INTERN REPORT HRO - TI

January 7, 2020

Paul Wondel
Stud.nr.: 0947421
Internship period: 1 September 2019 - 7 Februari 2020
First Attempt

Hogeschool Rotterdam
Technische Informatica

`Rotterdam University of Applied Sciences`

`Applied Computer Science`

Teachers: G.W.M. van Kruining, R. van Doorn
Intern Supervisor: A.C. van Rossum

Glossary

- **BLE** - Bluetooth Low Energie
- **Crownstone** - The smart device created by Crownstone B.V.
- **JSON** - JavaScript Object Notation: A extension file type
- **OS** - Operating System
- **CLI** - CommandLine Interface
- **pip** - Python Package Manager for downloading and installing packages and their dependencies for python
- **git** - Distributed version-control system for tracking changes in source code during software development.
- **npm** - npm is a package manager for the JavaScript programming language
- **CPU** - Computer Processing Unit
- **Crownstone code | Bluenet code** - Source code for the crownstone devices. On github the crownstone code repository is called bluenet.
- **MBR** - Master Boot Record

Contents

1 Introduction

In my third year of education I am required to do an internship at a IT company. In this report I will write about my experiences during the internship and the process of my assignment. The company where I did the internship is named Crownstone B.V. for 2 semesters. From 1st September 2019 until 7th Februari 2020 I will be working on the assignment, which will be explained in the next chapters.

Crownstone B.V. is a company that combines indoor localization with automation for home and office spaces. They are the creators of the Crownstones. Crownstones are smart devices that make your home or office smart. A crownstone can function as a switch, a dimmer a power monitor and a standby killer. The devices are connected to a smartphone using Bluetooth Low Energie (BLE). The crownstones use indoor positioning. When connected to an owner's smartphone they can calculate his posistion and execute their functions in a room based on the posistion of the owner's smartphone. The crownstones have machine learning capabilities. They have the ability to learn which type of rooms there are in the huis and where they are.

The assignment that I have been given is to make the crownstone compatible with the arduino environment.

2 Assignment Information

This assignment was given to me by the CEO of Crownstone B.V., Anne van Rossum. The assignment is to make the Crownstone compatible to run Arduino code and to be run by Arduino code. The reason for this assignment is to attract the open-source community. At the moment the Crownstone runs on its own code. The Crownstone has many functions such as dimming lights, turning lights on and off, locating your position in the house. All these functions need to be accessible to the programmer when writing a arduino script.

2.1 Requrement List

This assignment has a list of requirements that need to be met before it can be considered as finished.

- DFU (update over the air) flashing/uploading of code intead of OpenOCD/JTAG
- Should be able to run basic arduino code
- Needs to be compatible with Platformio
- Needs to be compatible with Arduino IDE
- Automatic download of the toolchain
- Crownstone needs to be added to the Arduino IDE board manager

3 Crownstone B.V. Background

Crownstone, founded in 2016, is a company that combines indoor localization with building automation for homes and offices. Crownstone went through the Rockstart accelerator (demo day July 2016). We acquired a large network of business contacts, have a scrutinized business model and it ranks our company as a potentially disruptive and scalable player. Crownstone has support from Almende as research company and from Almende Investments as investor. This guarantees that Crownstone can produce beyond state-of-the-art technology and has enough budget to grow organically. Crownstone's unique selling point is indoor localization. This has not been capitalized upon by any competitor in the home automation market, and only a few in the building automation market.

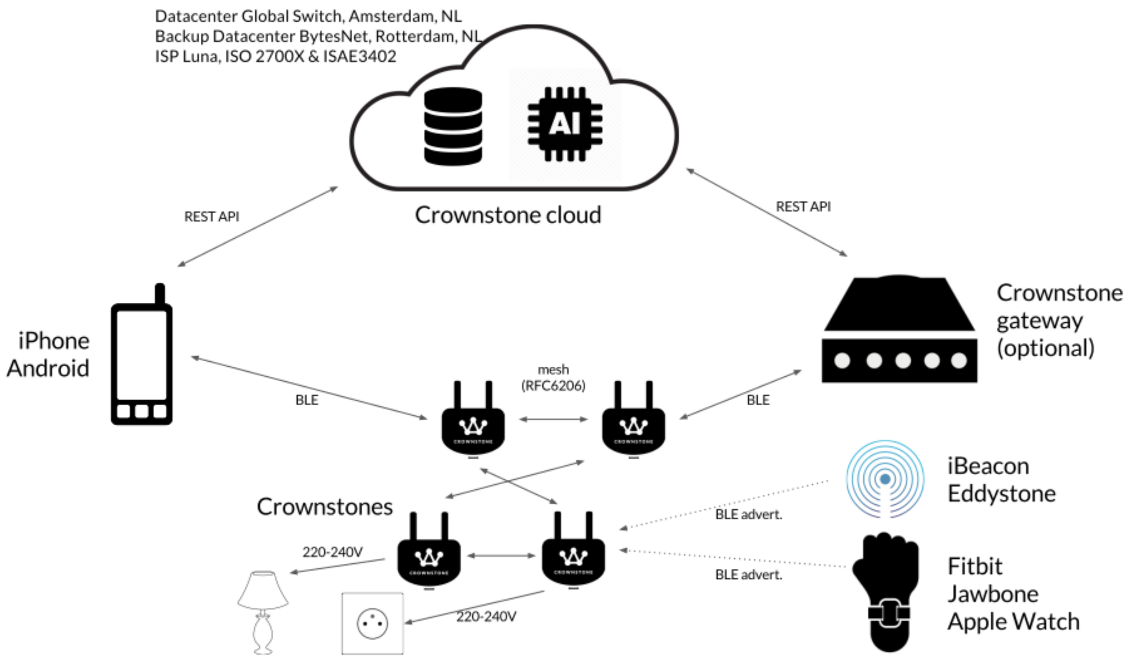


Figure 1: Original Crownstone Architecture

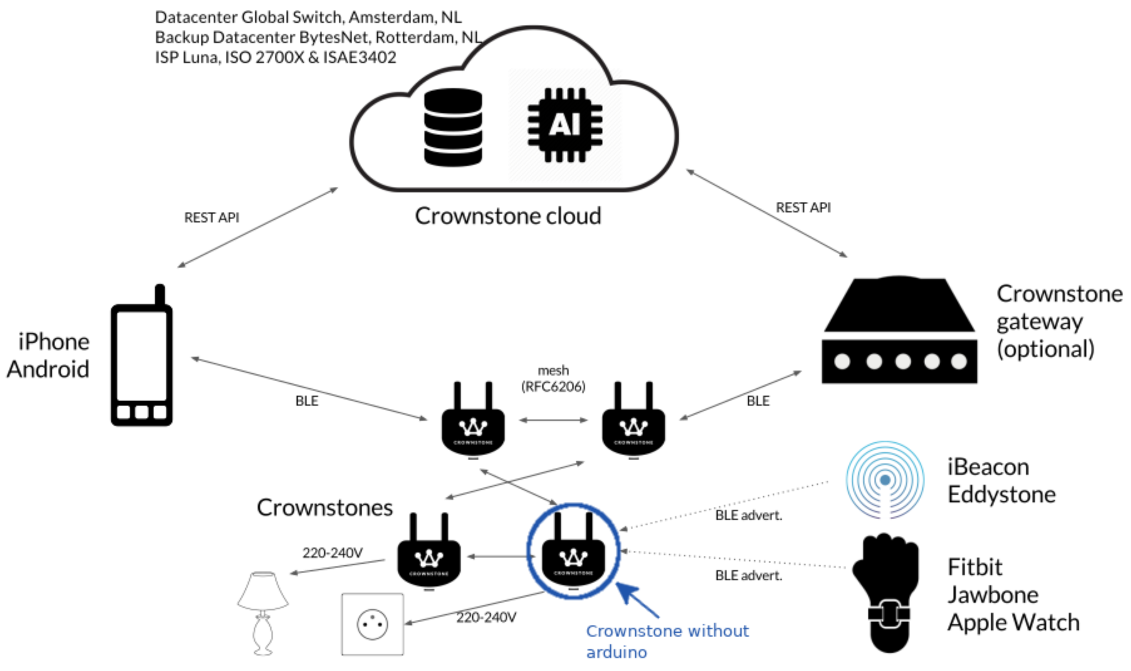


Figure 2: Current Crownstone Architecture

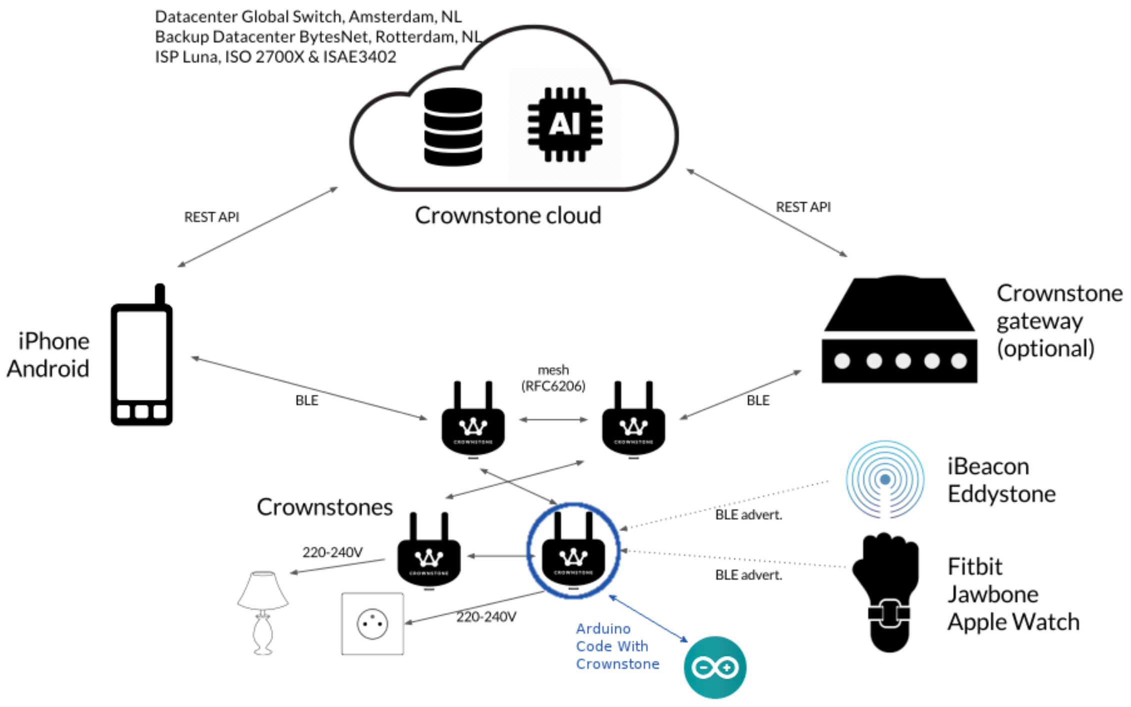


Figure 3: Desired Crownstone Architecture

4 Internship goals

4.1 Activities

Managing The graduate is able to manage the process of development, deployment and use of ICT systems in a given professional situation, taking into account the context and relevant stakeholders.

Analysing The graduate can analyze a problem by collecting data on existing or new technologies, users, processes, products or information flows, describing them, processing them into usable information, forming an opinion on this and selecting or formulating a solution direction based on this.

Advising The graduate can give substantiated and guiding advice about processes, software and/or new technologies and can present this convincingly and comprehensibly.

Designing The graduate can design a system in terms of functionality, interaction, structure and architecture within predefined frameworks.

Realizing The graduate can convert a design into a usable IT solution that connects to existing systems by writing, testing, debugging, optimizing and documenting. This ICT solution includes a combination of hardware and software, in the sense that software is written for hardware that is newly assembled from existing components (sensors, actuators, microcontrollers, communication equipment, etc.), or that software is written for an existing, special-purpose hardware system.

Behaald op stage:

Activity	Chapter Reference
Beheren	
Analyseren	5.18
Adviseren	
Ontwerpen	
Realiseren	

4.2 Projectgoals:

1. The student can clearly formulate and clearly define the scope of his own internship assignment.
2. The student can translate the requirements and wishes of the customer into a possible solution.
3. The student can perform a stakeholder analysis.
4. The student can weigh up alternatives and existing solutions, taking into account the stakeholders and / or the technology.
5. The student can map and apply the Software Configuration Management.
6. The student can draw up a risk log, actively keep it up and act accordingly.
7. The student applies version management to documentation.
8. The student can apply issue tracking.
9. The student can appropriately advise the client on the results and conclusions of the analysis.
10. The student can map the system architecture.
11. The student can determine which designs are relevant to their own assignment and prepare these designs as appropriate.
12. The student can clearly demonstrate the correct functioning of the realized prototype.
13. The student has demonstrably tested the realized solution with appropriate test forms and documented this.

Proof during internship

Project goal	Reference page
1	6
2	
3	13
4	
5	
6	11
7	
8	11
9	
10	7
11	
12	
13	50

4.3 Risk Register

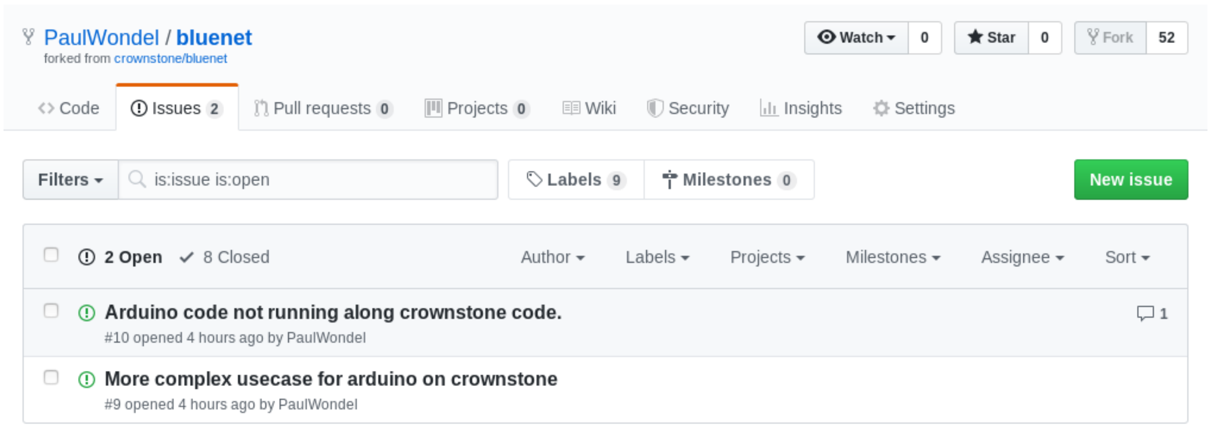
Every project comes with its own risk factors. In this table I have written down some risk that may play of might have played a role during the internship.

Table 1: Risk Register Table

Risk Register				
ID	Description	Probability (1 - 5)	Impact (1 - 5)	Risk Score (1 - 25)
1	Calling in sick	5	5	25
2	Missing a deadline	4	3	12
3	Hardware breaks	3	2	6
4	Supervisor is unavailable	3	2	6
5	Laptop breaks	1	5	5
6	Software gets corrupted	3	4	12
7	Documentation files are lost	5	5	25
8	Research takes longer than expected	5	3	15
9	Missing expertise in work area	5	5	25
10	Wrong approach on assignment (loss of time)	4	3	12
11	Arduino code doesn't compile	5	5	25
12	Crownstone Bluenet code doesn't compile	5	5	25
13	J-link connection with Nordic nRF52-DK doesn't work	5	5	25
14	Nordic Developer Software doesn't build or execute	5	5	25
15	GCC compiler doesn't work	5	5	25
16	Debugging of code takes too long	5	5	25
17	The linkerscript doesn't link the c program files	5	5	25

4.4 Issue Tracking

Issue tracking is done using the issue system from github. There I write down the issues I encounter during the assignment.

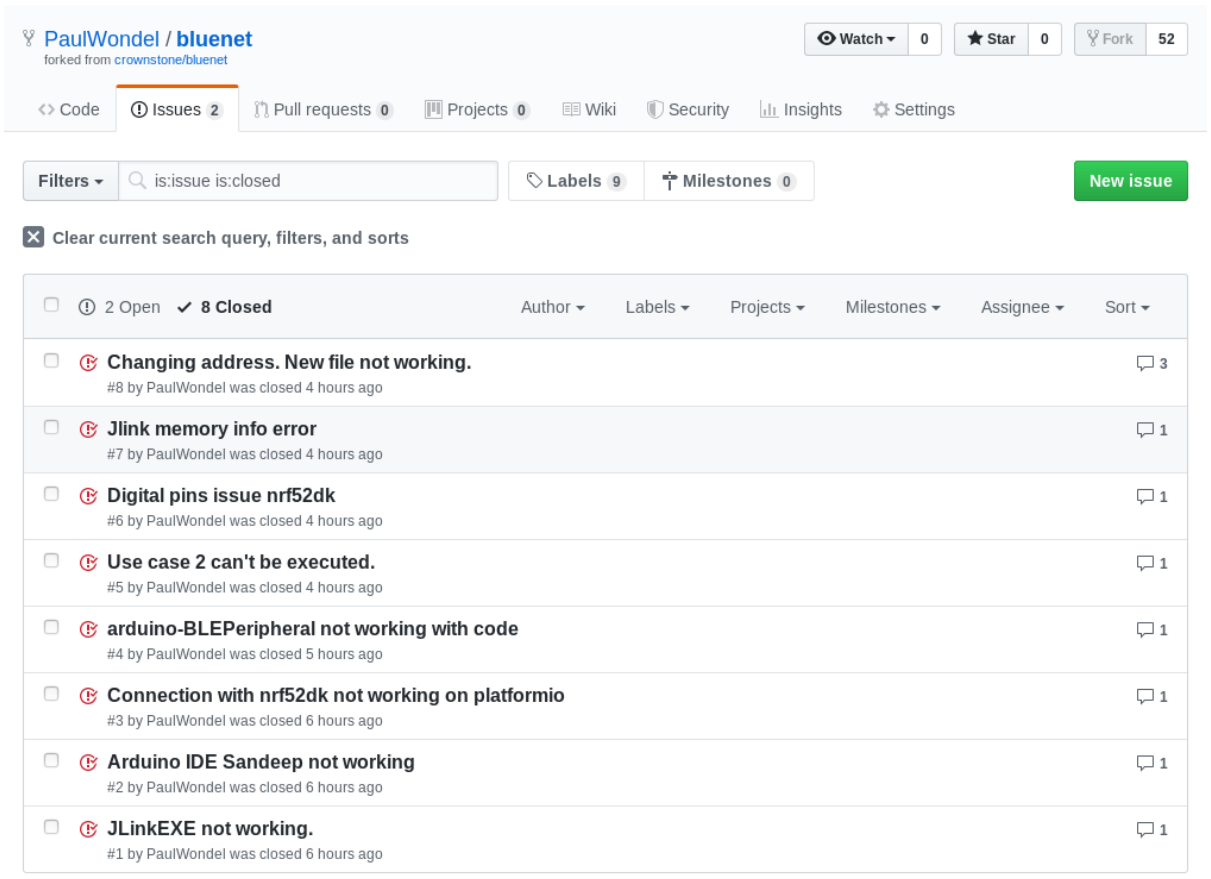

PaulWondel / bluenet
forked from crownstone/bluenet

Filters: is:issue is:open

2 Open ✓ 8 Closed

- Arduino code not running along crownstone code.
  #10 opened 4 hours ago by PaulWondel — 1
- More complex usecase for arduino on crownstone
  #9 opened 4 hours ago by PaulWondel

Figure 4: Example: Open issues on Github
(dated: 5 November 2019)


PaulWondel / bluenet
forked from crownstone/bluenet

Filters: is:issue is:closed

Clear current search query, filters, and sorts

2 Open ✓ 8 Closed

- Changing address. New file not working.
  #8 by PaulWondel was closed 4 hours ago — 3
- Jlink memory info error
  #7 by PaulWondel was closed 4 hours ago — 1
- Digital pins issue nrf52dk
  #6 by PaulWondel was closed 4 hours ago — 1
- Use case 2 can't be executed.
  #5 by PaulWondel was closed 4 hours ago — 1
- arduino-BLEPeripheral not working with code
  #4 by PaulWondel was closed 5 hours ago — 1
- Connection with nrf52dk not working on platformio
  #3 by PaulWondel was closed 6 hours ago — 1
- Arduino IDE Sandeep not working
  #2 by PaulWondel was closed 6 hours ago — 1
- JLinkEXE not working.
  #1 by PaulWondel was closed 6 hours ago — 1

Figure 5: Example: Closed issues on Github
(dated: 5 November 2019)

4.5 Version Control

4.6 Stakeholder Analysis

During this assignment there are stakeholders that have certain interests and influences on the assignment. In the table below the stakeholders are described along with their level of influence and relation.

