



# Internet of Things for Smart Health: Smart Medication Bottle Cap

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of requirements for the Degree of Bachelor of Science

**Submitted by:**

Maria I. Rios-Martinez

**Submitted to:**

Xinming Huang, advisor

Department of Electrical and Computer Engineering

## Abstract

The Internet of Things (IoT) is a technology that enables new applications using sensors and wireless networks. Smart Health is one area in which IoT can improve the quality of healthcare and bring benefits to patients. In this project, a smart medication bottle cap with a smartphone app was designed and implemented to address the problem of medication non-adherence. Using Nordic's nRF52 *Bluetooth Low Energy* SoC in conjunction with a capacitive touch sensor, the prototype successfully tracks when a patient takes medication. The Android app, *SmartMed*, maintains a log detailing medication consumption habits and reminds patients to take medication timely. The smart cap also serves as a location tracker which aids a user in locating the bottle using light, sound and wireless signal strength.

## Acknowledgements

The completion of this project would not have been possible without the guidance and resources provided by the following people: Project advisor Xinming Huang for providing timely feedback and keeping the project focused. Dylan Baranik and William Appleyard who served as a valuable resource during PCB design and prototyping. Sagar Mahurkar and Alex Wald who also served as a valuable resource during Android development. Lastly, a thank you to the WPI Electrical and Computer Engineering Department for covering project costs.

## Table of Contents

|  |    |
|--|----|
| Abstract .....   | 2  |
| Acknowledgments .....  | 3  |
| Table of Figures .....   | 4  |
| List of Tables .....   | 6  |
| 1 Introduction.....  | 10 |
| 1.0 Project Overview and Objectives.....   | 10 |
| 1.1 The Internet of Things.....  | 11 |
| 1.2 Common Elements of IoT and Communication Technologies.....                                 | 13 |
| 1.2.0 ZigBee.....  | 14 |
| 1.2.1 NFC.....   | 14 |
| 1.2.2 Bluetooth Low Energy .....   | 14 |
| 1.2.3 WiFi .....   | 15 |
| 2 Background.....  | 16 |
| 2.0 Selecting Wireless Communication Technology.....   | 16 |
| 2.1 BLE System on Chip (SoC) Solutions .....   | 17 |
| 2.2 Smart Health IoT Device Ideas .....  | 18 |
| 2.2.0 Fall Detection Device - Thermal Imaging .....  | 19 |
| 2.2.1 Breathing Detection Device – Monitoring Senior or Infant in Bed .....                    | 19 |
| 2.2.2 Smart Medication Tracker – A Medicine Bottle Smart Cap.....                              | 19 |
| 2.2.3 Non-Invasive Glucose Monitoring System.....  | 20 |
| 2.3 Basics of <i>Bluetooth Low Energy</i> .....  | 20 |
| 2.3.0 What is <i>Bluetooth Low Energy</i> and how does it differ from <i>Bluetooth</i> ? ..... | 21 |
| 2.3.1 Platforms for <i>BLE</i> Development.....  | 22 |
| 2.3.2 <i>BLE</i> Protocol Stack Overview .....   | 22 |
| 2.4 nRF52832 SoftDevice and API.....   | 30 |
| 3 Methodology .....  | 32 |
| 3.0 Project Design Overview – Medicine Bottle Smart Cap .....                                  | 32 |
| 3.1 Overview of Profiles and Services.....   | 34 |
| 3.1.0 Smart Medication Tracker Profile.....  | 34 |
| 3.2 Firmware Development Hardware.....   | 35 |

