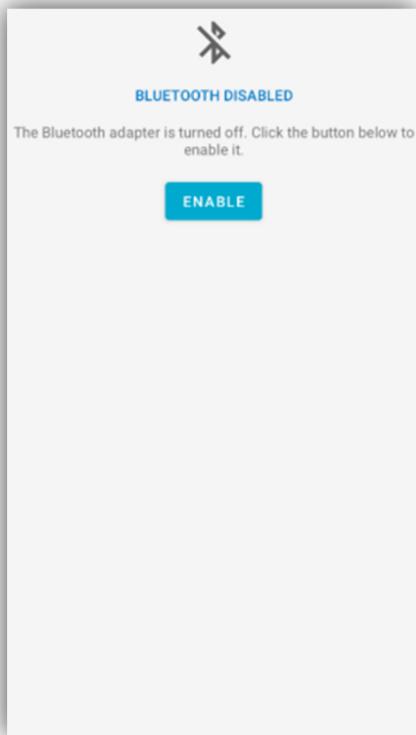
Anhang 2 – Klassendiagramm nRF5340



Anhang 3 – Sequenzdiagramm nRF5340



Anhang 4 – Activitys Applikation



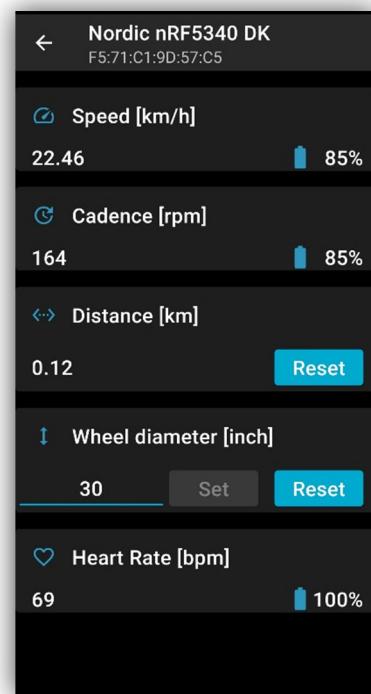
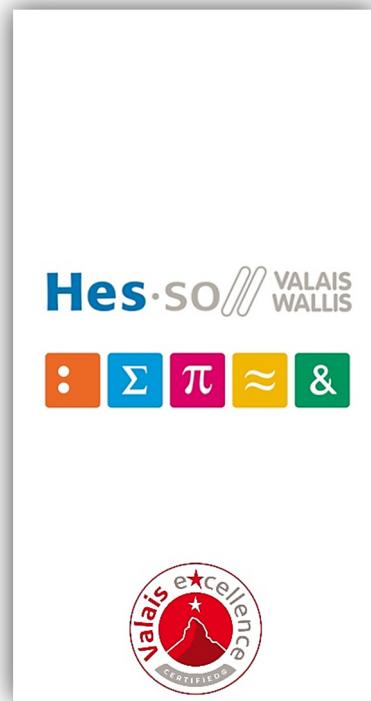
Anhang 5 – Anleitung Applikation

1. Beim Start der Applikation «HES-SO BIKE» wird eine Startseite mit dem HES-SO und dem Valais Excellence Logo angezeigt. Ist das Bluetooth auf dem verwendeten Smartphone ausgeschaltet, erscheint die Aufforderung dieses einzuschalten. Daselbe Prozedere mit dem Standortdienst.
2. Die Scanseite öffnet sich. Oben rechts können zwei Filter gesetzt werden. Mit dem ersten werden nur Geräte angezeigt, welche den implementierten «Data Service» benutzen. Der zweite dient zur Eingrenzung aller Geräte. Es werden nur die Geräte angezeigt, welche sich in der Nähe befinden. Wird kein Gerät gefunden, erscheint die Meldung, ob sich die korrekte Software auf dem Board befindet. Dies muss die Software «PerCen» sein.