Table 2: Stakeholder Analysis Table

Stakeholder	Info	Interest	Influence On Assignment	Relative priority
Anne Van Rossum	CEO & Product Owner	Has set the requirements of the end product	High	High
Alex de Mulder	Designer & Software developer	Works on the software which the assignment is depending on	Medium	Low
Bart van Vliet	Software developer	Is the main developer on the firmware for the crownstone	High	Medium
Arend de Jonge	Algorithm Designer & Firmware Developer	Works on firmware, which is a heavy influence on the assignment	Low	Low
Crownstone Customers	-	Is the target group of the assignment	Low	High
Arduino Community	Community that is filled with arduino code developers for various boards	After the assignment the product owner will reach out to arduino developers	Medium	High

4.7 Ontwerpen

5 Research

In this chapter I have collected all the information I have come across during my research of the project. Literature and theoretical cases that have been researched during this internship are also included in this chapter.

5.1 Crownstone Devices

Crownstones are smart devices that can change your home or office into a smart environment. A crownstone can function as a switch, a dimmer a power monitor and a standby killer. The devices are connected to a smartphone using Bluetooth Low Energie (BLE). The crownstones use indoor positioning. When connected to an owner's smartphone they can calculate his posistion and execute their functions in a room based on the posistion of the owner's smartphone. The crownstones have machine learning capabilities. They have the ability to learn which type of rooms there are in the huis and where they are.

5.1.1 Crownstone Plug

The Crownstone plug is a crownstone device that can be easily inserted into an outlet. The plug has the same properties as the in-built crownstone.

5.1.2 In-built Crownstone

The in-built crownstone is a crownstone device tat needs to be installed into the electrical circuit of the house, preferably near the outlets. The in-built has the same properties as the plug crownstone.

5.1.3 Guidestone

This is an iBeacon that is used for the positioning.

5.2 Crownstone Code

De Crownstone code is geschreven door het team van Crownstone B.V. zelf. Crownstone heeft eigen board waarop hij de code draait. De Crownstone draait op een nRF52 Chip van Nordic. De code van de crownstone is open source en heet bluenet. De code voor de crownstone is te vinden op: `https://github.com/crownstone/bluenet`.

5.3 Arduino IDE

Een van de onderdelen van de opdracht is het draaien van arduino code draaien op de crownstone via de arduino. Om borden te kunnen toevoegen aan de arduino IDE moet er een board

5.4 PlatformIO

PlatformIO[4] is a cross platform code builder and library manager for other platforms like Arduino or MBED support. PlatformIO has toolchains, debuggers and frameworks that work on popular plantforms

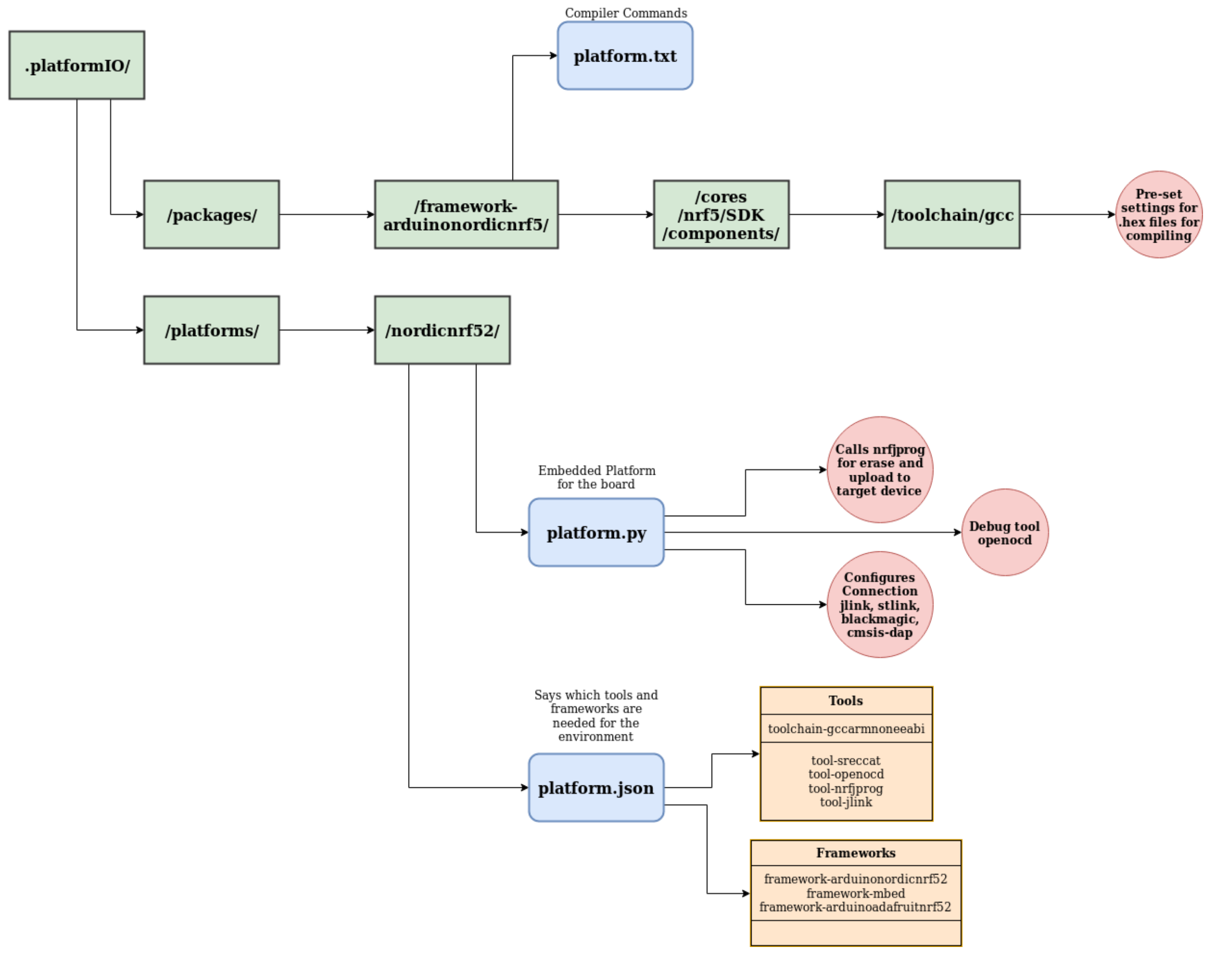

.platformIO/
Compiler Commands
platform.txt
/packages/
/framework-arduinonordicnrf5/
/cores /nrf5/SDK /components/
/toolchain/gcc
Pre-set settings for .hex files for compiling
/platforms/
/nordicnrf52/
Embedded Platform for the board
platform.py
Calls nrfjprog for erase and upload to target device
Debug tool openocd
Configures Connection jlink, stlink, blackmagic, cmsis-dap
Says which tools and frameworks are needed for the environment
platform.json
Tools
toolchain-gccarmnoneeabi
tool-sreccat
tool-openocd
tool-nrfjprog
tool-jlink
Frameworks
framework-arduinonordicnrf52
framework-mbed
framework-arduinoadafruitnrf52

Figure 6: File Structure PlatformIO

like Windows, Mac en linux laten werken. There is support for more than 200 developmnent boards along with 15 development platforms and 10 frameworks. Most of the popular boards are supported. PlatformIO was actually meant to be used through the command line, but with success it is able to be with other IDE's like Eclipse of Visual Studio.

PlatformIO has the options and possibility to add new boards. For a new board the be added on PlatformIO, there needs to be a **JSON structure (.json)** file made for it. PlatformIO published a installation manual for adding new boards.

- Source: PlatformIO documentation

5.4.1 Development platform

Bij platformIO hoort er ook een development platform te zijn dat de toolchains, build scripts en instellingen voor het bord. Een custom Development Platform heeft een paar onderdelen nodig om gebruikt te kunnen werken n.l. packages, een manifest file (platform.json), buildscript (main.py).

- Source: PlatformIO documentation

5.4.2 Advanced Scripting

PlatformIO Build System allows a programmer to extend the build process with their own custom scripts. Main and framework scripts can be found in the /builder/ folder of the platform board folder.

- Source: PlatformIO documentation

5.5 Pinlayout Nordic nRF52-DK

The Pinlayout of the Nordic nRF52-DK does not have a default layout for arduino use. Depending on the framework you pins are assigned variables and names in the header files. To find out how the pins are setup on the Nordic developer board, I had to read the datasheet and introduction guide of the Nordic nRF52-DK.

5.6 Bluetooth Low Energie (BLE)

5.7 DFU

5.8 Tools during the internship

My supervisor has expressed the expectations he has of me. The basic knowledge of certain compiler and debugging tools. He listed a few new terms for me that I should look into.

- Ldd-tool - print shared object dependencies *link*
- ld - ld combines a number of object and archive files, relocates their data and ties up symbol references *link*
- nm-tool - Shows list of symbols of object files
- Hexdump - display file contents in hexadecimal, decimal, octal, or ascii *link*
- objcopy - copy and translate object files *link 1 link 2*
- objdump - display information from object files. *link*
- readelf: Executable and Linkable Format and it defines the structure for binaries, libraries, and core files. *link 1 link 2*
- Checksum:
  - md5sum - compute and check MD5 message digest *link*
  - cksum - checksum and count the bytes in a file *link*
- nrfconnect - This is a tool with build gui for developing the nRF52832 developer board
- nrfjprog - This tool is used to flash hexfiles to the nRF52832
- srec_cat *link link examples*

5.9 Binary File

A binary file is a file stored in binary format. A binary file is computer-readable but not human-readable. All executable programs are stored in binary files, as are most numeric data files. In contrast, text files are stored in a form (usually ASCII) that is human-readable.

- Source: Wikipedia

5.10 HEX File

A HEX file is program file with binary information commonly used for programming microcontrollers. A compiler converts the program's code into machine code or assembly and outputs it into a HEX file. The HEX file is then read by a programmer to write the machine code into a PROM or is transferred to the target system for loading and execution. A hex file is unreadable for a person. Only machines can read it. To read the data in a hex file, the file needs to be converted into another file type.

- Source: Wikipedia

5.11 .ARM.exidx

LLVM and the ARM ELF .ARM.exidx* section [2, Article on ELF for ARM]

5.12 Random-Access Memory (RAM)

RAM (Random-Access Memory) is a form of computer data storage that is used to store working data and machine code that can be read or changed in any order. And it allows data items to be read or written almost the same amount of time irrespective of the physical location of data inside the memory. Random access memory is volatile, i.e., it requires a steady flow of electricity to maintain its contents, and as soon as the power goes off, whatever data that was in the RAM is lost. It is commonly known as read/write memory, and is an integral part in computers and other devices like printers.

- Source: Wikipedia

5.13 Flash Memory

Flash memory is a derivative of the EEPROM memory. It is designed to make storing large amounts of data in a small space possible, allowing reading and writing in multiple memory locations with the same operation. This type of memory, is based on the use of semiconductors. In addition to being non – volatile and rewritable, it possesses almost all the features of RAM, along with the added advantage that it is non-volatile, meaning, what is stored in this type of flash memory, does not get deleted when you disconnect the device from the PC or the apparatus, unlike RAM. Flash memories are extremely important, especially in today's computer world, owing to its low power consumption, portability and size, as well as safety and efficiency; makes them ideal for supporting data and information created with digital cameras, smartphones, audio devices, among other gadgets. Even, they are quite resistant to any blow or fall, which represents a huge improvement over portable mass storage devices of previous generation.[7]

- Source: Wikipedia

5.14 Linkerscripts

Linkerscripts are are text files with the commands to make object files compiled by a compiler into executable programs. The linker puts the input files into a single output file. The object files have a list of sections. Sections in object files have names and sizes. Most sections also have an associated block of data, known as the section contents A section may be marked as loadable, which means that the contents should be loaded into memory when the output file is run. Sections with no contents may be allocatable, which means that an area in the memory should be set aside. Nothing in particular should be loaded there. In some cases this memory area should be zeroed out. A section which is neither loadable nor allocatable typically contains some sort of debugging information.

Every loadable or allocatable output section has two addresses. The first is the VMA (virtual memory address) and the second is the LMA (load memory address). VMA is the address the section will have when the output file is run. LMA is the address at which the section will be loaded.

Every object file also has a list of symbols. A symbol may be defined or undefined. Each symbol has a name, and each defined symbol has an address, among other information. [6]

5.15 Interrupt Handler

Interrupts are signals that go to the processor to indicate that there is an event that needs the processor's immediate attention. When an interrupt occurs the processor completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler. ISR tells the processor what to do when the interrupt occurs. The interrupts can be either hardware interrupts or software interrupts. Interrupt handlers vary based on what triggered the interrupt and the speed at which the interrupt handler completes its task.

Every interrupt has an interrupt service routine (ISR) or interrupt handler. When an interrupt occurs, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its interrupt service routine, ISR. [8]

5.16 SoC support in SoftDevice Handler

When making handlers for the firmware, the handlers need to be registered so that the softdevice knows what they are and where they are located. Registering the handler goes through using the SoftDevice Handler.

```
#define NRF_SDH_SOC_EVENT_OBSERVERS
        (
            _name,
            _prio,
            _handler,
            _context,
            _cnt
        )
```

Listing 1: Macro Registering Event Handler

Macro for registering an array of nrf_sdh_soc_evt_observer_t. Modules that want to be notified about SoC events must register the handler using this macro. Each observer's handler will be dispatched an event with its relative context from _context. This macro places the observer in a section named "sdh_soc_observers".

```
#define NRF_SDH_SOC_OBSERVER
        (
            _name,
            _prio,
            _handler,
            _context
        )
```

Listing 2: Macro Registering Handler

Macro for registering nrf_sdh_soc_evt_observer_t. Modules that want to be notified about SoC events must register the handler using this macro.
This macro places the observer in a section named "sdh_soc_observers". [3]

5.17 Sections in linkerscripts

The SECTIONS command tells the linker how to map input sections into output sections, and how to place the output sections in memory. This can also make the linker script easier to understand because you can use those commands at meaningful points in the layout of the output file. If a SECTIONS command is not used in the linker script, the linker will place each input section into an identically named output section in the order that the sections are first encountered in the input files. If all input sections are present in the first file, for example, the order of sections in the output file will match the order in the first input file. The first section will be at address zero. [5]

5.18 Function Pointers in C/C++

Function pointers are variables that are used to store a address of a function to later be called through that function pointer. It is useful so that not all functions need to be loaded in the main file. [1]

6 Phase Progression

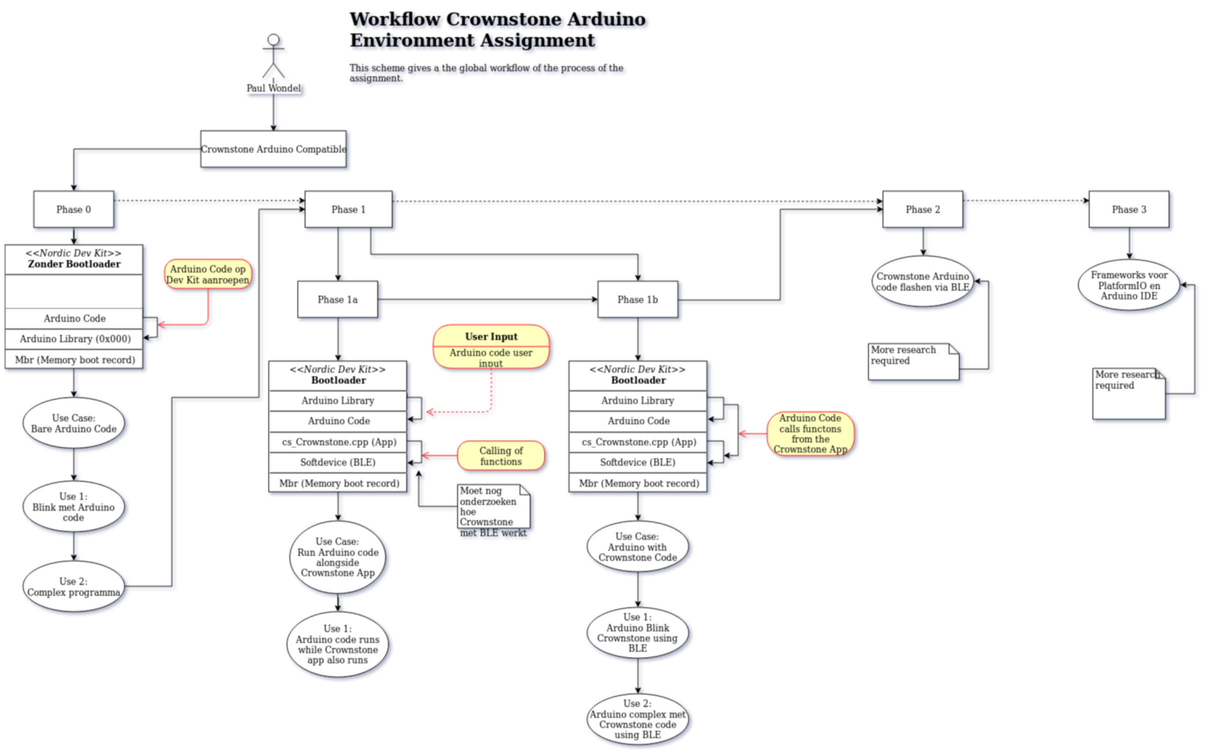

Workflow Crownstone Arduino Environment Assignment
This scheme gives a the global workflow of the process of the assignment.
Paul Wondel
Crownstone Arduino Compatible
Phase 0
Phase 1
Phase 2
Phase 3
<<Nordic Dev Kit>> Zonder Bootloader
Arduino Code
Arduino Library (0x000)
Mbr (Memory boot record)
Arduino Code op Dev Kit aanroepen
Use Case: Bare Arduino Code
Use 1: Blink met Arduino code
Use 2: Complex programma
Phase 1a
Phase 1b
<<Nordic Dev Kit>> Bootloader
Arduino Library
Arduino Code
cs_Crownstone.cpp (App)
Softdevice (BLE)
Mbr (Memory boot record)
User Input
Arduino code user input
Calling of functions
Moet nog onderzoeken hoe Crownstone met BLE werkt
Use Case: Run Arduino code alongside Crownstone App
Use 1: Arduino code runs while Crownstone app also runs
Arduino Code calls functons from the Crownstone App
Use Case: Arduino with Crownstone Code
Use 1: Arduino Blink Crownstone using BLE
Use 2: Arduino complex met Crownstone code using BLE
Crownstone Arduino code flashen via BLE
More research required
Frameworks voor PlatformIO en Arduino IDE
More research required

Figure 7: Phase Plan Workflow Scheme

6.1 Phase 0

The goal of phase 0 is to get arduino code to run on the Nordic nRF52-DK Developer kit without a bootloader and without the crownstone app. During the first attempt of connecting the I discovered that the dev kit has no way of flashing code on my windows 10 laptop. Windows 10 does not use the required drivers automatically. If Windows 10 is wished to be used then the drivers for the Nordic nRF52-DK Developer kit have to be installed manually. Therefore I switched from a Windows 10 to Ubuntu 18.4.1.