|       |   |    |
|-------|---|----|
| 3.2.0 | nRF52832 Development Kit .....  | 35 |
| 3.2.1 | SparkFun nRF52832 Breakout.....   | 36 |
| 3.2.2 | IMM-NRF52832 Micro-Module.....  | 37 |
| 3.2.3 | Standalone Momentary Capacitive Touch Sensor Breakout – AT42QT1010.....         | 38 |
| 3.3   | Setting up the Development Environment .....                                    | 40 |
| 3.3.0 | GNU Toolchain for ARM Cortex-M (ARM GCC) .....                                  | 40 |
| 3.3.1 | nRF5x Software Development Kit v12.0.0.....                                     | 41 |
| 3.3.2 | GNU Make.....   | 41 |
| 3.3.3 | nRF5x Command Line Tools.....   | 42 |
| 3.3.4 | Programming nRF52 Modules and Prototype Boards: Serial Wire Debug .....         | 43 |
| 3.4   | Overview of Smart Medication Cap Firmware .....                                 | 45 |
| 3.4.0 | Initializing the SoftDevice in main.c .....                                     | 45 |
| 3.4.1 | Initializing GAP and Advertisement Parameters .....                             | 47 |
| 3.4.2 | Overview of Services and Characteristics.....                                   | 52 |
| 3.5   | Smart Medication Cap Prototype Hardware .....                                   | 57 |
| 3.5.0 | Initial Hardware Design with Hand Soldered nRF52832 .....                       | 57 |
| 3.5.1 | Second Hardware Design Using IMM-NRF52832 Micro-Module .....                    | 63 |
| 3.6   | Testing <i>BLE</i> Applications with <i>nRF Connect</i> Mobile Application..... | 65 |
| 3.7   | nRF52832 TX and RX Current Measurements.....                                    | 66 |
| 3.8   | Overview of Smart Medication Cap Android Application .....                      | 68 |
| 4     | Results.....  | 69 |
| 4.0   | Results for Device Firmwares using <i>nRF Connect</i> Application .....         | 69 |
| 4.0.0 | Smart Medication Cap Device Firmware.....                                       | 69 |
| 4.1   | Prototype Hardware Results for Smart Medication Cap.....                        | 74 |
| 4.1.0 | Prototype Hardware Results for Initial Design Revision .....                    | 74 |
| 4.1.1 | Prototype Hardware Results for Second Design Revision.....                      | 76 |
| 4.1.2 | Notes on Hardware Design .....  | 78 |
| 4.2   | nRF52832 TX and RX Current Measurement Results .....                            | 79 |
| 4.3   | Smart Medication Cap Android Application Results.....                           | 83 |
| 5     | Conclusion .....  | 89 |
| 5.0   | Future Recommendations .....  | 90 |
| 6     | References.....   | 92 |

|     |   |    |
|-----|---|----|
| 7   | Appendix.....   | 95 |
| 7.0 | nRF52832 SoC Specifications .....                           | 95 |
| 7.1 | nRF52832 Mechanical Specifications .....                    | 96 |
| 7.2 | Smart Medication Cap First Hardware Design Schematic .....  | 97 |
| 7.3 | Smart Medication Cap Second Hardware Design Schematic ..... | 98 |