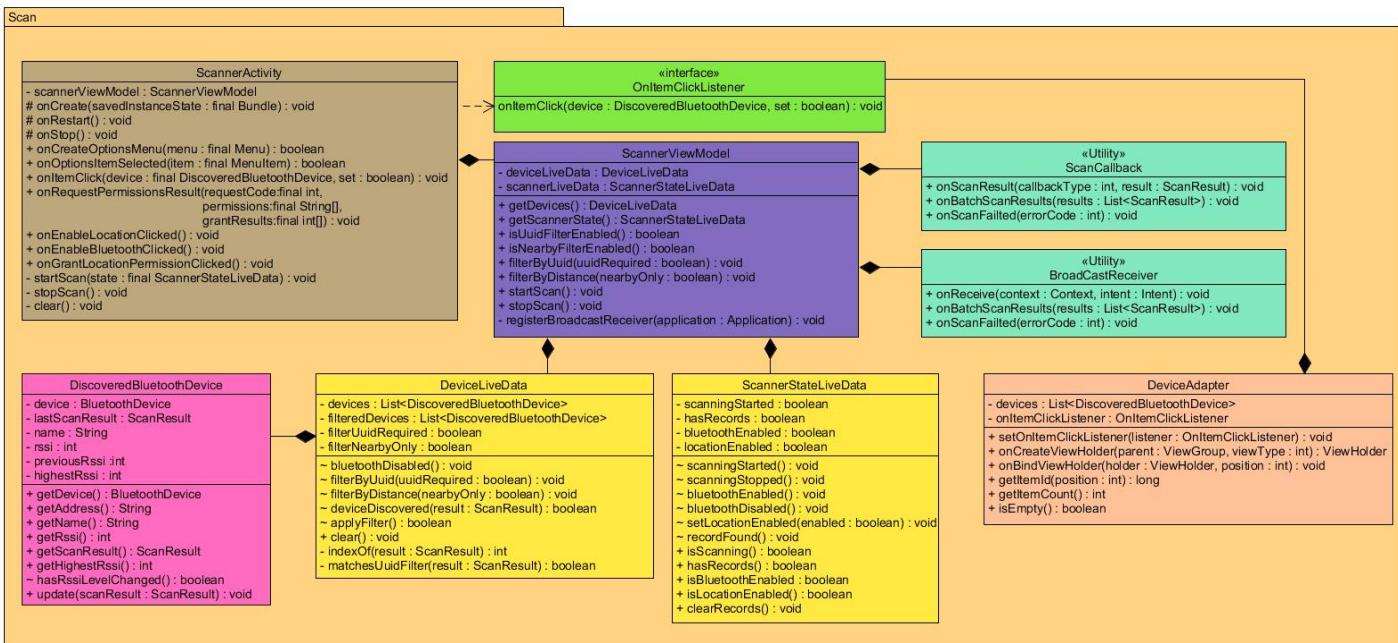
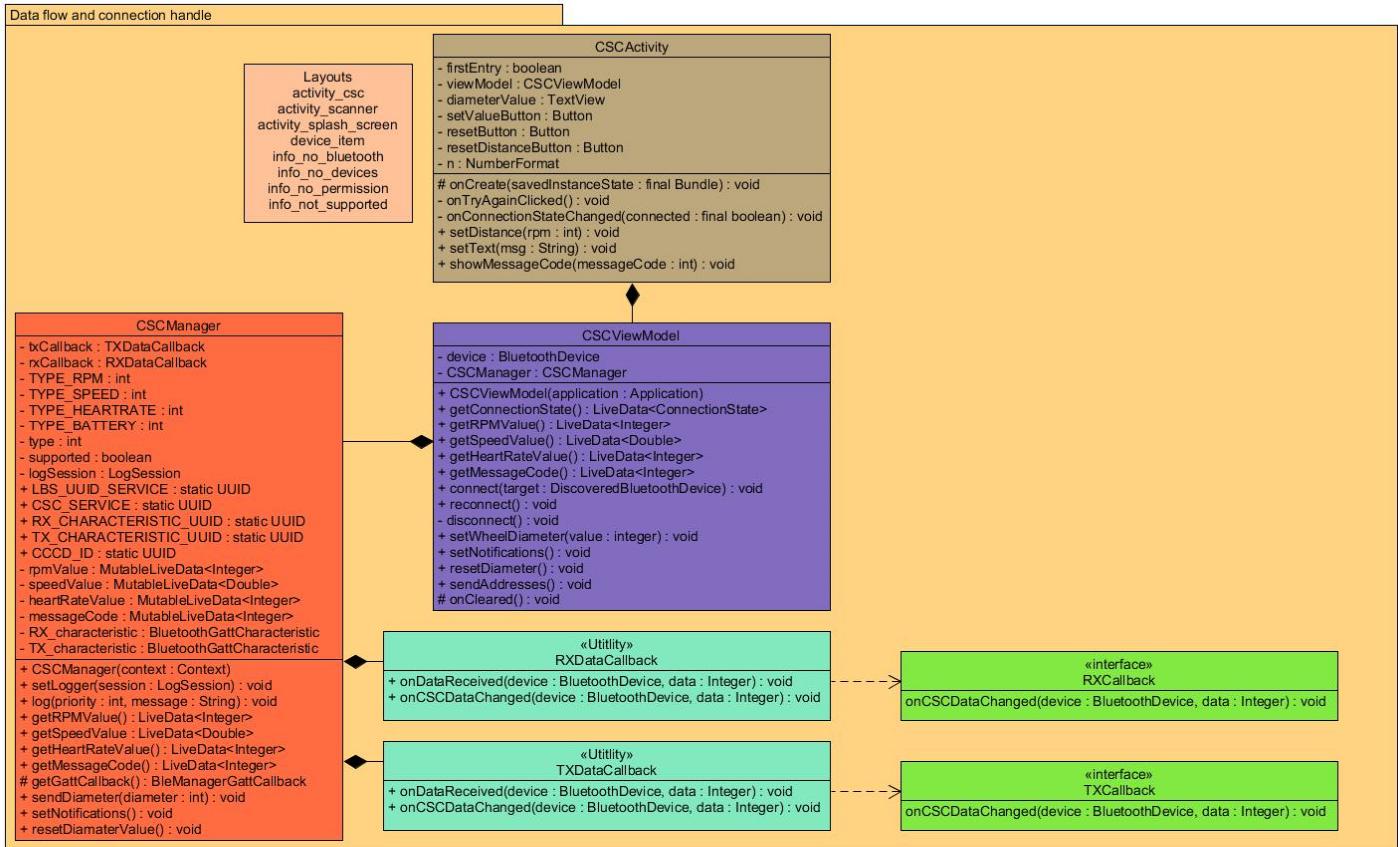