On the Linux system it is essential to have the right package for the J-Link tool installed. Not all Linux distributions can work with the package. The recommended Linux OS for the package is a Debian based OS such as Ubuntu & Linux Mint. I had Arch Linux installed which had difficulties with the package. It was not the right architecture for the package and the package was not available in the aur repository of Arch Linux. De official website of SEGGER has the right packages available for download. I downloaded the packages from the site and successfully installed the J-Link tool.

PlatformIO has to be installed on the operating system. This is required for programming the Nordic nRF52-DK with arduino code. It can be **PlatformIO Core (CLI)** or **PlatformIO IDE**. During this research I have made use of PlatformIO IDE on **VSCode**. In PlatformIO the right package is needed for the Nordic nRF52-DK board. PlatformIO already has the support for nRF52 Nordic boards. The platform files still need to be installed which can be done at the "platforms" tab in the PlatformIO IDE. After the required packages are installed, a new project needs to be created with the nRF52 as board and Arduino as framework.

Programming arduino code is the same as programming code for an Arduino Uno. The difference is that the pinlayout is not the same as with Arduino. The pins also have different names and are called upon in a different way.

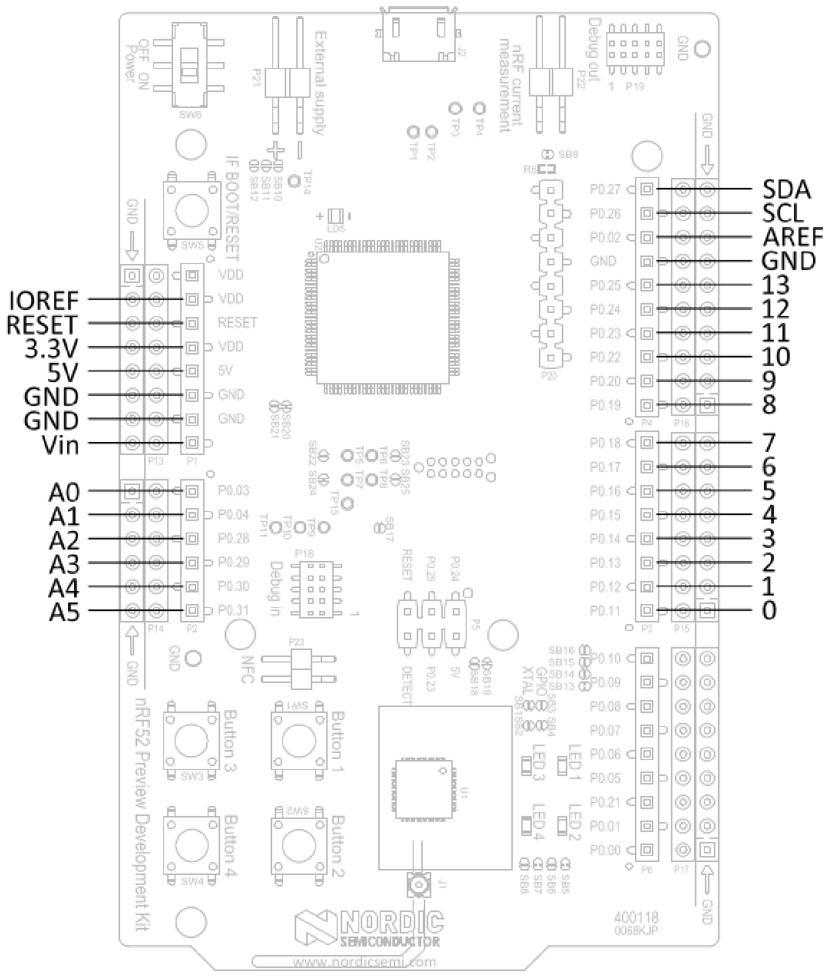
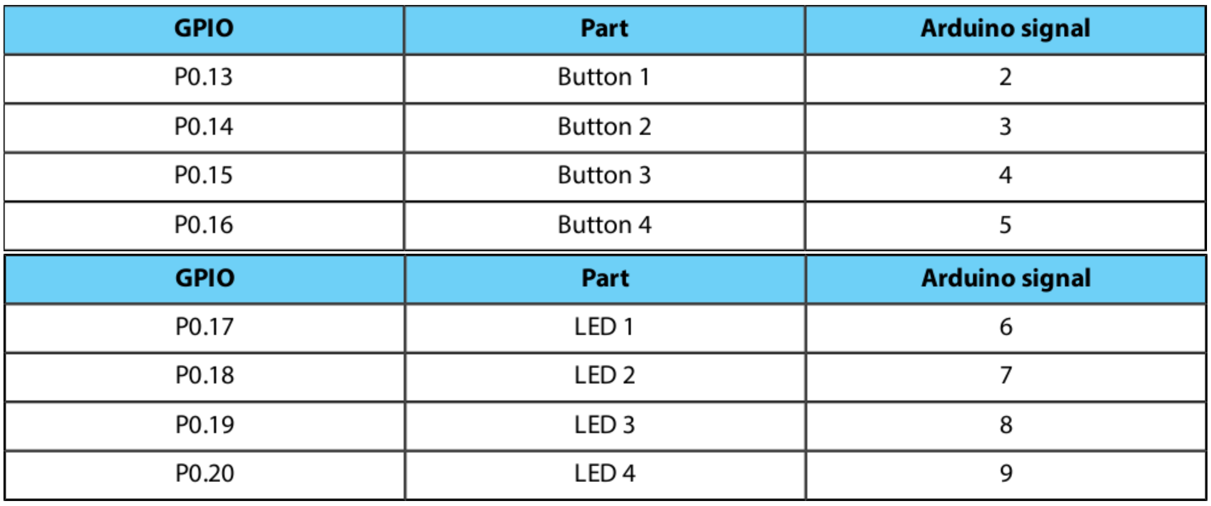


Figure 8: nRF52-DK Pinlayout

Source: nRF52 Preview Development Kit User Guide v1.2, Page 16

The analog pins are called upon by using them as {A0, A1, A2, A3, A4, A5} and the digital pins as {(0),(1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13)}. Some of the digital pins are already assigned to some buttons and LEDs that are on the developer board.

Here is a example of how the pin definition goes in arduino code:



GPIO	Part	Arduino signal
P0.13	Button 1	2
P0.14	Button 2	3
P0.15	Button 3	4
P0.16	Button 4	5
GPIO	**Part**	**Arduino signal**
P0.17	LED 1	6
P0.18	LED 2	7
P0.19	LED 3	8
P0.20	LED 4	9

Figure 9: Assigned Pins onboard LEDs & Buttons nRF52-DK

Source: RF52 Preview Development Kit User Guide v1.2, Page 17,18

```
#define ledPin (7) //dgital pin 7 on the board (P0.18)
#define irPin PIN_A1 //Analog pin A0 on the board (P0.05)
#define temPin PIN_AREF //AREF pin on the board (P0.02)
```

Listing 3: Definition PINS in arduino code

Use Case: Bare Arduino Code

In this use case the goal is to have bare arduino code run on the Nordic nRF52-DK. This is use case is devided into 2 use cases:

Use Case 1: Arduino Blink A simple arduino script to light up the onboard LEDs by using the onboard buttons.

Use Case 2: Auto Intensity Control of Power LED To test the control of the analog pins as well as the digital pins with pwm using sensors for input data.

The testplan for this phase can be found in the appendix (7).

6.2 Phase 1

In phase 1 we want the arduino code with arduino library to run simultaneously with the crownstone code. To achieve the goal more securely, this phase has been devided into phase 1a and phase 1b. In phase 1a the goal is to run the arduino code along side the crownstone code both independently. The crownstone should be running and should be able to call functions from the softdevice (BLE) regardless of the arduino code. And the arduino code should be able to call funtions of the crownstone.hex to use them in the arduino code.

6.2.1 Phase 1a

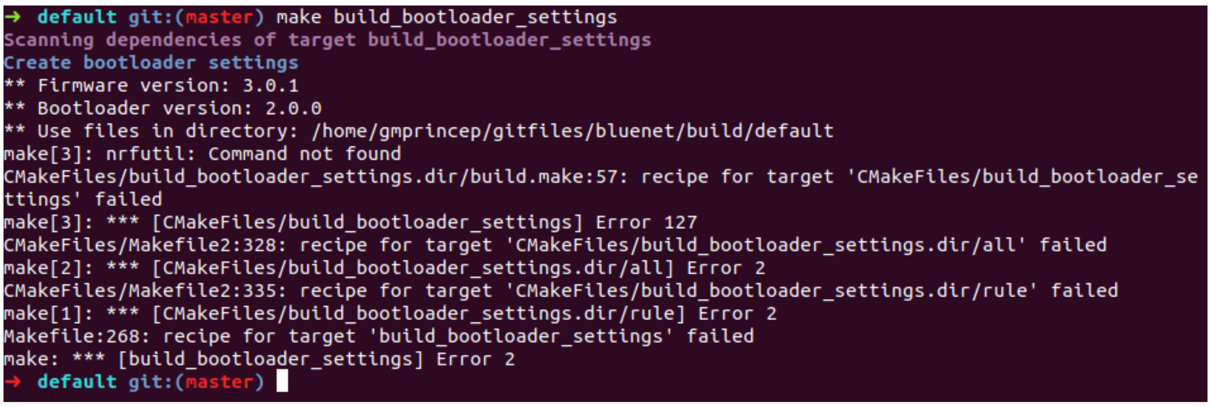
6.2.1.1 Installing Crownstone App (bluenet)

Installing the crownstone app unto the Nordic nRF52-DK requires a few programs te be pre-installed. These programs are assumed to be installed:

- `git` - sudo apt install git
- `pip3` - sudo apt install python3-pip
- `wget` - sudo apt install wget

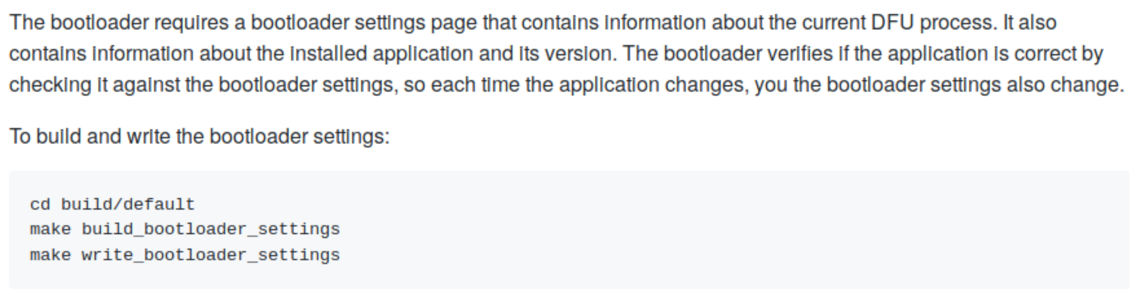
When the required programs are installed, the following install instructions can be followed on the repository of crownstone called bluenet: `https://github.com/crownstone/bluenet/blob/master/docs/INSTALL.md`.

During the installation I came across a complication that the tool nrfutil could not be found. The program pip is installing the tool into the python directory. This makes it so that the tool is only usable in a python environement. If this happens, make sure that you install the tool using `sudo pip3 install nrfutil`. If this doesn't help, search on the internet for a way to put the tool in the dir /usr/bin/local so that the operating system can use the `nrfutil`-command outside of the python environement. After successfully installing `nrfutil`, run the `make build_bootloader_settings` command again and follow the rest of the installation instructions.



```
➜ default git:(master) make build_bootloader_settings
Scanning dependencies of target build_bootloader_settings
Create bootloader settings
** Firmware version: 3.0.1
** Bootloader version: 2.0.0
** Use files in directory: /home/gmprincep/gitfiles/bluenet/build/default
make[3]: nrfutil: Command not found
CMakeFiles/build_bootloader_settings.dir/build.make:57: recipe for target 'CMakeFiles/build_bootloader_se
ttings' failed
make[3]: *** [CMakeFiles/build_bootloader_settings] Error 127
CMakeFiles/Makefile2:328: recipe for target 'CMakeFiles/build_bootloader_settings.dir/all' failed
make[2]: *** [CMakeFiles/build_bootloader_settings.dir/all] Error 2
CMakeFiles/Makefile2:335: recipe for target 'CMakeFiles/build_bootloader_settings.dir/rule' failed
make[1]: *** [CMakeFiles/build_bootloader_settings.dir/rule] Error 2
Makefile:268: recipe for target 'build_bootloader_settings' failed
make: *** [build_bootloader_settings] Error 2
➜ default git:(master)
```

Figure 10: The error that occurred.



The bootloader requires a bootloader settings page that contains information about the current DFU process. It also contains information about the installed application and its version. The bootloader verifies if the application is correct by checking it against the bootloader settings, so each time the application changes, you the bootloader settings also change.

To build and write the bootloader settings:

```
cd build/default
make build_bootloader_settings
make write_bootloader_settings
```

Figure 11: The step of the installation where the error occured.

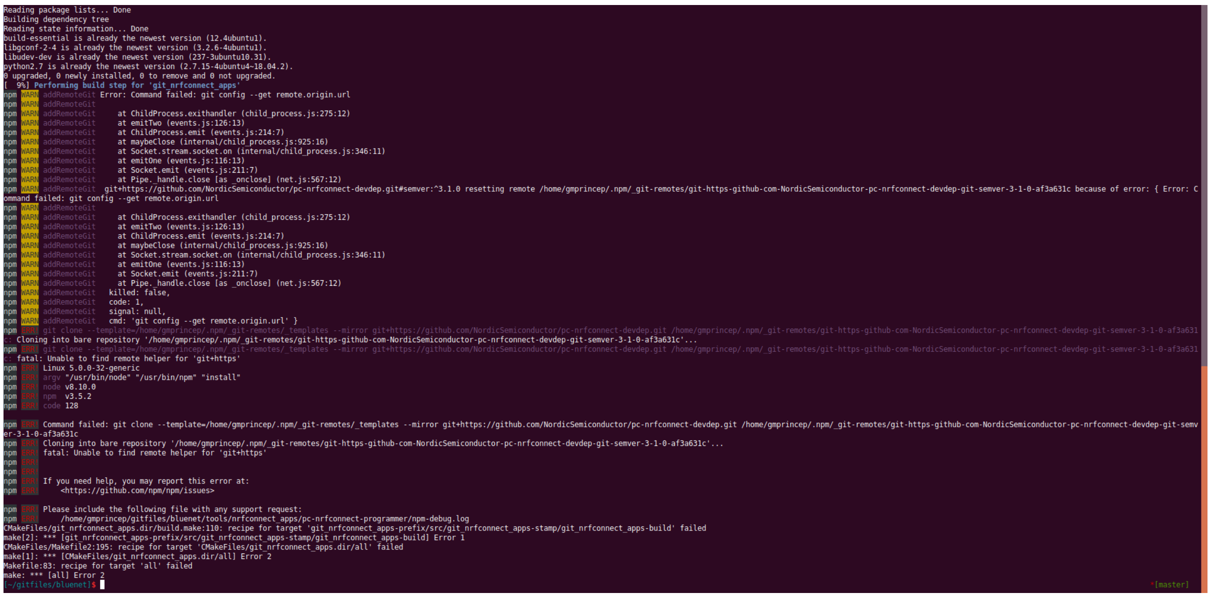
6.2.1.2 Flashing

The crownstone code is put into a hexfile which is that flashed to the nRF52 using the textttnrfjprog tool. In the bluenet repository there is a build directory in which you will find a file called CMakeList.txt. In this list you will find the commands that are used to flash the bootloader settings, the crownstone code itself and how to reset the nRF52. These are the commands:

```
    $ nrfjprog -f nrf52 --eraseall
    $ nrfjprog -f nrf52 --program softdevice_mainpart.hex --sectorerase
    $ nrfjprog -f nrf52 --program crownstone.hex --sectorerase
    $ nrfjprog -f nrf52 --program bootloader_settings.hex --sectorerase
    $ nrfjprog --reset
```

6.2.1.3 Memory Map

Nordic Semiconductor (The company that makes the Nordic nRF52-DK) has tools available for different uses. In this phase I wanted to know the current used memory registers. To get a visual of the realtime used memory registers I used the tool: `NRFCONNECT_PROGRAMMER`. I have installed the `nrfconnect_core` to get the tools. Within the menu of this tool I chose the programmer tool to be installed. I could not install the programmer seperately from the `nrfconnect_core` due to a npm package installation error.