## Table of Figures

|  |    |
|--|----|
| Figure 1.1: Overall vision of the IoT project .....  | 10 |
| Figure 1.2: Basic IoT device model [3]. .....  | 13 |
| Figure 2.1: The BLE Protocol Stack [9] .....   | 23 |
| Figure 2.2: Advertising Events [25].....   | 24 |
| Figure 2.3: Connection Events (M = master, S = Slave) [25].....  | 25 |
| Figure 2.4: General BLE packet structure [25].....   | 26 |
| Figure 2.5: GATT profile hierarchy [30].....   | 29 |
| Figure 2.6: Block diagram of nRF52 SoC software architecture [31]. .....   | 30 |
| Figure 3.1: Project Design Flow .....  | 33 |
| Figure 3.2: Smart Medication Tracker Profile with its respective services and characteristics.....   | 34 |
| Figure 3.3: Official nRF52832 Development Kit used for initial firmware development and breakout board/prototype programing [31].....                        | 36 |
| Figure 3.4: SparkFun nRF52832 Breakout board [32]. .....   | 37 |
| Figure 3.5: IMM-NRF52832 Micro-Module [33]. .....  | 38 |
| Figure 3.6: Adafruit’s Momentary Capacitive Touch Sensor Breakout - AT42QT1010 .....   | 39 |
| Figure 3.7: Contents of Makefile.windows file. ....  | 41 |
| Figure 3.8: Terminal output of make showing linked source files and output hex file. ....  | 42 |
| Figure 3.9: nrfjprog command line tools for programming the nRF52 SoC with BLE applications.....   | 43 |
| Figure 3.10: Serial Wire Debug (SWD) pins on Debug Out Connector P20 for SWD programming [37].<br>.....  | 43 |
| Figure 3.11: SWD programing interface between nRF52 Development Kit (Debug Unit) and external nRF52 SoC [37]. .....  | 44 |
| Figure 3.12: Physical setup for SWD programming of the IMM-NRF52832 micro-module breakout board. ....  | 45 |
| Figure 3.13: Breadboarded prototype circuit using SparkFun’s nRF52 breakout board. ....  | 58 |
| Figure 3.14: NPN transistor driver circuit for piezoelectric buzzer/magnetic transducer audio elements..   | 59 |
| Figure 3.15: PCB layout for complete custom prototype with hand soldered nRF52 SoC. ....   | 61 |
| Figure 3.16: ATMEL AT42QT1010 sensor IC circuitry [34]. .....  | 62 |
| Figure 3.17: Breadboarded prototype circuit with IMM-NRF52832 micro-module. ....   | 63 |
| Figure 3.18: PCB layout for final revision of smart medication cap prototype using IMM-NRF52832 module. ....   | 64 |
| Figure 3.20: nRF Connect BLE device scanner listing discovered devices, including the MQP_SMART IoT device. ....   | 65 |
| Figure 3.21: Ten ohm resistor and oscilloscope probe placement for current measurement. ....   | 67 |
| Figure 4.1: nRF Connect Scanner listing the discovered smart medication cap device by name and listing important properties of advertising/scan packets..... | 69 |
| Figure 4.2: Variations in advertising interval duration for the smart medication cap in fast advertising mode along with RSSI value plot. ....               | 70 |
| Figure 4.3: Variations in advertising interval duration for the smart medication cap in slow advertising mode along with RSSI value plot. ....               | 71 |

|  |    |
|--|----|
| Figure 4.4: nRF Connect listing tracker service and its characteristics upon connection to the smart medication cap device.....              | 72 |
| Figure 4.5: nRF Connect listing sensor service and its characteristics upon connection to the smart medication cap device.....               | 72 |
| Figure 4.6: Debug window output on nRF Connect upon writing 0x01 to LED state characteristic on tracker service. ....                        | 72 |
| Figure 4.7: Debug window output upon enabling notifications for sensor characteristic and receiving sensor data.....                         | 73 |
| Figure 4.11: Front and back of manufactured PCB for smart cap using hand soldered nRF52 SoC.....   | 74 |
| Figure 4.12: Hand soldered nRF52 SoC on prototype PCB. ....  | 75 |
| Figure 4.13: Incorrectly placed signal and VCC vias in coin cell battery area which would lead to multiple shorts.....                       | 76 |
| Figure 4.14: Front and back of manufactured PCB for smart cap using IMM-NRF52832 micro-module.....   | 76 |
| Figure 4.15: Assembled prototype PCB for smart medication cap device. ....   | 77 |
| Figure 4.16: Finished prototype with PCB hardware and medication bottle.....   | 78 |
| Figure 4.17: Advertising event for nRF52 SoC when advertising interval is 1 s (red signal). ....   | 79 |
| Figure 4.18: Advertising event for nRF52 SoC when advertising interval is 100 ms (red signal). ....  | 81 |
| Figure 4.19: Connection event for nRF52 SoC (red signal). ....   | 82 |
| Figure 4.20: Welcome screen for SmartMed android application, a companion app to the smart medication cap. ....                              | 83 |
| Figure 4.21: SmartMed application scanner which has discovered the prototype device “MQP_SMART.”<br>.....                                    | 84 |
| Figure 4.22: Control panel for the SmartMed android application.....   | 85 |
| Figure 4.23: Settings screen where the user can input medication stats such as name and time at which medication is taken. ....              | 86 |
| Figure 4.24: Find medication screen in which the user can flash or ring the device and get approximate distance through signal strength..... | 87 |
| Figure 4.25: Touch sensor timestamp log in SmartMed application. ....  | 88 |
| Figure 7.1: 6mm x 6mm QFN48 package and dimensions [18]:.....  | 96 |
| Figure 7.2: QFN48 package dimensions (mm) [18]. ....   | 96 |