Durch einen Klick auf ein Feld, färbt sich dieses Feld hellblau und ist nun ausgewählt. Durch eine erneute Betätigung desselben Feldes, wechselt die Farbe des Feldes wieder zur Standardfarbe und das Gerät ist nicht mehr ausgewählt. Hier müssen jetzt das Board, sowie alle gewünschten Sensoren ausgewählt werden. Mit einer Betätigung der Taste «Connect», wird der nächste Schritt eingeleitet.

3. Es erscheint ein Ladefenster, auf dem zuerst der Text «Connecting» und anschliessend «Initializing» zu sehen ist. Während dieser Zeit wird die Verbindung von der Applikation zum Board aufgebaut.
4. Wurde die Verbindung erfolgreich aufgebaut, erscheint die Daten Seite. Hier werden Informationen wie Geschwindigkeit, Trittfrequenz, gefahrene Distanz und Herzfrequenz zu sehen sein. Am unteren Bildschirmrand erscheinen Kurzmitteilungen, zu welchen Sensoren eine Verbindung aufgebaut werden konnte. Sobald alle Verbindung aufgebaut wurden und sich ein Geschwindigkeitssensor darunter befindet, erscheint die Aufforderung zur Eingabe des Rad-durchmessers. Dieser ist notwendig, um die aktuelle Fahrgeschwindigkeit zu berechnen.
Ansonsten werden direkt die aktuellen Daten angezeigt.
5. Am rechten Rand neben jeder Anzeige befindet sich der aktuelle Batteriestand des jeweiligen Sensors.



Anhang 6 – Klassendiagramm Applikation



Anhang 7 – Tests Applikation, nRF5340

In der folgenden Tabelle wurden Tests vom Start der Applikation, bis zum Datenaustausch mit den Sensoren durchgeführt. Hierbei wurden alle verschiedenen Verbindungsmöglichkeiten zwischen der Android Applikation und dem nRF5340 getestet. Die Tests 1 bis 12 kontrollieren einen gesamten Durchgang für die Nutzung der Applikation mit der Verwendung aller Sensoren. Der Test Nummer 6 wurde 50-mal durchgeführt. In diesem Fall werden zu allen Sensoren (Geschwindigkeits-, Trittfrequenz- und Herzfrequenzsensor) eine Verbindung aufgebaut. In den Test 13 bis 18 wurden die anderen Konfigurationsmöglichkeiten getestet. Bei diesen wurden der Test 13 und 16 auch 50-mal durchgeführt. So wurden die Verbindungsmöglichkeiten zu einem, zwei und drei Sensoren 50-mal getestet.