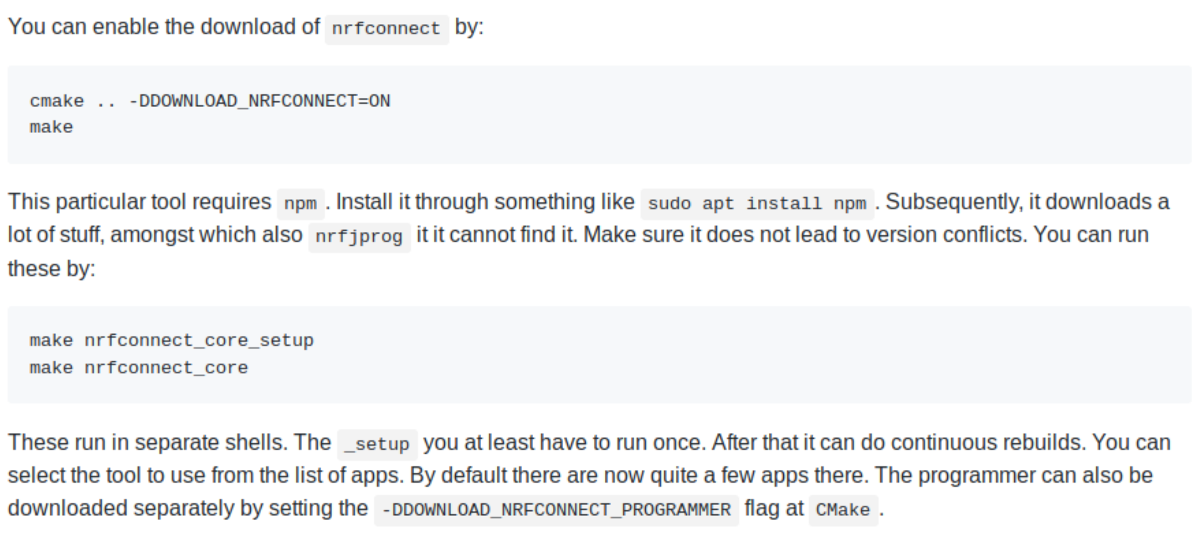
```
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
libgconf-2-4 is already the newest version (3.2.6-4ubuntu1).
libudev-dev is already the newest version (237-3ubuntu10.31).
python2.7 is already the newest version (2.7.15-4ubuntu4~18.04.2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[  9%] Performing build step for 'git_nrfconnect_apps'
npm WARN addRemoteGit Error: Command failed: git config --get remote.origin.url
npm WARN addRemoteGit
npm WARN addRemoteGit     at ChildProcess.exithandler (child_process.js:275:12)
npm WARN addRemoteGit     at emitTwo (events.js:126:13)
npm WARN addRemoteGit     at ChildProcess.emit (events.js:214:7)
npm WARN addRemoteGit     at maybeClose (internal/child_process.js:925:16)
npm WARN addRemoteGit     at Socket.stream.socket.on (internal/child_process.js:346:11)
npm WARN addRemoteGit     at emitOne (events.js:116:13)
npm WARN addRemoteGit     at Socket.emit (events.js:211:7)
npm WARN addRemoteGit     at Pipe._handle.close [as _onclose] (net.js:567:12)
npm WARN addRemoteGit  git+https://github.com/NordicSemiconductor/pc-nrfconnect-devdep.git#semver:^3.1.0 resetting remote /home/gmprincep/.npm/_git-remotes/git-https-github-com-NordicSemiconductor-pc-nrfconnect-devdep-git-semver-3-1-0-af3a631c because of error: { Error: C
ommand failed: git config --get remote.origin.url
npm WARN addRemoteGit
npm WARN addRemoteGit     at ChildProcess.exithandler (child_process.js:275:12)
npm WARN addRemoteGit     at emitTwo (events.js:126:13)
npm WARN addRemoteGit     at ChildProcess.emit (events.js:214:7)
npm WARN addRemoteGit     at maybeClose (internal/child_process.js:925:16)
npm WARN addRemoteGit     at Socket.stream.socket.on (internal/child_process.js:346:11)
npm WARN addRemoteGit     at emitOne (events.js:116:13)
npm WARN addRemoteGit     at Socket.emit (events.js:211:7)
npm WARN addRemoteGit     at Pipe._handle.close [as _onclose] (net.js:567:12)
npm WARN addRemoteGit   killed: false,
npm WARN addRemoteGit   code: 1,
npm WARN addRemoteGit   signal: null,
npm WARN addRemoteGit   cmd: 'git config --get remote.origin.url' }
npm ERR! git clone --template=/home/gmprincep/.npm/_git-remotes/_templates --mirror git+https://github.com/NordicSemiconductor/pc-nrfconnect-devdep.git /home/gmprincep/.npm/_git-remotes/git-https-github-com-NordicSemiconductor-pc-nrfconnect-devdep-git-semver-3-1-0-af3a631
c: Cloning into bare repository '/home/gmprincep/.npm/_git-remotes/git-https-github-com-NordicSemiconductor-pc-nrfconnect-devdep-git-semver-3-1-0-af3a631c'...
npm ERR! git clone --template=/home/gmprincep/.npm/_git-remotes/_templates --mirror git+https://github.com/NordicSemiconductor/pc-nrfconnect-devdep.git /home/gmprincep/.npm/_git-remotes/git-https-github-com-NordicSemiconductor-pc-nrfconnect-devdep-git-semver-3-1-0-af3a631
c: fatal: Unable to find remote helper for 'git+https'
npm ERR! Linux 5.0.0-32-generic
npm ERR! argv "/usr/bin/node" "/usr/bin/npm" "install"
npm ERR! node v8.10.0
npm ERR! npm  v3.5.2
npm ERR! code 128

npm ERR! Command failed: git clone --template=/home/gmprincep/.npm/_git-remotes/_templates --mirror git+https://github.com/NordicSemiconductor/pc-nrfconnect-devdep.git /home/gmprincep/.npm/_git-remotes/git-https-github-com-NordicSemiconductor-pc-nrfconnect-devdep-git-semv
er-3-1-0-af3a631c
npm ERR! Cloning into bare repository '/home/gmprincep/.npm/_git-remotes/git-https-github-com-NordicSemiconductor-pc-nrfconnect-devdep-git-semver-3-1-0-af3a631c'...
npm ERR! fatal: Unable to find remote helper for 'git+https'
npm ERR!
npm ERR!
npm ERR! If you need help, you may report this error at:
npm ERR!     <https://github.com/npm/npm/issues>

npm ERR! Please include the following file with any support request:
npm ERR!     /home/gmprincep/gitfiles/bluenet/tools/nrfconnect_apps/pc-nrfconnect-programmer/npm-debug.log
CMakeFiles/git_nrfconnect_apps.dir/build.make:110: recipe for target 'git_nrfconnect_apps-prefix/src/git_nrfconnect_apps-stamp/git_nrfconnect_apps-build' failed
make[2]: *** [git_nrfconnect_apps-prefix/src/git_nrfconnect_apps-stamp/git_nrfconnect_apps-build] Error 1
CMakeFiles/Makefile2:195: recipe for target 'CMakeFiles/git_nrfconnect_apps.dir/all' failed
make[1]: *** [CMakeFiles/git_nrfconnect_apps.dir/all] Error 2
Makefile:83: recipe for target 'all' failed
make: *** [all] Error 2
[~/gitfiles/bluenet]$                                                                                      *[master]
```

Figure 12: NRFCONNECT_PROGRAMMER Download Error

The installation guide can be found on the git repository of Crownstone: git@github.com:crownstone/bluenet. Make sure that npm installed before following the instructions.



You can enable the download of `nrfconnect` by:

```
cmake .. -DDOWNLOAD_NRFCONNECT=ON
make
```

This particular tool requires `npm`. Install it through something like `sudo apt install npm`. Subsequently, it downloads a lot of stuff, amongst which also `nrfjprog` it it cannot find it. Make sure it does not lead to version conflicts. You can run these by:

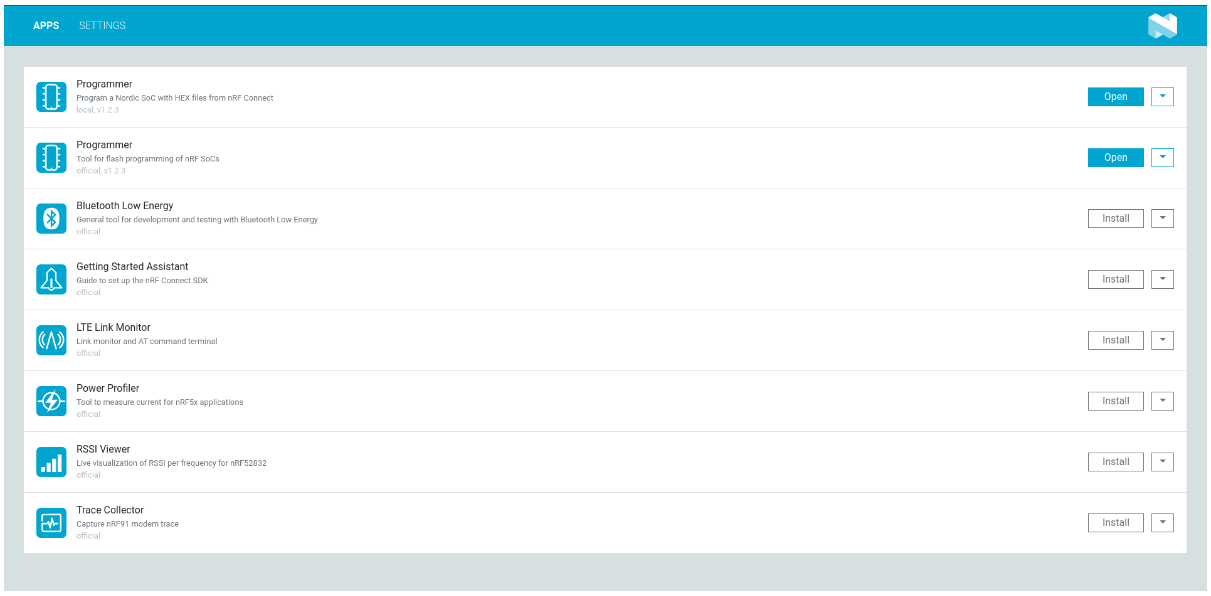
```
make nrfconnect_core_setup
make nrfconnect_core
```

These run in separate shells. The `_setup` you at least have to run once. After that it can do continuous rebuilds. You can select the tool to use from the list of apps. By default there are now quite a few apps there. The programmer can also be downloaded separately by setting the `-DDOWNLOAD_NRFCONNECT_PROGRAMMER` flag at `CMake`.

Figure 13: NRFCONNECT Install Guide

Source: *https://github.com/crownstone/bluenet/blob/master/docs/BUILD_SYSTEM.md*

After installing the nrfconnect_core you will need to run the `nrfconnect_programmer`. To start the tool, run the command `make nrfconnect_core` in the bluenet directory. The `nrfconnect_core` gui tool looks like this.



Programmer
Program a Nordic SoC with HEX files from nRF Connect
local, v1.2.3

Programmer
Tool for flash programming of nRF SoCs
official, v1.2.3

Bluetooth Low Energy
General tool for development and testing with Bluetooth Low Energy
official

Getting Started Assistant
Guide to set up the nRF Connect SDK
official

LTE Link Monitor
Link monitor and AT command terminal
official

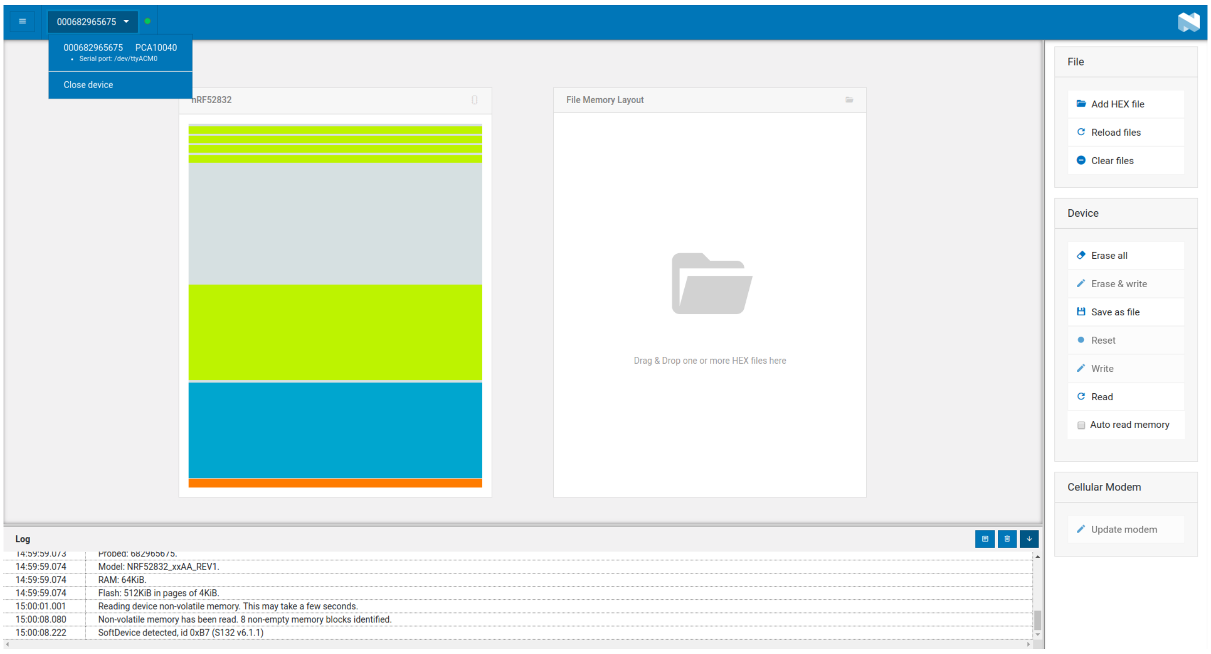
Power Profiler
Tool to measure current for nRF5x applications
official

RSSI Viewer
Live visualization of RSSI per frequency for nRF52832
official

Trace Collector
Capture nRF91 modem trace
official

Figure 14: NRFCONNECT_CORE GUI

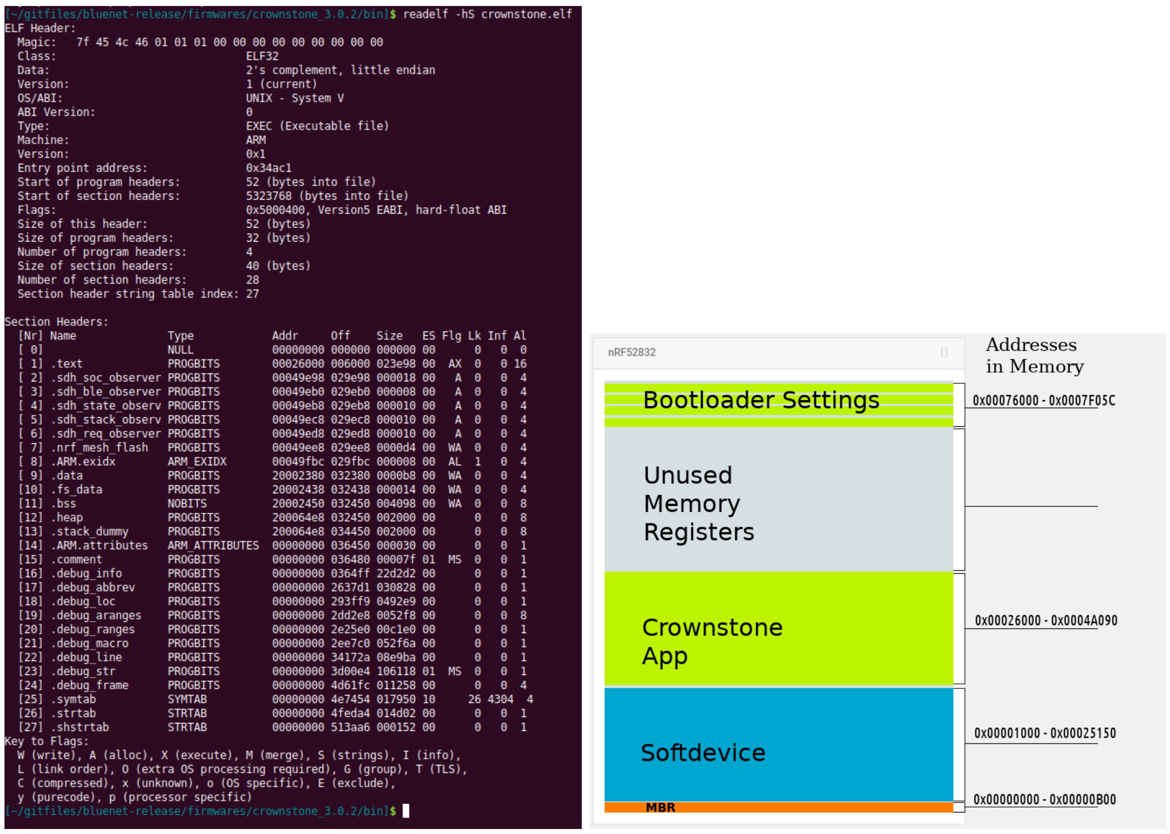
With the `nrfconnect_programmer` tool the memory map of the current device can be displayed in realtime. It also displays the addresses the files are loaded unto. The programmer tool gui looks like this.



```
14:59:59.073   Probed: 682965675.
14:59:59.074   Model: NRF52832_xxAA_REV1.
14:59:59.074   RAM: 64KiB.
14:59:59.074   Flash: 512KiB in pages of 4KiB.
15:00:01.001   Reading device non-volatile memory. This may take a few seconds.
15:00:08.080   Non-volatile memory has been read. 8 non-empty memory blocks identified.
15:00:08.222   SoftDevice detected, id 0xB7 (S132 v6.1.1)
```