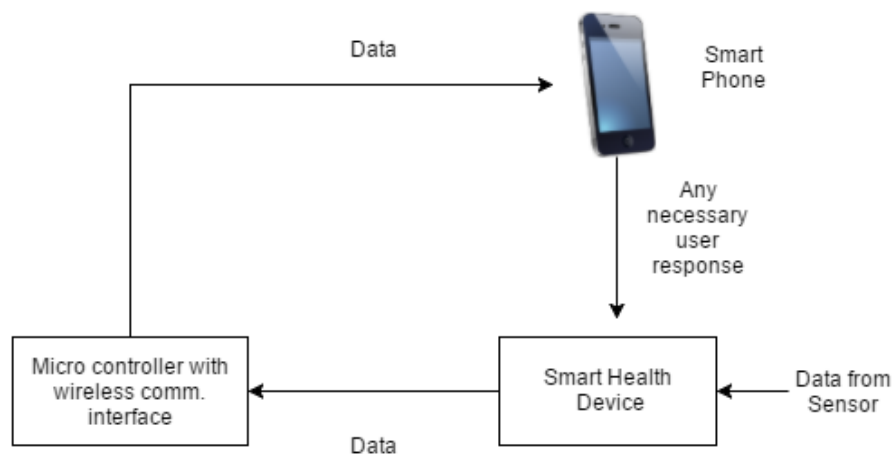
## List of Tables

|   |    |
|---|----|
| Table 2-1: Overview of Wireless Communication Technologies [12], [13].....              | 16 |
| Table 3-1: GAP initialization parameters for smart medication cap.....                  | 49 |
| Table 3-2: Fast and slow advertising parameters for smart medication cap. ....          | 52 |
| Table 3-3: Operation properties for characteristics in tracker and sensor services..... | 53 |

# 1 Introduction

## 1.0 Project Overview and Objectives

Advances in wireless communication technologies and the growing popularity of the Internet of Things (IoT) have led countless startups and big name companies to invest in the development of marketable IoT products in fields such as smart health, smart home and industrial automation. The purpose of this project was to design and implement an IoT device related to the field of smart health. Component requirements of the device include a sensor, an embedded microprocessor, a wireless communication interface and a smartphone app. Figure 1.1 shows a block diagram of the overall project vision.



*Figure 1.1: Overall vision of the IoT project*

Design guidelines are as follows:

- The device must be small as a wearable or portable product and efficient enough to be powered with just a cell battery.
- The device must have a sensor for data collection.
- The device must have peripherals for interaction.
- The IoT device must be able to connect to a smartphone through wireless communication technology and an app which serves as user interface.

While keeping the design guidelines in consideration, the goal was to completely design and implement the firmware and embedded circuitry of the device along with the smartphone application. In addition, a small item tracker was designed using the same *Bluetooth Low Energy (BLE)* technology that was used for the smart health device. This report details the specific tools and methods used to accomplish these tasks.