Test N°	Beschreibung des Tests	Was geschieht?	OK/NOT OK	Beschreibung Fehler / Hinweis
1	Die Applikation wird gestartet.	Die erste Seite wird angezeigt. Zu sehen ist das HES-SO Logo und das Valais Excellence Logo.	OK	
2a	Der Scan Vorgang wird gestartet.	Nach einer Sekunde wird die nächste Seite, die Scan Seite angezeigt. Es werden alle erkannten Geräte angezeigt.	OK	
2b	Der Bluetooth ist ausgeschaltet.	Eine neue Seite öffnet sich mit der Aufforderung Bluetooth einzuschalten. Durch Betätigung der Taste «Enable» wird dies erreicht.	OK	
2c	Der Standort ist ausgeschaltet.	Eine neue Seite öffnet sich mit der Aufforderung den Standortzugriff zuzulassen. Durch Betätigung der Taste «Grant Permission» wird dies erreicht. Durch Betätigung der Taste «Settings» werden die Standorteinstellung des Smartphones aufgerufen.	OK	
2d	Filter «Only devices with Data service» wird aktiviert.	Es werden nur noch die Geräte angezeigt, welche den Data Service benutzen.	OK	
2e	Filter «Only nearby devices» wird aktiviert.	Es werden nur noch die Geräte angezeigt, welche sich in der Nähe des Smartphones befinden.	OK	
2f	Es werden keine Geräte gefunden.	Es wird eine Information angezeigt, ob das korrekte Programm auf das Board geladen wurde.	OK	
3	Es werden einige erkannte Geräte mittels eines Klicks ausgewählt.	Die Farbe der gesamten Zeile des ausgewählten Gerätes ist hellblau. Durch erneute Betätigung des Feldes, erscheint das Feld in der vorderen Farbe.	OK	
4a	Betätigung der Taste «Connect» ohne eine Auswahl von Geräten.	Es erscheint eine Mitteilung, dass mindestens ein Gerät ausgewählt werden muss.	OK	

4b	Betätigung der Taste «Connect» ohne die Auswahl eines Nordic Boards.	Es erscheint eine Mitteilung, dass ein Nordic Board ausgewählt werden muss.	OK	
4c	Betätigung der Taste «Connect» mit einer Auswahl von 5 oder mehr Geräten.	Es erscheint eine Mitteilung, dass maximal vier Geräte ausgewählt werden können. (Nordic Board, Geschwindigkeitssensor, Trittfrequenzsensor, Herzfrequenzsensor)	OK	
4d	Betätigung der Taste «Connect» mit einer Auswahl in der sich ein unbekanntes Gerät befindet.	Es erscheint eine Mitteilung, dass unbekannte Sensoren ausgewählt wurden.	OK	
4e	Betätigung der Taste «Connect» mit einer Anzahl von ausgewählten von Geräten die kleiner als 5 ist und ein Nordic Board befindet sich darunter.	Es öffnet sich eine neue Seite. Zuerst ist ein Balken mit der Schrift «Connecting...» zu sehen. Anschliessend ist die Schrift «Initializing» ersichtlich. Es wird eine Verbindung zum Nordic Board aufgebaut.	OK	
5a	Der Initialisierungsvorgang war nicht erfolgreich.	Der Service für den Datenaustausch wurde nicht gefunden. Es wird ein passender Text mit Informationen angezeigt.	OK	
5b	Der Initialisierungsvorgang war erfolgreich.	Es öffnet sich eine neue Seite auf der Geschwindigkeit, Trittfrequenz, die gefahrene Strecke und die Herzfrequenz ersichtlich ist. Zu Beginn sind alle Werte auf 0. Es gibt ein Eingabefeld, hier muss eventuell der Durchmesser eingegeben werden.	OK	In seltenen Fällen werden die Benachrichtigungen in der Applikation deaktiviert. In diesem Fall wird die Verbindung zur Applikation abgebrochen und der Nutzer wird aufgefordert zurück zur Scan Seite zu navigieren und erneut eine Verbindung aufzubauen.