Figure 15: NRFCONNECT_PROGRAMMER GUI

After displaying the memory map the addresses in the memory had to be found. The programmer tool has the option to show the start address of the hex files. I placed the hex files of the crownstone firmware along with the bootloader settings in the upload section of the tool to display the sizes of the files with their start addresses. This way I could see their addresses in the memory. These files are available in a different git repository that can be downloaded from: git@github.com:crownstone/bluenet-release. The files that are revelevant here are the: crownstone.hex, bootloadersettings.hex, softdevice_mainpart.hex. For the certainty of having the right start addresses of the files, I inspected the files seperately in commandline. In the .elf files the start address of the program is located. To read a .elf file, you use the command `readelf [option] [inputfile.elf]`. To get the addresses I used the command `readelf -Sh crownstone.elf`. I have applied this to each of the files and came up with an estimation of their sizes.



```
[~/gitfiles/bluenet-release/firmwares/crownstone_3.0.2/bin]$ readelf -hS crownstone.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x34ac1
  Start of program headers:          52 (bytes into file)
  Start of section headers:          5323768 (bytes into file)
  Flags:                             0x5000400, Version5 EABI, hard-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         4
  Size of section headers:           40 (bytes)
  Number of section headers:         28
  Section header string table index: 27

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00026000 006000 023e98 00  AX  0   0 16
  [ 2] .sdh_soc_observer PROGBITS        00049e98 029e98 000018 00   A  0   0  4
  [ 3] .sdh_ble_observer PROGBITS        00049eb0 029eb0 000008 00   A  0   0  4
  [ 4] .sdh_state_observ PROGBITS        00049eb8 029eb8 000010 00   A  0   0  4
  [ 5] .sdh_stack_observ PROGBITS        00049ec8 029ec8 000010 00   A  0   0  4
  [ 6] .sdh_req_observer PROGBITS        00049ed8 029ed8 000010 00   A  0   0  4
  [ 7] .nrf_mesh_flash   PROGBITS        00049ee8 029ee8 0000d4 00  WA  0   0  4
  [ 8] .ARM.exidx        ARM_EXIDX       00049fbc 029fbc 000008 00  AL  1   0  4
  [ 9] .data             PROGBITS        20002380 032380 0000b8 00  WA  0   0  4
  [10] .fs_data          PROGBITS        20002438 032438 000014 00  WA  0   0  4
  [11] .bss              NOBITS          20002450 032450 004098 00  WA  0   0  8
  [12] .heap             PROGBITS        200064e8 032450 002000 00      0   0  8
  [13] .stack_dummy      PROGBITS        200064e8 034450 002000 00      0   0  8
  [14] .ARM.attributes   ARM_ATTRIBUTES  00000000 036450 000030 00      0   0  1
  [15] .comment          PROGBITS        00000000 036480 00007f 01  MS  0   0  1
  [16] .debug_info       PROGBITS        00000000 0364ff 22d2d2 00      0   0  1
  [17] .debug_abbrev     PROGBITS        00000000 2637d1 030828 00      0   0  1
  [18] .debug_loc        PROGBITS        00000000 293ff9 0492e9 00      0   0  1
  [19] .debug_aranges    PROGBITS        00000000 2dd2e8 0052f8 00      0   0  8
  [20] .debug_ranges     PROGBITS        00000000 2e25e0 00c1e0 00      0   0  1
  [21] .debug_macro      PROGBITS        00000000 2ee7c0 052f6a 00      0   0  1
  [22] .debug_line       PROGBITS        00000000 34172a 08e9ba 00      0   0  1
  [23] .debug_str        PROGBITS        00000000 3d00e4 106118 01  MS  0   0  1
  [24] .debug_frame      PROGBITS        00000000 4d61fc 011258 00      0   0  4
  [25] .symtab           SYMTAB          00000000 4e7454 017950 10     26 4304  4
  [26] .strtab           STRTAB          00000000 4feda4 014d02 00      0   0  1
  [27] .shstrtab         STRTAB          00000000 513aa6 000152 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  y (purecode), p (processor specific)
[~/gitfiles/bluenet-release/firmwares/crownstone_3.0.2/bin]$
```

Figure 16: Addresses of the .hex files in the memory registers on the nRF52

6.2.1.4 Arduino HEX-file

When compiling arduino code in platformio, the .hex files are stored in a directory. This directory is hidden and only used for the .hex file to be stored in and to be flashed from. Because I needed the .hex and .elf files of the arduino code I had to get them from /home/$USER/Documents/PlatformIO/Projects/PROJECT_NAME/.pio/build/nrf52dk/. In there the files firmware.hex and firmware.elf are located, which contains the arduino.cpp converted into assembly code for the chip to be read.

6.2.1.5 Flashing Arduino HEX-file

PlatformIO uses the same tools that bluenet from crownstone uses to compile, build & flash to the nRF52-DK. The tools are:

- `nrfjprog` - For writing files to nrf52
- `jlink` - For connecting to nrf52
- `nrfutil` - A Python package that includes the nrfutil utility and the nordicsemi library
- `gcc-arm-none-eabi` - For compiling and building the .elf and .hex files

The start address of the firmware.hex is 0x00000000.

When trying to flash the arduino file, I need to know if the file will any complications with the addresses. By running the command `readelf -e firmware.elf` I could read the start address of the program which indeed happened to be 0x00000000. This comes in conflict with the softdevice of the crownstone that already has that address reserved for itself.

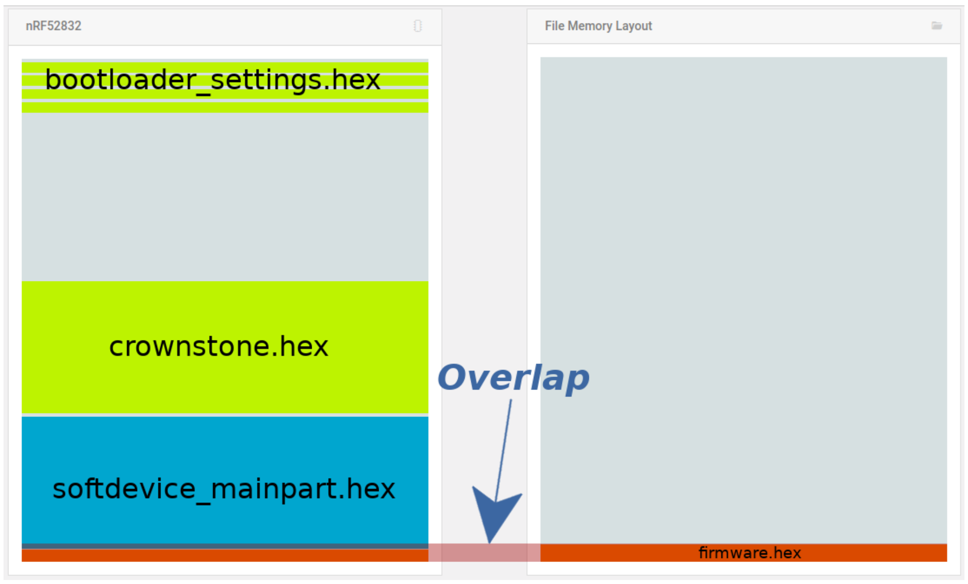


Figure 17: Crownstone Code VS Arduino Code

The solution is to change the address of the `firmware.hex` to a different address. To edit a hex file its .elf file needs to be edited and then converted to .hex. To write to an .elf file I used the arm-none-eabi-objcopy tool. By running the command:

```
arm-none-eabi-objcopy --change-addresses 0x00026000 -I elf32-little firmware.elf -O
elf32-little test.elf
```

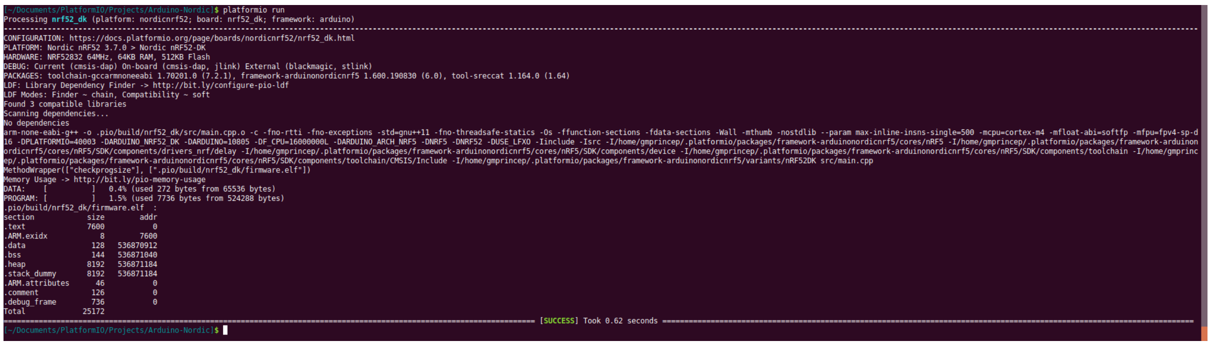
I was able to change all the addresses of the sections within the firmware.elf and called the output file test.elf. After changing the address in test.elf I had to convert the file into a .hex file. To do that I ran the command: `arm-none-eabi-objcopy -O ihex test.elf test.hex` and this created the output file test.hex. I flashed the test.hex along with the softdevice_mainpart.hex and bootloader_settings.hex using the following commands:

```
$ nrfjprog -f nrf52 --eraseall
$ nrfjprog -f nrf52 --program softdevice_mainpart.hex --sectorerase
$ nrfjprog -f nrf52 --program test.hex --sectorerase
$ nrfjprog -f nrf52 --program bootloader-settings.hex --sectorerase
$ nrfjprog --reset
```

After the successfull flashing of files, the code did not execute.

6.2.1.6 Re-compiling the Arduino Hex-file in PlatformIO

After discussing my research with my supervisor, we came to the conclusion that the approach tochanging the start address of the `firmware.hex` does not work and is unnecessary. We came up with a different approach of changing the address. Instead of changing the address of the already compiled hexile, I need to delve into the compiling process of platformio where it explicitly uses the commando or script and change the start address there. To find out where in the process PlatformIO does the compiling I had to refer to the documentation of PlatformIO. The command for compiling code in platformio is `platformio run`. I needed to know what platformio does during this command, so running the command with verbose would show me at least some of the processes that are run in the background.



```
[~/Documents/PlatformIO/Projects/Arduino-Nordic]$ platformio run
Processing nrf52_dk (platform: nordicnrf52; board: nrf52_dk; framework: arduino)
----------------------------------------------------------------------------------------------------------------------------------------------------------------
CONFIGURATION: https://docs.platformio.org/page/boards/nordicnrf52/nrf52_dk.html
PLATFORM: Nordic nRF52 3.7.0 > Nordic nRF52-DK
HARDWARE: NRF52832 64MHz, 64KB RAM, 512KB Flash
DEBUG: Current (cmsis-dap) On-board (cmsis-dap, jlink) External (blackmagic, stlink)
PACKAGES: toolchain-gccarmnoneeabi 1.70201.0 (7.2.1), framework-arduinonordicnrf5 1.600.190830 (6.0), tool-sreccat 1.164.0 (1.64)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 3 compatible libraries
Scanning dependencies...
No dependencies
arm-none-eabi-g++ -o .pio/build/nrf52_dk/src/main.cpp.o -c -fno-rtti -fno-exceptions -std=gnu++11 -fno-threadsafe-statics -Os -ffunction-sections -fdata-sections -Wall -mthumb -nostdlib --param max-inline-insns-single=500 -mcpu=cortex-m4 -mfloat-abi=softfp -mfpu=fpv4-sp-d
16 -DPLATFORMIO=40003 -DARDUINO_NRF52_DK -DARDUINO=10805 -DF_CPU=16000000L -DARDUINO_ARCH_NRF5 -DNRF5 -DNRF52 -DUSE_LFXO -Iinclude -Isrc -I/home/gmprincep/.platformio/packages/framework-arduinonordicnrf5/cores/nRF5 -I/home/gmprincep/.platformio/packages/framework-arduinon
ordicnrf5/cores/nRF5/SDK/components/drivers_nrf/delay -I/home/gmprincep/.platformio/packages/framework-arduinonordicnrf5/cores/nRF5/SDK/components/device -I/home/gmprincep/.platformio/packages/framework-arduinonordicnrf5/cores/nRF5/SDK/components/toolchain -I/home/gmprinc
ep/.platformio/packages/framework-arduinonordicnrf5/cores/nRF5/SDK/components/toolchain/CMSIS/Include -I/home/gmprincep/.platformio/packages/framework-arduinonordicnrf5/variants/nRF52DK src/main.cpp
MethodWrapper(["checkprogsize"], [".pio/build/nrf52_dk/firmware.elf"])
Memory Usage -> http://bit.ly/pio-memory-usage
DATA:    [          ]   0.4% (used 272 bytes from 65536 bytes)
PROGRAM: [          ]   1.5% (used 7736 bytes from 524288 bytes)
.pio/build/nrf52_dk/firmware.elf  :
section              size         addr
.text                7600            0
.ARM.exidx              8         7600
.data                 128    536870912
.bss                  144    536871040
.heap                8192    536871184
.stack_dummy         8192    536871184
.ARM.attributes        46            0
.comment              126            0
.debug_frame          736            0
Total               25172
============================================================================== [SUCCESS] Took 0.62 seconds ==============================================================================
[~/Documents/PlatformIO/Projects/Arduino-Nordic]$ 
```

Figure 18: Verbose output platformio run command

In the output that `platformio run` gave, the command `arm-none-eabi-gc++` is shown to be called upon. In the command .platformio/packages/framework-arduinonordicnrf5/cores/nRF5/SDK/components/toolchain is set as input in this command for compiling. When I went into the folder I found the linkerscripts that are used to build the firmware.hex file. The linkerscripts are located in the gcc folder and the script with the start address is named nrf52_xxaa.ld. The contents of the file should display as:

```
/* Linker script to configure memory regions. */

SEARCH_DIR(.)
GROUP(-lgcc -lc -lnosys)

MEMORY
{
  FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x80000
  RAM (rwx) :  ORIGIN = 0x20000000, LENGTH = 0x10000
}


INCLUDE "nrf52_common.ld"
```

Listing 4: Arduino linkerscript

The FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x80000 is the start address for where the firmware.hex is going to be placed in the memory of the nRF52-DK. The reason why platformio puts the address at 0x000000000 is because that is where the CPU starts. I changed the address to 0x00026000 and then uploaded the hex file along with the softdevice_mainpart.hex and bootloader_settings.hex. After uploading the files and resetting the device, the arduino code was being executed by the cpu. The arduino code is the same code that is made in chapter 6.1.

6.2.1.7 Running Arduino code alongside Crownstone (both independently) The crownstone application is the only one that is being executed by the cpu. The arduino code and crowstone code function as two different applications. There is no way for one application to just call a function or an object in another application. There is also no function in the crownstone to refer cpu to the arduino code to execute it. There needs to be a reference in the crownstone code to the address of the arduino code. For that we need the use of a handler to refer the cpu to the address of the arduino code. More info about handlers can be found in chapter 5.15.

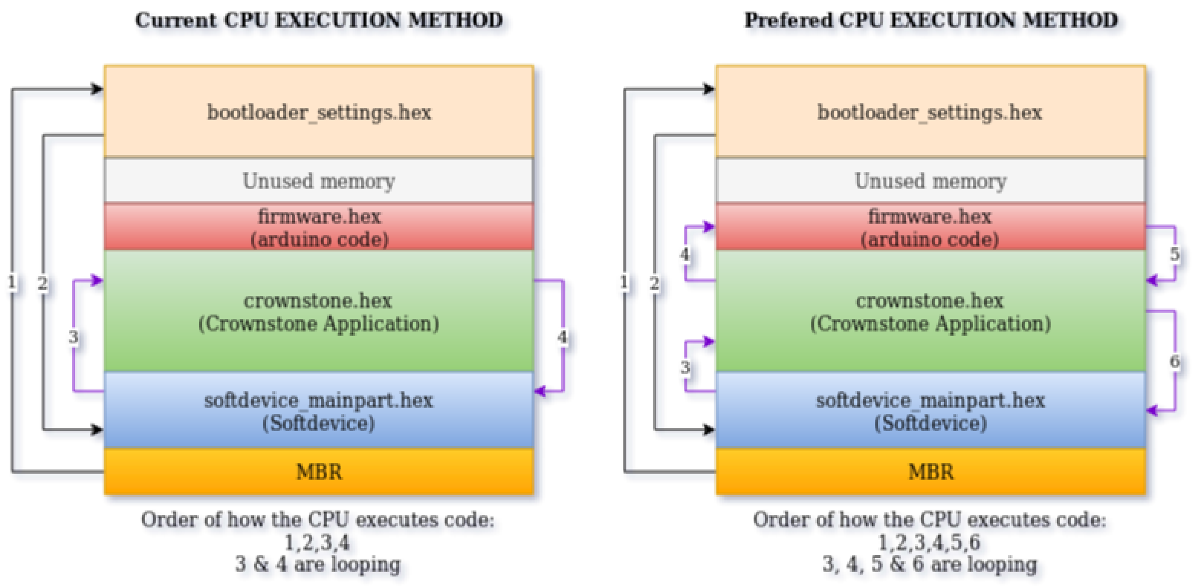
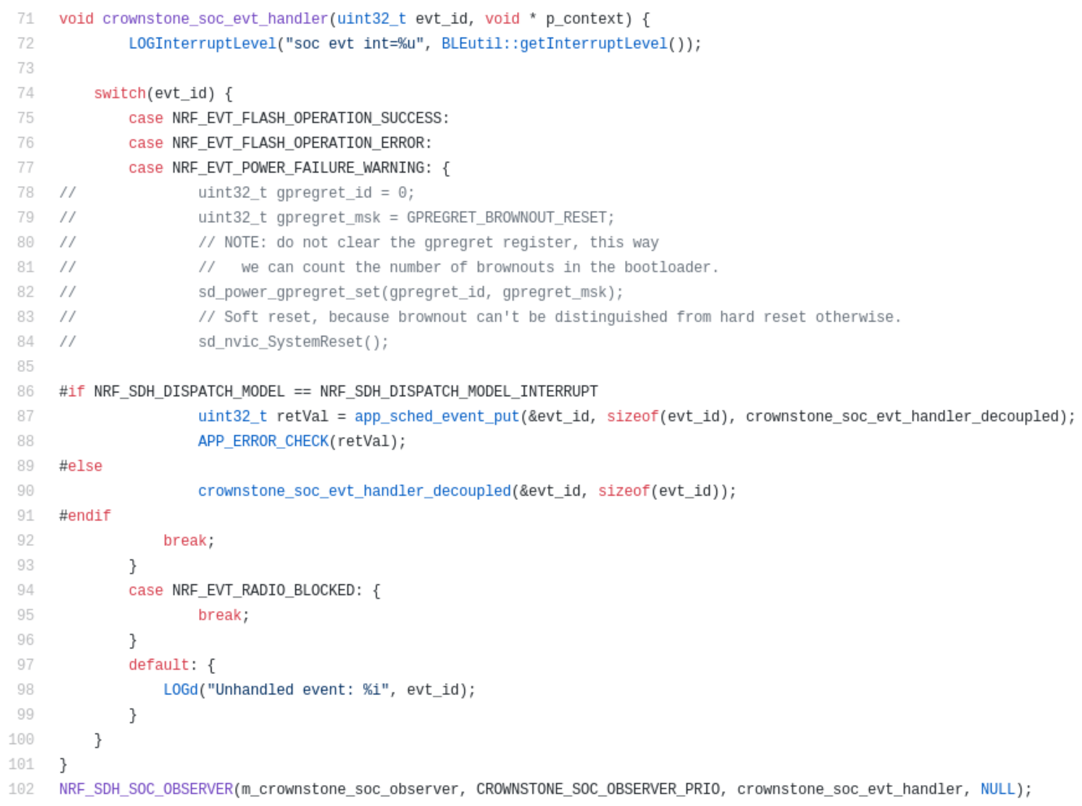

Current CPU EXECUTION METHOD
bootloader_settings.hex
Unused memory
firmware.hex (arduino code)
crownstone.hex (Crownstone Application)
softdevice_mainpart.hex (Softdevice)
MBR
1
2
3
4
Order of how the CPU executes code: 1,2,3,4 3 & 4 are looping
Prefered CPU EXECUTION METHOD
bootloader_settings.hex
Unused memory
firmware.hex (arduino code)
crownstone.hex (Crownstone Application)
softdevice_mainpart.hex (Softdevice)
MBR
1
2
3
4
5
6
Order of how the CPU executes code: 1,2,3,4,5,6 3, 4, 5 & 6 are looping

Figure 19: CPU Execution of code in the Memory Map Scheme

6.2.1.8 Handler & linkerscript

A handler for the arduino code needs to be created in the crownstone code. The all the handlers that the crownstone uses are in bluenet/source/src/common/cs_Handlers.cpp file. In the cs_Handler.cpp file I took one of the handlers as an example to create a handler for the ar-



```
 71  void crownstone_soc_evt_handler(uint32_t evt_id, void * p_context) {
 72          LOGInterruptLevel("soc evt int=%u", BLEutil::getInterruptLevel());
 73
 74      switch(evt_id) {
 75          case NRF_EVT_FLASH_OPERATION_SUCCESS:
 76          case NRF_EVT_FLASH_OPERATION_ERROR:
 77          case NRF_EVT_POWER_FAILURE_WARNING: {
 78  //              uint32_t gpregret_id = 0;
 79  //              uint32_t gpregret_msk = GPREGRET_BROWNOUT_RESET;
 80  //              // NOTE: do not clear the gpregret register, this way
 81  //              //   we can count the number of brownouts in the bootloader.
 82  //              sd_power_gpregret_set(gpregret_id, gpregret_msk);
 83  //              // Soft reset, because brownout can't be distinguished from hard reset otherwise.
 84  //              sd_nvic_SystemReset();
 85
 86  #if NRF_SDH_DISPATCH_MODEL == NRF_SDH_DISPATCH_MODEL_INTERRUPT
 87                  uint32_t retVal = app_sched_event_put(&evt_id, sizeof(evt_id), crownstone_soc_evt_handler_decoupled);
 88                  APP_ERROR_CHECK(retVal);
 89  #else
 90                  crownstone_soc_evt_handler_decoupled(&evt_id, sizeof(evt_id));
 91  #endif
 92              break;
 93          }
 94          case NRF_EVT_RADIO_BLOCKED: {
 95                  break;
 96          }
 97          default: {
 98              LOGd("Unhandled event: %i", evt_id);
 99          }
100      }
101  }
102  NRF_SDH_SOC_OBSERVER(m_crownstone_soc_observer, CROWNSTONE_SOC_OBSERVER_PRIO, crownstone_soc_evt_handler, NULL);
```

Figure 20: Crownstone event handler in cs_Handler.cpp

duino code. On line 71 in the cs_Handlers.cpp the `void crownstone_soc_evt_handler` has 2 inputs, `uint32_t evt_id & void * p_context`. The `uint_t evt_id` value goes into a switch case method which gives back a certain message back. On line 182 there is a `NRF_SDH_SOC_OBSERVER` function that is called upon. In chapter 5.16 there is more info about the SoC observer handler.

In the bluenet/source/include/third/nrf/ dir of the bluenet repository is a linkerscript named `generic_gcc_nrf52.ld`. This file containts the information to be compiled with for the `crownstone.elf` and `crownstone.hex`. In the linkerscript the start address for the memory register is specified. Sections are also defined in the linkerscript.

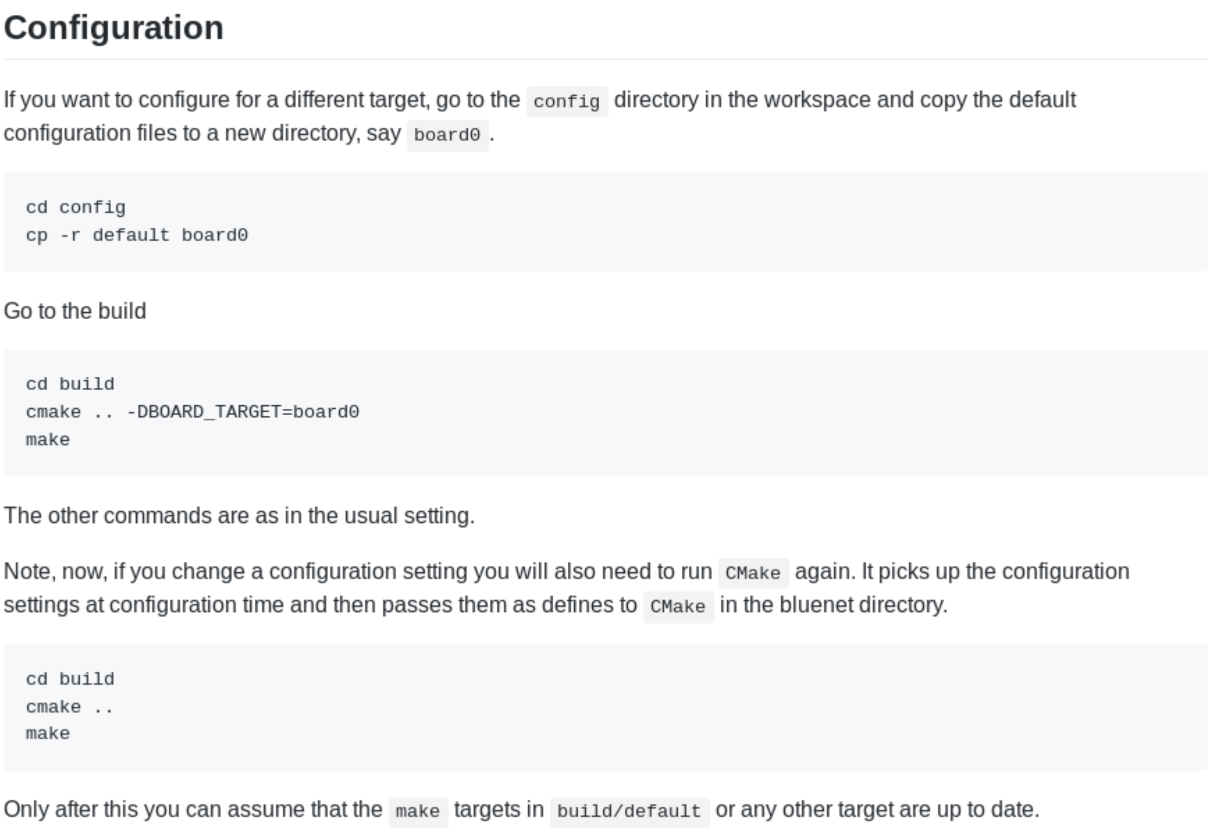
```
SECTIONS
{
  .mem_section_dummy_rom :
  {
  }
  .sdh_ant_observers :
  {
    PROVIDE(__start_sdh_ant_observers = .);
    KEEP(*(SORT(.sdh_ant_observers*)))
    PROVIDE(__stop_sdh_ant_observers = .);
  } > FLASH
  .sdh_soc_observers :
  {
    PROVIDE(__start_sdh_soc_observers = .);
    KEEP(*(SORT(.sdh_soc_observers*)))
    PROVIDE(__stop_sdh_soc_observers = .);
  } > FLASH
  .sdh_ble_observers :
  {
    PROVIDE(__start_sdh_ble_observers = .);
    KEEP(*(SORT(.sdh_ble_observers*)))
    PROVIDE(__stop_sdh_ble_observers = .);
  } > FLASH
  .sdh_state_observers :
  {
    PROVIDE(__start_sdh_state_observers = .);
    KEEP(*(SORT(.sdh_state_observers*)))
    PROVIDE(__stop_sdh_state_observers = .);
  } > FLASH
  .sdh_stack_observers :
  {
    PROVIDE(__start_sdh_stack_observers = .);
    KEEP(*(SORT(.sdh_stack_observers*)))
    PROVIDE(__stop_sdh_stack_observers = .);
  } > FLASH
  .sdh_req_observers :
  {
    PROVIDE(__start_sdh_req_observers = .);
    KEEP(*(SORT(.sdh_req_observers*)))
    PROVIDE(__stop_sdh_req_observers = .);
  } > FLASH
```

Listing 5: Crownstone linkerscript generic_gcc_nrf52.ld

First there needs to be a section for the function of the event handler. The sections makes sure that there is enough memory saved for the functions.

```
    .arduino_handler :
    {
     PROVIDE(__start_arduino_handler = .);
     KEEP(*(SORT(.arduino_handler*)))
     PROVIDE(__stop_arduino_handler = .);
    } > FLASH
```

After creating a new section for the arduino handler I called `.arduino_handler`, I need to see if it compiles the way it is supposed to. For compiling bluenet code for this "test & prototype phase", I configured the code to be used for a different board instead of having the default bluenet code. By following the instructions on https://github.com/crownstone/bluenet/blob/master/docs/INSTALL.md I could change the settings for a new board.



Configuration

If you want to configure for a different target, go to the `config` directory in the workspace and copy the default configuration files to a new directory, say `board0`.

```
cd config
cp -r default board0
```

Go to the build

```
cd build
cmake .. -DBOARD_TARGET=board0
make
```

The other commands are as in the usual setting.

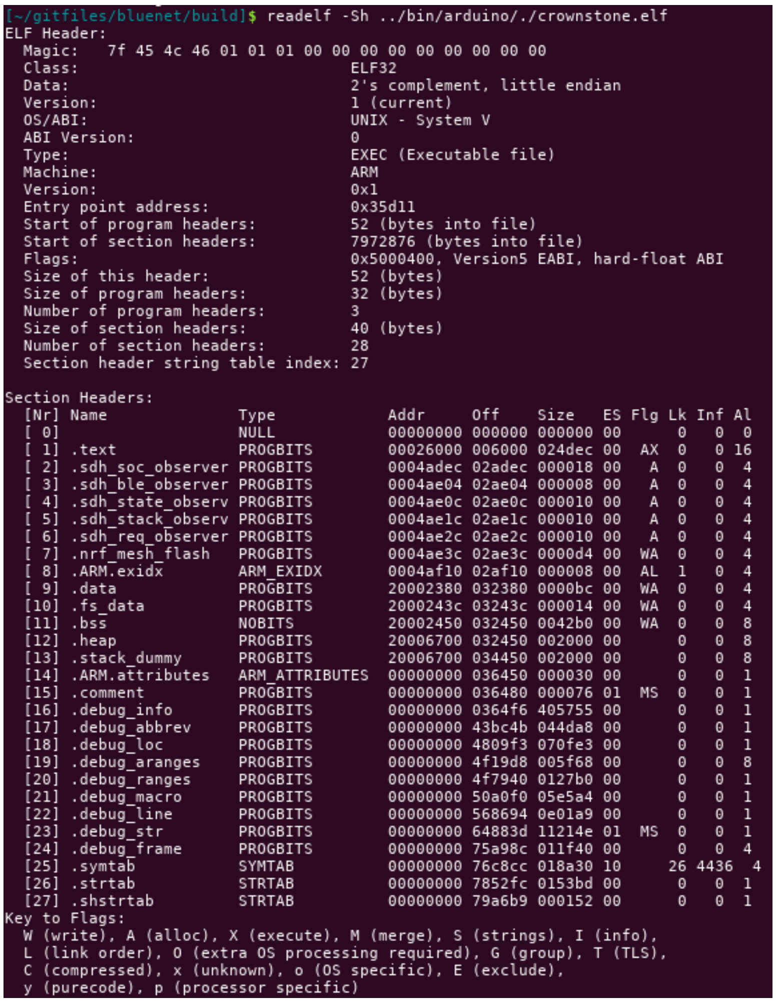
Note, now, if you change a configuration setting you will also need to run `CMake` again. It picks up the configuration settings at configuration time and then passes them as defines to `CMake` in the bluenet directory.

```
cd build
cmake ..
make
```

Only after this you can assume that the `make` targets in `build/default` or any other target are up to date.

Figure 21: Configuration instructions on github

I named this build **arduino**. After following the instructions I compiled the bluenet code with the edited version of the linkerscript. The compiled files of bluenet code of the build are stored in the bin directory of bluenet. The default build uses the bluenet/bin/default where the `bootloader.elf`, `crownstone.bin` `crownstone.elf` and `crownstone.hex` are located. So the same goes for my new **arduino** build; bluenet/bin/arduino.



```
[~/gitfiles/bluenet/build]$ readelf -Sh ../bin/arduino/./crownstone.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x35d11
  Start of program headers:          52 (bytes into file)
  Start of section headers:          7972876 (bytes into file)
  Flags:                             0x5000400, Version5 EABI, hard-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         28
  Section header string table index: 27

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00026000 006000 024dec 00  AX  0   0 16
  [ 2] .sdh_soc_observer PROGBITS        0004adec 02adec 000018 00   A  0   0  4
  [ 3] .sdh_ble_observer PROGBITS        0004ae04 02ae04 000008 00   A  0   0  4
  [ 4] .sdh_state_observ PROGBITS        0004ae0c 02ae0c 000010 00   A  0   0  4
  [ 5] .sdh_stack_observ PROGBITS        0004ae1c 02ae1c 000010 00   A  0   0  4
  [ 6] .sdh_req_observer PROGBITS        0004ae2c 02ae2c 000010 00   A  0   0  4
  [ 7] .nrf_mesh_flash   PROGBITS        0004ae3c 02ae3c 0000d4 00  WA  0   0  4
  [ 8] .ARM.exidx        ARM_EXIDX       0004af10 02af10 000008 00  AL  1   0  4
  [ 9] .data             PROGBITS        20002380 032380 0000bc 00  WA  0   0  4
  [10] .fs_data          PROGBITS        2000243c 03243c 000014 00  WA  0   0  4
  [11] .bss              NOBITS          20002450 032450 0042b0 00  WA  0   0  8
  [12] .heap             PROGBITS        20006700 032450 002000 00      0   0  8
  [13] .stack_dummy      PROGBITS        20006700 034450 002000 00      0   0  8
  [14] .ARM.attributes   ARM_ATTRIBUTES  00000000 036450 000030 00      0   0  1
  [15] .comment          PROGBITS        00000000 036480 000076 01  MS  0   0  1
  [16] .debug_info       PROGBITS        00000000 0364f6 405755 00      0   0  1
  [17] .debug_abbrev     PROGBITS        00000000 43bc4b 044da8 00      0   0  1
  [18] .debug_loc        PROGBITS        00000000 4809f3 070fe3 00      0   0  1
  [19] .debug_aranges    PROGBITS        00000000 4f19d8 005f68 00      0   0  8
  [20] .debug_ranges     PROGBITS        00000000 4f7940 0127b0 00      0   0  1
  [21] .debug_macro      PROGBITS        00000000 50a0f0 05e5a4 00      0   0  1
  [22] .debug_line       PROGBITS        00000000 568694 0e01a9 00      0   0  1
  [23] .debug_str        PROGBITS        00000000 64883d 11214e 01  MS  0   0  1
  [24] .debug_frame      PROGBITS        00000000 75a98c 011f40 00      0   0  4
  [25] .symtab           SYMTAB          00000000 76c8cc 018a30 10     26 4436  4
  [26] .strtab           STRTAB          00000000 7852fc 0153bd 00      0   0  1
  [27] .shstrtab         STRTAB          00000000 79a6b9 000152 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  y (purecode), p (processor specific)
```

Figure 22: Section Headers in crownstone.elf

I ran the readelf -Sh ../bin/arduino/./crownstone.elf to see the section headers. The section .arduino_handler is not displayed after compiling. This means that just editing the linkerscript generic_gcc_nrf52.ld is not enough.

I ran the command DEBUG=1 make in the build folder to see what the output was while compiling and got this error for the output:

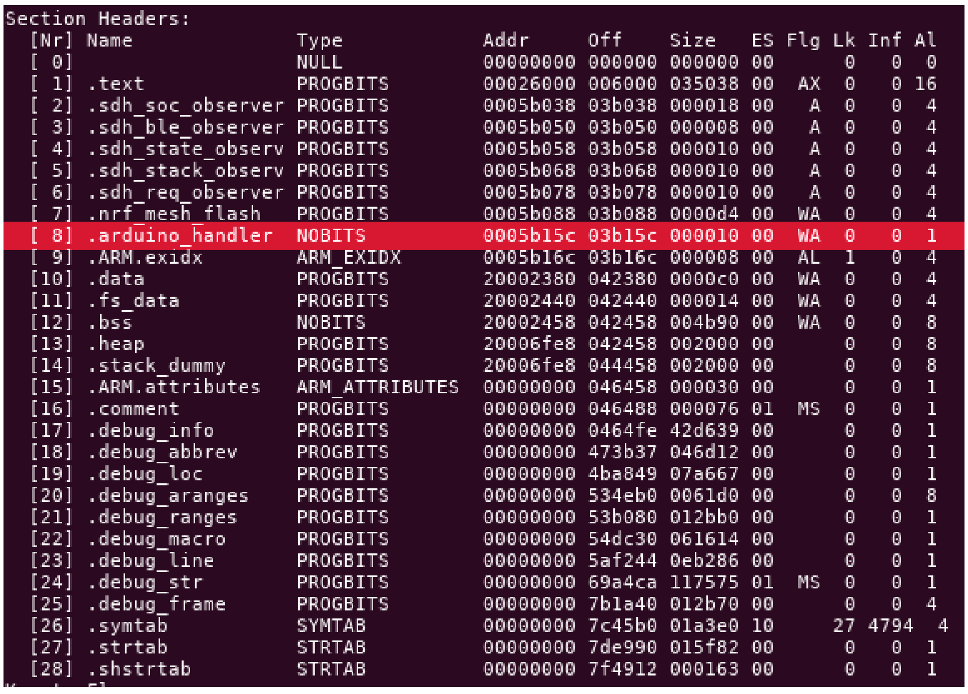
```
  /home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/bin/../lib/gcc/arm-none-
  eabi/8.3.1/../../../../arm-none-eabi/bin/ld: crownstone section ‘.arduino_handler’
  will not fit in region ‘FLASH’
/home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/bin/../lib/gcc/arm-none-eabi
  /8.3.1/../../../../arm-none-eabi/bin/ld: region FLASH overflowed with .data and user
   data
/home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/bin/../lib/gcc/arm-none-eabi
  /8.3.1/../../../../arm-none-eabi/bin/ld: region ‘FLASH’ overflowed by 8202724 bytes
/home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/bin/../lib/gcc/arm-none-eabi
  /8.3.1/../../../../arm-none-eabi/bin/ld: .arduino_handler has both ordered [‘.ARM.
  exidx’ in /home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/bin/../lib/gcc/
  arm-none-eabi/8.3.1/../../../../arm-none-eabi/lib/thumb/v7e-m+fp/hard/crt0.o] and
```

```
    unordered [‘.ARM.extab’ in /home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/
    bin/../lib/gcc/arm-none-eabi/8.3.1/../../../../arm-none-eabi/lib/thumb/v7e-m+fp/hard
    /crt0.o] sections
  /home/gmprincep/gitfiles/bluenet/tools/gcc_arm_none_eabi/bin/../lib/gcc/arm-none-eabi
    /8.3.1/../../../../arm-none-eabi/bin/ld: final link failed: bad value
```