6	Aufbau der Verbindungen zu den Sensoren.	Es erscheinen Nachrichten, sobald die Verbindungen aufgebaut wurden. Anschliessend folgt eine Nachricht mit der Aufforderung zur Eingabe des Durchmessers der Räder. Es ist auch möglich, dass nach dem Verbindungsaufbau der Service nicht gefunden wird, in diesem Fall wird die Verbindung getrennt und neu aufgebaut. Manchmal wird die Verbindung zu einem Sensor abrupt unterbrochen (Fehlercode 0x08 Connection Timeout), dann wird nach diesem Sensor erneut gesucht und wieder eine Verbindung aufgebaut.	OK	Dieser Test wurde 50-mal durchgeführt. In 3 Fällen wurde das Programm nicht mehr wunschgemäß ausgeführt. Es blieb im Zustand «idle» ¹⁹ . Dieser Fall tritt ein, wenn zuerst ein Timeout Fehler (0x08) auftritt und nach erneuter Verbindung der Service nicht gefunden wird. Das Board muss zurückgesetzt werden.
7a	Es wird kein Durchmesser der Räder im Eingabefeld eingegeben.	Die Geschwindigkeitsanzeige bleibt bei 0 km/h.	OK	
7b	Es wird ein Durchmesser der kleiner als 1 Inch ist eingegeben.	Es erscheint eine Mitteilung, dass der Wert grösser als 1 Inch sein muss.	OK	
7c	Es wird ein Durchmesser der grösser als 255 Inch ist eingegeben.	Es erscheint eine Mitteilung, dass der Wert kleiner als 255 Inch sein muss.	OK	
7d	Es wird ein Durchmesser zwischen 1 und 255 Inch im Eingabefeld eingegeben.	Nach der Betätigung der Taste «Set» werden direkt die aktuellen Daten des verbundenen Geschwindigkeitssensors angezeigt. Die Taste «Set» ist blockiert und kann nicht mehr betätigt werden.	OK	
7e	Es wird die Taste «Reset» beim «Wheel Diameter» betätigt.	Der Geschwindigkeitswert zeigt '0' an. Der Wert für den Durchmesser wird auch auf '0' gesetzt. Die Taste «set» ist wieder verfügbar.	OK	
8	Es wird die Taste «Reset» beim Feld «Distance» betätigt.	Der Wert, welcher bei «Distance» angezeigt wird, wird auf '0' zurückgesetzt.	OK	

¹⁹ Leerlauf

9	Auf der Seite, auf der die Daten zu sehen sind, wird die «Seite zurück» Taste oben links betätigt.	Es öffnet sich wieder die Scan Seite. Es werden alle erkannten Geräte angezeigt. Es erscheint eine Mitteilung, dass die Verbindung zum Nordic Board unterbrochen wurde.	OK	Die Felder die zuvor ausgewählt wurden sind noch hellblau. Aber es sind keine Geräte gespeichert.
10	Es wird erneut das Nordic Board mit den Sensoren für den Verbindungsaufbau ausgewählt.	Es öffnet sich die Seite mit den «Connecting...» und «Initializing» Balken. Anschliessend öffnet sich das Fenster mit den Daten. Die Daten sind wieder ersichtlich. Nach dem der Durchmesser der Räder eingegeben wurde, wird auch die Geschwindigkeit wieder angezeigt.	OK	
11	Die Applikation wird während der Nutzung geschlossen.	Die Verbindung zum nRF5340 wird getrennt. Die Verbindungen zu den Sensoren bleiben bestehen.	OK	
12	Die Applikation wird erneut geöffnet.	Es erscheint wieder die Scan Seite. Hier muss wieder der nRF5340 und die Sensoren ausgewählt werden. Anschliessend wird die Verbindung zum nRF5340 aufgebaut. Da die Sensoren immer noch verbunden sind, werden direkt die Daten angezeigt. Um die Geschwindigkeit zu erhalten, muss erneut der Raddurchmesser angegeben werden.	OK	Wenn neue Sensoren verwendet werden wollen, muss der nRF5340 zurückgesetzt werden.
13	Im Scan Fenster wird nur ein Geschwindigkeitssensor ausgewählt.	Es wird eine Verbindung zu diesem Sensor aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Nach dem der Durchmesser eingegeben wurde und der Nutzer losfährt, wird die aktuelle Geschwindigkeit und die gefahrene Distanz angezeigt.	OK	Dieser Test wurde 50-mal durchgeführt. In allen Fällen wurde das Programm wunschgemäß ausgeführt.
14	Im Scan Fenster wird nur ein Trittfrequenzsensor ausgewählt.	Es wird eine Verbindung zu diesem Sensor aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt, wird die aktuelle Trittfrequenz angezeigt.	OK	

15	Im Scan Fenster wird nur ein Herzfrequenzsensor ausgewählt.	Es wird eine Verbindung zu diesem Sensor aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt.	OK	
16	Im Scan Fenster wird ein Geschwindigkeits- und ein Trittfrequenzsensor ausgewählt.	Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt wird die aktuelle Trittfrequenz angezeigt. Nach der Eingabe des Raddurchmesser wird auch die aktuelle Geschwindigkeit angezeigt.	OK	Dieser Test wurde 50-mal durchgeführt. In 3 Fällen wurde das Programm nicht mehr wunschgemäß ausgeführt. Nach dem mehrmals hintereinander der Fehler 0x08 (Timeout Error) nach erneuter Verbindung auftrat, gab es einen «Assertion Fault» ²⁰ . Das Board musste neu
17	Im Scan Fenster wird ein Geschwindigkeits- und ein Herzfrequenzsensor ausgewählt.	Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt. Nach der Eingabe des Raddurchmesser wird auch die aktuelle Geschwindigkeit angezeigt.	OK	
18	Im Scan Fenster wird ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt.	Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt wird die aktuelle Trittfrequenz angezeigt. Nachdem der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt.	OK	

Durchgeführt am: 16.08.2021

Durchgeführt durch Bastian Schwery Bastian

²⁰ Behauptungsfehler, stoppt das Programm.

13 Quellenverzeichnis

Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK_SPL.jpg 3	2
Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK_SPL.jpg 3	2
Quelle: https://play-lh.googleusercontent.com/WshzYj8MVD1cDz55UP3z0Hl3eqUiBlkez5GSYr0yxwGSbVTkdLILb0m0SXkqX-tWC 3	2
Quelle: https://www.mouser.de/images/marketingid/2019/microsites/153282850/2020-12-21_15-07-43.png 4	3
Quelle: https://static.garmincdn.com/en/products/010-12103-00/g/cadence_v03-eb1598f4-1e01-4e01-8691-8ddf6e108677.jpg 5	4
Quelle: https://media.alltricks.com/hd/15743025eaaa35aa3e5d9.44386646.jpg 6	4
Quelle: https://miro.medium.com/max/1838/1*7R_hkwuk8v7gClsvXtoqPw.png 7	8
Quelle: https://learn.adafruit.com/assets/86833 8	9
Quelle: https://learn.adafruit.com/assets/86833 9	9
Quelle:	
https://www.mouser.ch/images/marketingid/2017/img/116326113_Nordic_Thiny52IoTSensorDevelopmentKit.png?v=052820.0204 10	13
Quelle: Eigene Darstellung 11	20
Quelle: Eigene Darstellung 12	21
Quelle: Eigene Darstellung 13	23
Quelle: Eigene Darstellung 14	29
Quelle: Eigene Darstellung 15	29
Quelle: Eigene Darstellung 16	29
Quelle: Eigene Darstellung 17	31
Quelle: Eigene Darstellung 18	34
Quelle: Eigene Darstellung 19	36
Quelle: Eigene Darstellung 20	37