The error explains that the `.arduino_handler` section doesn't fit in the FLASH. Because the size of `.arduino_handler` isn't defined, I tried assigning a size to the arduino handler section. I found a solution on stackoverflow [9] that I applied to the this situation. I changed the information in the `.arduino_handler` section into:

```
.arduino_handler :
{
  __arduino_handler_size = 0x04;
  __arduino_handler_start = .;
  . = . + __arduino_handler_size;
  __arduino_handler_end = .;
} > FLASH
```

After running the command `DEBUG=1 make` without recieving errors, I listed the sections headers again to see if the new section is indeed created by the compiler. This is the output of the `readelf -Sh ../bin/arduino/./crownstone.elf`:



```
Section Headers:
  [Nr] Name              Type             Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL             00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS         00026000 006000 035038 00  AX  0   0 16
  [ 2] .sdh_soc_observer PROGBITS         0005b038 03b038 000018 00   A  0   0  4
  [ 3] .sdh_ble_observer PROGBITS         0005b050 03b050 000008 00   A  0   0  4
  [ 4] .sdh_state_observ PROGBITS         0005b058 03b058 000010 00   A  0   0  4
  [ 5] .sdh_stack_observ PROGBITS         0005b068 03b068 000010 00   A  0   0  4
  [ 6] .sdh_req_observer PROGBITS         0005b078 03b078 000010 00   A  0   0  4
  [ 7] .nrf_mesh_flash   PROGBITS         0005b088 03b088 0000d4 00  WA  0   0  4
  [ 8] .arduino_handler  NOBITS           0005b15c 03b15c 000010 00  WA  0   0  1
  [ 9] .ARM.exidx        ARM_EXIDX        0005b16c 03b16c 000008 00  AL  1   0  4
  [10] .data             PROGBITS         20002380 042380 0000c0 00  WA  0   0  4
  [11] .fs_data          PROGBITS         20002440 042440 000014 00  WA  0   0  4
  [12] .bss              NOBITS           20002458 042458 004b90 00  WA  0   0  8
  [13] .heap             PROGBITS         20006fe8 042458 002000 00      0   0  8
  [14] .stack_dummy      PROGBITS         20006fe8 044458 002000 00      0   0  8
  [15] .ARM.attributes   ARM_ATTRIBUTES   00000000 046458 000030 00      0   0  1
  [16] .comment          PROGBITS         00000000 046488 000076 01  MS  0   0  1
  [17] .debug_info       PROGBITS         00000000 0464fe 42d639 00      0   0  1
  [18] .debug_abbrev     PROGBITS         00000000 473b37 046d12 00      0   0  1
  [19] .debug_loc        PROGBITS         00000000 4ba849 07a667 00      0   0  1
  [20] .debug_aranges    PROGBITS         00000000 534eb0 0061d0 00      0   0  8
  [21] .debug_ranges     PROGBITS         00000000 53b080 012bb0 00      0   0  1
  [22] .debug_macro      PROGBITS         00000000 54dc30 061614 00      0   0  1
  [23] .debug_line       PROGBITS         00000000 5af244 0eb286 00      0   0  1
  [24] .debug_str        PROGBITS         00000000 69a4ca 117575 01  MS  0   0  1
  [25] .debug_frame      PROGBITS         00000000 7b1a40 012b70 00      0   0  4
  [26] .symtab           SYMTAB           00000000 7c45b0 01a3e0 10     27 4794  4
  [27] .strtab           STRTAB           00000000 7de990 015f82 00      0   0  1
  [28] .shstrtab         STRTAB           00000000 7f4912 000163 00      0   0  1
```

Figure 23: `.arduino_handler` in Section Headers in `crownstone.elf`

After talking with my supervisor my next goal was clear. I need to print the address of the .arduino_handler. I need to print the address of the contents of the address of the .arduino_handler. So the I need to print the address of the __arduino_handler_start and __arduino_handler_end. Since the __arduino_handler_size = 0x10 which is 16bits, there should be 16 address value slots that are reserved for the .arduino_handler section. Then I need to print the address of the arduino handler function. After printing the address I need to put the address of the arduino handler function into the one of the values slots of the .arduino_handler.

To list the sections of crownstone.elf run the command readelf -Sh [file] and to list the symbols with their addresses run the command nm [file] | grep [symbol].

```
SECTIONS
{
  ..
  ..
  .arduino_handler :
  {
  __arduino_handler_size = 0x10;
  __arduino_handler_start = .;
  . = . + __arduino_handler_size;
  __arduino_handler_end = .;
  } > FLASH
} INSERT AFTER .text;
```

Listing 6: Code added to the bluenet linker file

Before applying the method to the .arduino_handler section I was advised to write a c program to test out the theory first so that I can see if the method works

```
    #include <stdio.h>
    void callback() {
        printf("This is the arduino callback function! \n");
    }

    void functionAdress(){
    }

    int main(int argc, const char* argv[]) {

        int var1;
        int *p;

        p = &var1;

        printf("Address of the variable: %p\n", &var1); //Prints the address of the
    variable
        printf("Value of the target of the pointer: %d\n",*p); //Prints the value of the
    pointer

        var1 = 20;

        printf("Value of the target of the pointer: %d\n",*p); //Prints the value of the
    pointer

        callback();
        printf("Address of callback function: %p\n", &callback);


        return 0;
    }
```

Listing 7: main.c test file

The command for compiling and running I used are:

- gcc main.c
- ./a.out

The result I got for this was:

```
Address of the variable: 0x7fff250f00ac
Value of the target of the pointer: 21894
Value of the target of the pointer: 20
This is the arduino callback function!
Address of callback function: 0x55866e8266fa
```

Listing 8: Results of test run

Next I created a function pointer. First I created a variable to store the address for callback().

```
static size_t func;
```

To declare a function pointer I needed to add the following line into main():

```
size_t (*func)(int,int);
```

When I compiled the main.c with the new lines I recieved this error:

```
main.c: In function 'main':
main.c:30:9: warning: assignment makes integer from pointer without a cast [-Wint-
conversion]
func = &callback;
```

The types of callback() and the variable func are not the same. Therefore the pointer cannot be used for the function. callback() does not have ask for any input data, so the (int, int) in the size_t (*func)(int,int); needs to be removed. The line should be size_t (*func)();. main.c: In function 'main': On a site: Function Pointers in C and C++ are function pointers explained. With the help of this article I was able to fix the compiler error and have the program executed successfully. I first tested it with another function before applying it to the initial function that I was working on.

```
int functionAddress(int x){
  printf("%d\n", x);
}

int (*foo)(int);
foo = &functionAddress;
(*foo)(2);
```

After these codelines were successfully executed, I modified it for callback().

```
static size_t func; //variable to store function address into

void (*func)(); // declare function pointer
func = &callback; // assign address to pointer
(*func)(); // call function indirectly
```

Before assigning the address of callback() the function pointer needs to be declared first or else the function pointer will function as a variable. After declaring the function pointer I assigned the address to the function pointer. The code should look like this now.

```
#include <stdio.h>

static size_t func; //variable to store function address into

void callback() {
    printf("This is the arduino callback function! \n");
}

int functionAddress(int x){
    printf("%d\n", x);
}


int main(int argc, const char* argv[]) {

    int var1;
    int *p;

    p = &var1;

    printf("Address of the variable: %p\n", &var1); //Prints the address of the
variable
    printf("Value of the target of the pointer: %d\n",*p); //Prints the value of the
 pointer

    var1 = 20;

    printf("Value of the target of the pointer: %d\n",*p); //Prints the value of the
 pointer

    callback();
    printf("Address of callback function: %p\n", &callback);

    //int *ptr_func = &callback;
    //func = ptr_func;
    //func = &callback;

    int (*foo)(int);
    foo = &functionAddress;
    (*foo)(2);

    //typedef void (*func)(); //declaring a function pointer

    void (*func)(); // declare function pointer
    func = &callback; // assign address to pointer
    (*func)(); // call function indirectly
    printf("func: %p\n", func); // I need to print the type of the &callback

    return 0;
}
```

Listing 9: Updated main.c

```
Address of the variable: 0x7ffda7f3618c
Value of the target of the pointer: 0
Value of the target of the pointer: 20
This is the arduino callback function!
Address of callback function: 0x5635c4a4a6fa
2
This is the arduino callback function!
func: 0x5635c4a4a6fa
```

Listing 10: Output main.c

6.2.1.9 arduino_handler section: This method was dropped due to the time that is not available to reach end results

The idea is to have a section in the memory where the jumper function is stored. This section needs to be used by both the arduino binary and crownstone binary. First I need to print the addresses of the arduino_handler section. By running the command nm -a crownstone.elf | grep arduino I get as output:

```
0005b15c b .arduino_handler
0005b160 B __arduino_handler_end
0005b160 B __arduino_handler_size
0005b15c B __arduino_handler_start
```

Using the ld-tool I can link object files so that the compiler knows which libraries to use. I want to try using the linkerscript to just create a section in the executable file a.out. I am using the code in main.c.

I made a linkerscript called arduino_handler.ld:

```
OUTPUT_FORMAT(elf32-i386)
ENTRY(main)
OUTPUT(Test)

SECTIONS
{
    .text : { *(.text) }
    . = ALIGN(0x10);
    .data : { *(.data) }
    . = ALIGN(0x10);
    .bss : { *(.bss)}
    . = ALIGN(0X10);
    .arduino_handler :
    {
    __arduino_handler_size = 0x10;
    __arduino_handler_start = .;
    . = . + __arduino_handler_size;
```

```
        __arduino_handler_end = .;
        }
    }
}
```

Listing 11: "arduino_handler.ld"

I also created a Makefile for this.

```
    #This keeps the echo for every command silent that is executed in this makefile
    ifndef VERBOSE
    .SILENT:
    endif

    IDIR = /include #Input directory
    CC = gcc #Compiler
    DEPS = stdio.h main.h #Dependency
    OBJ = main.o #Object files
    CFLAGS = -I. # Compiler flags

    #This is the command for suppressing the echo per command.
    #.SILENT: main link_main link_test readelf_main readelf_test symbols_test clean
    compile

    %.o: %.c $(DEPS)
        $(CC) -c -o $@ $< $(CFLAGS)

    main: $(OBJ)
        $(CC) -o $@ $^ $(CFLAGS)

    compile:
        $(CC) -c *.c #-I$(IDIR)

    link_main:
        ld main.o -e main -T arduino_handler.ld

    link_test:
        ld -T arduino_handler.ld add.o

    readelf_test:
        readelf -Sh Test

    readelf_main:
        readelf -Sh main

    symbols_test:
        nm Test

    clean:
        rm -f *.o Test main
```

Listing 12: Makefile

When I ran the command ld -T arduino_handler.ld main.c I recieved an error.

```
main.o: In function ‘callback’:
main.c:(.text+0xc): undefined reference to ‘puts’
main.o: In function ‘functionAddress’:
main.c:(.text+0x30): undefined reference to ‘printf’
main.o: In function ‘main’:
main.c:(.text+0x71): undefined reference to ‘printf’
main.c:(.text+0x8a): undefined reference to ‘printf’
main.c:(.text+0xaa): undefined reference to ‘printf’
main.c:(.text+0xcc): undefined reference to ‘printf’
main.o:main.c:(.text+0x110): more undefined references to ‘printf’ follow
main.o: In function ‘main’:
main.c:(.text+0x129): undefined reference to ‘__stack_chk_fail’
```

The ‘printf()‘ is not known because the library that contains the ‘printf()‘ is not linked to the ‘main.c‘. When compiling the ‘main.c‘ with gcc, gcc already uses the library but because it is not linked to ‘main.c‘ the ‘ld‘ command cannot find the library. Because I couldn’t find a way to link the library, I made very basic (simple) c program.

```c
#include <stdio.h>

int main(void){
    int x = 5;
    x++;
}
```

Listing 13: add.c source code

Compiling and linking these files is successful and I get as output:

```
ELF Header:
Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF32
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              EXEC (Executable file)
Machine:                           Intel 80386
Version:                           0x1
Entry point address:               0x0
Start of program headers:          52 (bytes into file)
Start of section headers:          2097564 (bytes into file)
Flags:                             0x0
Size of this header:               52 (bytes)
Size of program headers:           32 (bytes)
Number of program headers:         2
Size of section headers:           40 (bytes)
Number of section headers:         8
Section header string table index: 7

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 200000 000016 00  AX  0   0  1
[ 2] .eh_frame         PROGBITS        00000018 200018 000038 00   A  0   0  8
[ 3] .arduino_handler  NOBITS          00000050 200050 000010 00  WA  0   0  1
[ 4] .comment          PROGBITS        00000000 200050 00002c 01  MS  0   0  1
[ 5] .symtab           SYMTAB          00000000 20007c 000090 10      6   5  4
[ 6] .strtab           STRTAB          00000000 20010c 00004b 00      0   0  1
[ 7] .shstrtab         STRTAB          00000000 200157 000045 00      0   0  1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
p (processor specific)
```

Listing 14: Output readelf

In the output it is displayed that .arduino_handler is indeed created.

After talking with my supervisor, he explained that it wasn't necessarly needed that the section is being made by the linker file, but that it had more to do with 'gcc' vs 'ld'. 'gcc' uses 'ld' under the hood during compiling, so it automatically gives the standard libraries to link with. 'ld' by itself doesn't do that, so I need to find the library and link it so that it can be linked with 'arduino_handler.ld' and 'main.o'. 'gcc -T' is also a way to use the linkerscript during compiling the 'main.c'. So I executed the command 'gcc -T arduino_handler.ld add.c' and recieved as output:

```
/usr/bin/ld: cannot find -lgcc_s
/usr/bin/ld: cannot find -lgcc_s
collect2: error: ld returned 1 exit status
```

I had Anne take a look at the code and he tried to find a solution. Because the system is a x86-64 the memory harder to be accessed than a arm system. Because this will take too much time, he advised me to just drop this method and go straight to the device to do it on.

6.2.1.10 Call a function from a different binary I found a post on stackoverflow that proved a way of adding a section to the file without editing or creating a linkerscript. I made my own version using the info on the post.

```
#include<stdio.h>
#include<iostream>

#include<cs_ArduinoHandler.h>

// Initialize the variable in the arduino_handler section with address,
static void (*jumpToCallback)() __attribute__((section("arduino_handler"))); //
Variable in the section that functions as a pointer
extern unsigned char __start_arduino_handler;
extern unsigned char __stop_arduino_handler;

// Pointers to start and stop
static unsigned char * p_arduino_handler_start = &__start_arduino_handler;
static unsigned char * p_arduino_handler_end = &__stop_arduino_handler;

static void (*func)(); //variable to store function address into
```

```

    // function to be put in section
    void callback(void){
        printf("Go to arduino firmware\n"); // will be replaced with a addresss or pointer to address
    }
    //void (*jumpToCallback)();
    //void (*func)();
    /*
    //Created a new function to use the variables because the bluenet compiler gives errors if they are not explicitly used
    void variableCheck(unsigned char jumpvar, size_t funcvar){

        if(jumpvar == funcvar){
            printf("Jump To Callback Address: %u\n", jumpvar);
        }
        else{
            printf("jumpToCallback: %u | func: %u\n",jumpvar,funcvar);
        }
    }
    */
    // Can't call it main() since cs_Crownstone.cpp already has the main. This has to be called by the main.
    int jumpMain(){
        printf("section `arduino_handler` starts at %p, ends at %p, and the 1st byte is %c\n",
                p_arduino_handler_start, p_arduino_handler_end, p_arduino_handler_start[0]);

    //  void (*func)(); // declare function pointer
        func = &callback; // assign address to pointer
        (*func)(); // call function indirectly

    //  void (*jumpToCallback)();
        jumpToCallback = func;
        (*jumpToCallback)();
    //  variableCheck(jumpToCallback, func); //function to use the variables to satisfy the compiler

        if(jumpToCallback == func){
            printf("Jump To Callback Address: %p\n", jumpToCallback);
        }
        else{
            printf("jumpToCallback: %p | func: %p\n",jumpToCallback,func);
        }

        return 0;
    }
```

Listing 15: cs_ArduinoHandler.cpp

```
    //static unsigned char jumpToCallback __attribute__((section("arduino_handler")));
    //static size_t func;

    void callback(void);

    //void variableCheck(unsigned char, size_t);

    int jumpMain();
```

Listing 16: cs_ArduinoHandler.h

According to the post there is no need to edit the linker-script to create a new section. You can simple just do it in the source code. In the source code I initialized a pointer in the section using the '__attribute__((section("arduino_handler")))'. I created a variable in the section by typing 'static unsignedchar pling __attribute__((section("arduino_handler"))) = '!';'. I could also give it it's own byte size by filling a number where the ' **!** ' stands. After creating the variable in the in the '.arduino_handler' section I declared a function pointer called 'jumpToCallback' that points to the address of the function 'callback()'.

My superviser informed me that creating a section in source code might work but there are no guaranties. I also need to check if the section created by the linkerscript is the same section in the source code, since crownstone does work with the linkerscript sections.

I placed the the 'cs_ArduinoHandler.cpp' in 'bluenet/source/src/' and the 'cs_ArduinoHandler.h' in 'bluenet/source/src/' directory. I ran the command 'DEBUG=1 make' to compile. After running 'readelf -Sh arduino/crownstone.elf' I could see that the '.arduino_handler' section is created. Sadly when using the nm command, the function symbol 'jumpToCallback' is not found. Also reading the output during compiling, there is no building process taking place for cs_ArduinoHandler.cpp. To fix that I edited the file 'bluenet/source/conf/cmake/crownstone.src.cmake' and added the cs_ArduinoHandler.cpp to the list.

```
    # The arduino handler specified to be compiled
    LIST(APPEND FOLDER_SOURCE "${SOURCE_DIR}/cs_ArduinoHandler.cpp")
```

Listing 17: Added to the cpp file

After I compiled the code I checked if the the functions were indeed inserted into the section. The symbols func, jumpToCallback or callback were not found. This could only mean that the compiler did not create the section from source code, or deleted the section because it was empty. There were 2 aproaches I could take to handle this problem:

1. I could either find a way for the compiler to create the section from source code.

2. I could create the section in the linkerscript and try to get the address from it and then create a function to write something unto the address.

Due to the fact that I don't have a lot of time left to finish this approach, my supervisor added new code for me to work with in bluenet. This way he is helped me so that wouldn't stay too long on this part of the assignment.

What my supervisor added:

- Have an ARDUINO configuration option.
- Create .arduino_handlers section in linker script

- Create arduinoCommand function that will be in this section.
- This function just logs something.
- Call this function (for now) from cs_Crownstone
- Use struct rather than function pointers
- Have method with char as argument.
- Create a REGISTER macro that automatically assigns a name as well.