14 Literaturverzeichnis

- [1] „Nordic Semiconductor,“ [Online]. Available: <https://www.nordicsemi.com/>. [Zugriff am 10 Mai 2021].
- [2] „Nordic Semiconductor,“ [Online]. Available: <https://www.nordicsemi.com/Products/Development-hardware/nrf5340-dk/getstarted>. [Zugriff am 10 Mai 2021].
- [3] „Garmin,“ [Online]. Available: <https://buy.garmin.com/de-CH/CH/p/641221>. [Zugriff am 15 Juni 2021].
- [4] „Polar,“ [Online]. Available: <https://www.polar.com/ch-de>. [Zugriff am 28 Juli 2021].
- [5] „Polar,“ [Online]. Available: https://www.polar.com/ch-de/about_polar/who_we_are. [Zugriff am 28 Juli 2021].
- [6] „Nordic Semiconductor,“ [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/gs_installing.html. [Zugriff am 10 Mai 2021].
- [7] „Github,“ [Online]. Available: <https://github.com/zephyrproject-rtos/zephyr/tree/main/samples/basic/blinky>. [Zugriff am 25 Mai 2021].
- [8] „Zephyr,“ [Online]. Available: <https://docs.zephyrproject.org/latest/application/index.html>. [Zugriff am 2 Juni 2021].
- [9] „Nordic Semiconductor,“ [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/app_boards.html. [Zugriff am 11 Mai 2021].
- [10] „Nordic Semiconductor,“ [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/include/bluetooth/services/bas_client.html. [Zugriff am 22 Juli 2021].
- [11] „Centare,“ [Online]. Available: <https://www.centare.com/insights/what-is-bluetooth-low-energy>. [Zugriff am 14 Juni 2021].
- [12] „Adafruit,“ [Online]. Available: <https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap>. [Zugriff am 10 Juni 2021].
- [13] „Adafruit,“ [Online]. Available: <https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>. [Zugriff am 10 Juni 2021].
- [14] „Github,“ [Online]. Available: <https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap>. [Zugriff am 26 Mai 2021].
- [15] „Google Play Store,“ [Online]. Available: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=de_CH&gl=US. [Zugriff am 17 Mai 2021].

-
- [16] „Nordic Semiconductor,“ [Online]. Available: <https://www.nordicsemi.com/Software-and-tools/Prototyping-platforms/Nordic-Thingy-52>. [Zugriff am 10 Juni 2021].
 - [17] „Github,“ [Online]. Available: <https://github.com/zephyrproject-rtos/zephyr/tree/main/samples/bluetooth/central>. [Zugriff am 1 Juni 2021].
 - [18] „Nordic Semiconductor,“ [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/include/bluetooth/scan.html. [Zugriff am 22 Juni 2021].
 - [19] „Github,“ [Online]. Available: https://github.com/nrfconnect/sdk-nrf/tree/master/samples/nrf9160/lte_ble_gateway. [Zugriff am 1 Juni 2021].
 - [20] „Bluetooth,“ [Online]. Available: <https://www.bluetooth.com/de/specifications/specs/cycling-speed-and-cadence-profile-1-0/>. [Zugriff am 14 Juni 2021].
 - [21] Y. Rossier, „IoT bike gateway for Swiss Cycling,“ Sion, 2020.
 - [22] „Nordic Semiconductor,“ [Online]. Available: <https://devzone.nordicsemi.com/nordic/nrf-connect-sdk-guides/b/getting-started/posts/ncs-ble-tutorial-part-1-custom-service-in-peripheral-role>. [Zugriff am 14 Juli 2021].
 - [23] „UUID online,“ [Online]. Available: <https://uuidonline.net/>. [Zugriff am 14 Juli 2021].
 - [24] „Android Studio,“ [Online]. Available: https://developer.android.com/studio?hl=de&gclid=CjwKCAjwiLGGBhAqEiwAgq3q_o36qDS-GY-Np7uyRZC4-uOIHB8YfH2V_JkCmjwhXgom77PBtf_UPhoCiOoQAvD_BwE&gclsrc=aw.ds. [Zugriff am 14 Juni 2021].
 - [25] „Github,“ [Online]. Available: <https://github.com/NordicSemiconductor/Android-nRF-Blinky>. [Zugriff am 3 Juni 2021].
 - [26] „Apple Store,“ [Online]. Available: <https://apps.apple.com/ch/app/nrf-connect-for-mobile/id1054362403>. [Zugriff am 7 Juni 2021].
 - [27] „Github,“ [Online]. Available: https://github.com/nrfconnect/sdk-nrf/tree/master/samples/bluetooth/peripheral_lbs. [Zugriff am 26 Mai 2021].
 - [28] „Nordic Semiconductor,“ [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_gsg_ses%2FUG%2Fgsg%2Fsoftdevices.html. [Zugriff am 12 Mai 2021].
 - [29] „Medium,“ [Online]. Available: <https://medium.com/swlh/the-gnu-toolchain-explained-4bf14666bc03>. [Zugriff am 11 Mai 2021].