Show how to iterate over registered handles as example. This shows better the indirection required between binaries. The only thing required in another binary is __start_arduino_handlers which is autogenerated by the linker through the proper .ld file and is exposedto c/c++ by using extern ... syntax.

What I needed to do next:

- Create a separate binary that uses the same named section.
- Call the function from that binary.
- Add parameter to the function.
- Create large switch statement to pass through any arduino call.
- Create second function to call Arduino code from Crownstone.

The newly added files were cs_Arduino.c and cs_Arduino.h.

```c
#pragma once

#ifdef __cplusplus
extern "C" {
#endif

#include <drivers/cs_Serial.h>

typedef void (*arduino_func_t)(const char);

struct arduino_handler {
    arduino_func_t f;
    char *name;
};

#define REGISTER_ARDUINO_HANDLER(func)                              \
    static struct arduino_handler __arduino_handler__ ## func      \
    __attribute__((__section__(".arduino_handlers")))              \
    __attribute__((__used__)) = {                                  \
        .f = func,                                                 \
        .name = #func,                                             \
    }

extern struct arduino_handler __start_arduino_handlers;
extern struct arduino_handler __stop_arduino_handlers;

/*
__attribute__((used)) static void arduino_init_all() {
    struct arduino_handler *iter = &__start_arduino_handlers;
    for (; iter < &__stop_arduino_handlers; ++iter) {
        LOGw("Registered handler %s", iter->name);
    }
}
*/

#ifdef __cplusplus
}
#endif
```

Listing 18: cs_Arduino.c

```
#include <arduino/cs_Arduino.h>

#include <drivers/cs_Serial.h>

/**
 * Arduino command.
 */
static void arduinoCommand(const char value) {
    LOGd("Arduino command %i", value);
}

REGISTER_ARDUINO_HANDLER(arduinoCommand);
```

Listing 19: cs_Arduino.h

There were are also some added lines in the cs_Crownstone.cpp and crownstone.src.cmake.

```
#include <arduino/cs_Arduino.h>
..
..
..
Crownstone::Crownstone(boards_config_t& board) :
..
..
..
#ifdef ARDUINO
    //arduino_init_all();
    struct arduino_handler *iter = &__start_arduino_handlers;
    for (; iter < &__stop_arduino_handlers; ++iter) {
        LOGd("Call handler %s", iter->name);
        iter->f(8);
    }
#endif
..
..
..
void Crownstone::init(uint16_t step)
```

Listing 20: cs_Crownstone.cpp

```
  IF(ARDUINO AND "${ARDUINO}" STRGREATER "0")
LIST(APPEND FOLDER_SOURCE "${SOURCE_DIR}/arduino/cs_Arduino.c")
  ENDIF()
  ..
  ..
  ..
  ADD_DEFINITIONS("-DARDUINO=${ARDUINO}")
```

Listing 21: crownstone.src.cmake

In the config of the build ‘ARDUINO=1‘ has to be added to the file to enable the use of the handler. The config file in ‘bluenet/config/‘.

I took the example blinky in ble_central from the nordic SDK 16 to create the binary. I edited the linkerscript by changing the address of the FLASH. I also created two new files to help create the .arduino_handlers section: ‘extra.c‘ & ‘extra.h‘. To have the binary files work with the exact same section, the arduino_handler section needs to be assigned to the same address in both binary files.

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x60000, LENGTH = 0x94000
  RAM (rwx) :  ORIGIN = 0x20001db8, LENGTH = 0xe248
  /* Specific memory space for the arduino section */
  ARDUINO (rx) : ORIGIN = 0x94000, LENGTH = 0x95000
}
```

Listing 22: Lines added in linker file

I gave it the permissions to be read and executed (rx) After compiling the section address displayed that they were on the same address

```
                                          (Address)
    [10] .arduino_handlers PROGBITS      00094000 024000 000020 00   AX  0   0  4
```

Listing 23: in blinky binary

```
                                          (Address)
    [ 8] .arduino_handlers PROGBITS      00094000 044000 000008 00   WA  0   0  4
```

Listing 24: In crownstone binary

6.2.2 Phase 1b

Use Case: Arduino code with Crownstone code

In this use case the goal is run arduino code on a nrf52-dk that also runs crownstone code. The crownstone code is to be used in arduino code for certain features.

Use Case 1: Arduino Blink A simple arduino script to light up LEDs by using the onboard buttons. The arduino script is only to be executed. It has no function calling yet from the crownstone.

Use Case 2: Arduino Bluetooth Sensor The arduino code is going to read values from the sensor using bluetooth connected sensor. The arduino must not directly connect the radio of the nrf52-dk board but must call upon the bluetooth functions of the crownstone code.

References

[1] A. Allain. Function pointers in c and c++. https://www.cprogramming.com/tutorial/function-pointers.html.

[2] R. Earnshaw. *ELF for the ARM Architecture*. ARM: Development systems Division - Compiler Tools Group, v0.3 edition, December 2003. document nr: GENC-003538.

[3] Nordic Semiconductors. Soc support in softdevice handle. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v14.2.0%2Fgroup__nrf__sdh__soc.html. v14.2.0.

[4] PlatformIO Revision. What is platformio? https://docs.platformio.org/en/latest/what-is-platformio.html.

[5] Red Hat. Sections command. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Using_ld_the_GNU_Linker/sections.html.

[6] Red Hat Developers. Linker scripts. https://sourceware.org/binutils/docs/ld/Scripts.html#Scripts.

[7] K. Reddy. What is flash memory? https://www.quora.com/What-is-flash-memory, October 2016.

[8] Team @ Tutorial Point. Embedded systems - interrupts. https://www.tutorialspoint.com/embedded_systems/es_interrupts.htm.

[9] Turbo J. How can i make an empty section with gnu ld? https://stackoverflow.com/questions/8870548/how-can-i-make-an-empty-section-with-gnu-ld.

7 Appendix

List of Figures

List of Tables

Listings

Testplan
Phase 0

Paul Wondel
0947421
`Technische Informatica`

HRo - Crownstone B.V. — November 15, 2019

Project Information

In phase 0 the goal is to run raw arduino code on the nordic develepor kit (Nordic nRF52-DK). Phase 0 has been broken down to 2 use cases.

- Use case 1: Run Arduino Blink
- Use case 2: Auto Intensity Control of Power LED

This test is for use case 1. The goal is to test the basic functions of the arduino library on the Nordic nRF52 DK. The functions digitalWrite(), digitalRead(), analogRead() & analogWrite() are subjected to the test in this plan. Also the posibility to use the analog and digital pins on the Nordic nRF52-DK is to be tested. In order to test this there are a few requirements that need to be met.

1 Use Case 1

1.1 Requirements

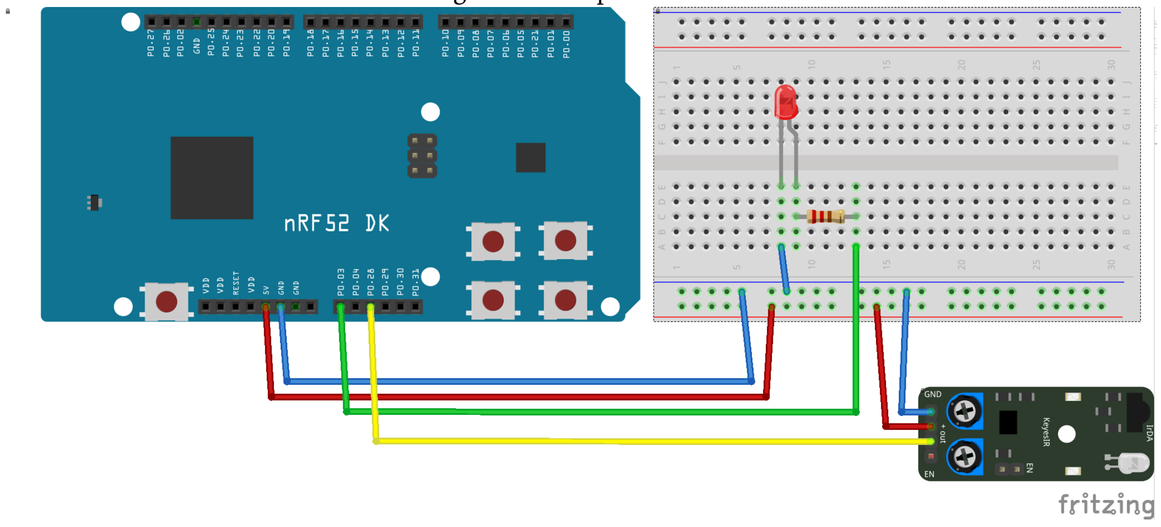
In preparations of this test these are the requirements that should be met:

- An Editor (Visual Code, Atom, eclipse, etc)
- Platformio IDE installed.
- Segger J-Link & tool-jlink (in Platformio) installed
- 1x Nordic nRF52 Developer Kit
- 1x Small LED
- 2x Circuit building wires
- 1x 220K Resistor
- Connecting Wires
- 1x Breadboard
- 1x Infrared Sensor
- Arduino Script: *'Blink'*

1.2 Constructing

These are the schematics and code for the test setup. Setup the parts in the same positions as in the schematics. Then upload the code to the board.

Figure 1: Setup Schemetic



Listing 1: Test Code

```
#include <Arduino.h>

#define ledPin PIN_A0
#define irPin PIN_A1

void setup() {
  Serial.begin(9600);
  pinMode(ledPin,OUTPUT);
  pinMode(irPin,INPUT);
}

void loop() {
  Serial.println(analogRead(irPin)); // Shows the values from the sensor
  if(analogRead(irPin) < 600){
    digitalWrite(ledPin, HIGH); // set the LED on
    Serial.println("Led␣On");
  }
  else {
    digitalWrite(ledPin, LOW); // set the LED off
    Serial.println("Led␣Off");
  }
}
```

1.3 Testing

Instructions: After constructing the circuit and uploading the script, the Infrared Sensor has to be configured. The Infrared Sensor has it's own resistor on the circuitboard which can be adjusted with a screwdriver. When the sensor has been configured, test the object detection. The values should appear in the serial monitor. The low values (when there is no object detected) should be between 0-200 and the high values should be higher than 600. To test the setup place an object in front of the sensor. The led should light up when the object is detected by the sensor.

Questions during the performance For performing the test itself there are a few questions that need to be answered during the test.

Question 1

Does the LED turn on when an object is placed in front the sensor?

a) Yes b) No

Question 2

Does the Infrared Sensor detect the object when placed in front of the sensor? (The light on the sensor itself wil light up when detecting an object)

a) Yes b) No

Question 3

Do the printed lines ’Led on’ or ’Led off’ appear in the serial monitor?

a) Yes b) No

2 Use Case 2

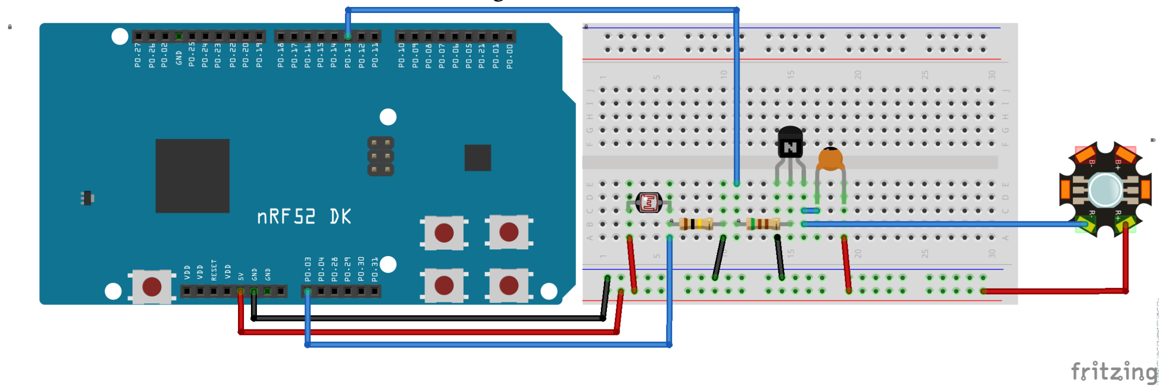
2.1 Requirements

- A Editor (Visual Code, Atom, eclipse, etc)
- Platformio IDE installed.
- Segger J-Link & tool-jlink (in Platformio) installed
- 1x Nordic nRF52 Developor Kit
- 1x LDR (Light Density Resistor) Sensor
- 1x Resistor (510, 100k ohm)
- 1x Capacitor (0.1uF)
- 1x Transistor 2N2222
- 1x 1 watt Power LED
- Connecting Wires
- 1x Breadboard
- Flashlight or mobile light source

2.2 Constructing

These are the schematics and code for the test setup. Setup the parts in the same positions as in the schematics. Then upload the code to the board.

Figure 2: Schematic



Listing 2: Test Code

```
#include <Arduino.h>

int pwmPin = (2); // assigns pin 12 to variable pwm
int pot = A0; // assigns analog input A0 to variable pot
int c1 = 0;    // declares variable c1
int c2 = 0;    // declares variable c2

void setup()  // setup loop
{
  pinMode(pwmPin, OUTPUT);
  pinMode(pot, INPUT);
  Serial.begin(9600);
}

void loop()
{
  int value = analogRead(pot);
  Serial.println(value);
  c1= value;
  c2= 500-c1;         // subtracts c2 from 1000 ans saves the result in c1

  if (value < 500)
  {
  digitalWrite(pwmPin, HIGH);
  delayMicroseconds(c2);
  digitalWrite(pwmPin, LOW);
  delayMicroseconds(c1);
  }
  if (value > 500)
  {
    digitalWrite(pwmPin,LOW);
  }
}
```

2.3 Testing

Instructions: After constructing the circuit and uploading the script, the setup is ready to be tested. To test the setup you must aim the light source in straight unto the LDR Sensor. This should cause the LED (that is turned on by default) to turn off.

Questions during the performance For performing the test itself there are a few questions that need to be answered during the test.

Question 4

Does the LED turn off when you shine light unto the LDR?

a) Yes b) No

Question 5

Does the serial monitor show the values given by the LDR?

a) Yes b) No

Testplan
Phase 1

Paul Wondel
0947421
`Technische Informatica`

HRo - Crownstone B.V. — December 6, 2019

Project Information

The goal in phase 1a is to run the arduino code that is created in phase 0 use case 2, at the same time that the crownstone code is running.

1 Use Case 1

b

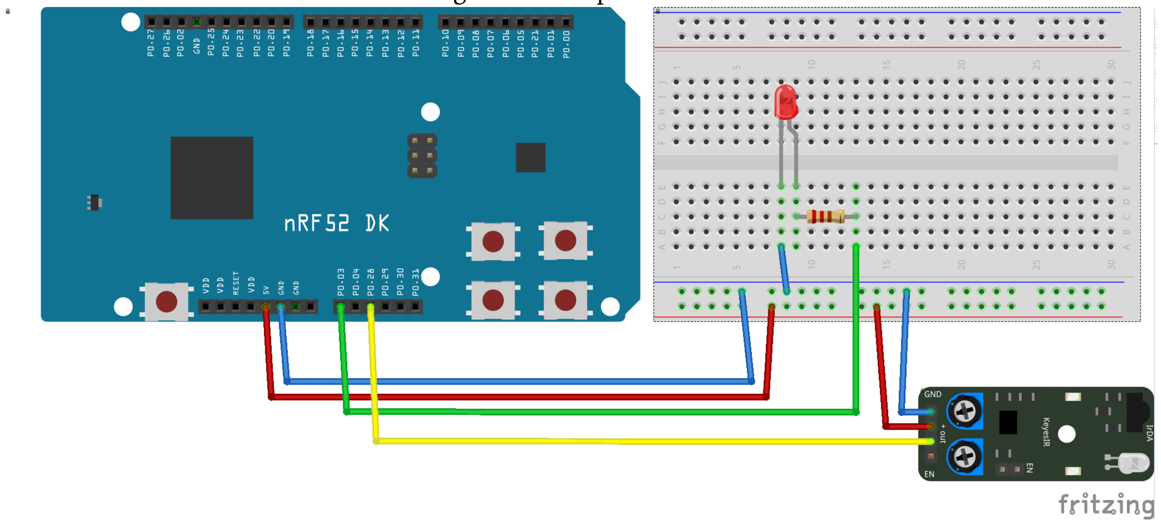
1.1 Requirements

- Crownstone bluenet cloned git repository
- An Editor (Visual Code, Atom, eclipse, etc)
- Platformio IDE installed.
- Segger J-Link & tool-jlink (in Platformio) installed
- 1x Nordic nRF52 Developer Kit
- 1x Small LED
- 2x Circuit building wires
- 1x 220K Resistor
- Connecting Wires
- 1x Breadboard
- 1x Infrared Sensor
- Arduino Script: *'Blink with use of Infrared Sensor'*
- Crownstone Smartphone App (Iphone or Andriod)

1.2 Constructing

These are the schematics and code for the test setup. Setup the parts in the same positions as in the schematics.

Figure 1: Setup Schemetic



Listing 1: Test Code

```
#include <Arduino.h>

#define ledPin PIN_A0
#define irPin PIN_A1

void setup() {
  Serial.begin(9600);
  pinMode(ledPin,OUTPUT);
  pinMode(irPin,INPUT);
}

void loop() {
  Serial.println(analogRead(irPin)); // Shows the values from the sensor
  if(analogRead(irPin) < 600){
    digitalWrite(ledPin, HIGH); // set the LED on
    Serial.println("Led␣On");
  }
  else {
    digitalWrite(ledPin, LOW); // set the LED off
    Serial.println("Led␣Off");
  }
}
```

1.3 Testing

Instructions: After constructing the circuit and uploading the bluenet code along with the crownstone code, the Infrared Sensor has to be configured. The Infrared Sensor has it's own resistor on the circuitboard which can be adjusted with a screwdriver. When the sensor has been configured, test the object detection. The values should appear in the serial monitor. The low values (when there is no object detected) should

be between 0-200 and the high values should be higher than 600. To test the setup place an object in front of the sensor. The led should light up when the object is detected by the sensor. During testing the arduino code has to function normally as it would on a arduino uno. You place an object in front of the sensor and the LED should turn on, remove the object and the LED should turn off. While that is being tested, the Crownstone App on the smartphone should detect and connect normally to the nordic device as if it was a released Crownstone plug.
If the nordic device cannot be detected, then it means that the bluenet code is not being executed. If the LED are not working but the nordic is being detected, it means that the arduino code is not being executed.

Questions during the performance For performing the test itself there are a few questions that need to be answered during the test.

Question 1

Does the LED turn on when an object is placed in front the sensor?

a) Yes b) No

Question 2

Does the Infrared Sensor detect the object when placed in front of the sensor? (The light on the sensor itself wil light up when detecting an object)

a) Yes b) No

Open the crownstone app on a smartphone.

Question 3

Can your smartphone detect the nordic device with bluetooth using the crownstone app?

a) Yes b) No

Question 4

Can your smartphone connect the nordic device with bluetooth using the crownstone app?

a) Yes b) No