## 1.1 The Internet of Things

The past few years have brought a surging interest in the Internet of Things (IoT) with a projected market of \$7.1 trillion by the year 2020. The boom has been led not only by countless startups, but also big name players like Google and Samsung with their respective billion dollar takeovers of IoT companies such as Nest and SmartThings [1]. In its broadest sense, IoT can be defined as a “scenario” in which computing ability and network connectivity extend to ordinary, everyday objects which are not considered computers. In such a scenario, these objects can now communicate with other devices such as smartphones with the purpose of exchanging useful data with little user intervention [2]. A device in an IoT network contains embedded technology which allows it to communicate with external devices and to sense and interact with its surroundings. With this embedded technology it is also able to collect and exchange raw numerical data which can be processed using cloud computing services or the computing capabilities of a receiving device. These devices are also uniquely identifiable over a network using IP addresses [3].

It is useful to think of IoT as a means to improving an already existing product, the best example being a smartwatch. A device whose sole purpose was to indicate time can now communicate with a user’s smartphone and collect useful health information such as heart rate and sleep pattern. The collected information can then be presented to the user through elegant graphical displays provided by a smartphone or computer application. But IoT applications do not end here. In this young business the possibilities seem endless with the most prominent areas being smart infrastructure, smart homes and smart healthcare [1], [4].

But the popularity of IoT has particularly been heightened by its endless possibilities in the field of smart health. For healthcare providers, caring for their patients often means dividing their time between interacting with a patient and searching through manual documentation and other patient records. With IoT solutions, it would be possible for healthcare professionals to access patient information in real-time to improve quality of service. It would also be possible to integrate data from consumer health devices such as fitness watches, glucose meters and other wearables into hospital databases [5]. For instance, a diabetic patient could be measuring blood glucose levels at home and this information would automatically be added to official medical records. With such a system a practitioner would be alerted of changes in the patient's glucose level and he or she could make necessary adjustments in medication, all without the patient having to visit the hospital.

However, the advent of IoT has come with its notable security and privacy concerns. When developing an IoT network device, security requirements can be categorized into the three areas of confidentiality, integrity and availability. With confidentiality, a set of rules are applied to limit the unauthorized access to sensitive information while with integrity, the provision of a reliable service is ensured [6]. This is especially applicable to IoT devices in the field of smart health. A patient such as the one described previously would want assurance that blood glucose measurements and other patient information reaches the hands of authorized health personnel in a secure manner without eavesdropping from an unwanted party. In addition, the patient and practitioner would need confirmation that blood glucose data received is authentic (i.e integrity is preserved). Finally for devices in other prominent IoT fields, such as a smart home security system, constant service availability is a major concern [6]. This means providing appropriate protection from outside attackers for an almost guaranteed uninterrupted service.

Although exciting and full of innovation, the world of IoT technology does present significant design challenges depending on the complexity of the specific application. For smart home and industry systems, security is most likely the primary concern when it comes to managing network connections and accessibility. The same is true for other health and wearable devices but the problem of power provision

and efficiency now also becomes a primary concern. For instance, in this project, the goal is to create an IoT device that is powered simply by a coin cell battery and can last for weeks or months.

## 1.2 Common Elements of IoT and Communication Technologies

In its most basic form, a typical IoT device has I/O interfaces for sensor input and actuators along with a processing unit with memory. Other interfaces are for connection with external devices [3]. Depending on the specific application, the sensor and actuator can take the bulk of the design process since these two components are responsible for the interaction of the IoT device with its surrounding environment. The sensor collects data and it is transferred to a connected device using wireless communication technologies. Processing of this data can be performed using cloud computing services or the computing capabilities of the receiving device itself such as a smartphone or computer. The actuator is a component that acts on the environment in some way. For example, when a temperature sensor senses a temperature above a certain threshold, it alerts the user through a smartphone app and based on the user's input, the actuator is instructed to turn on the AC [3]. Figure 1.1 shows a simple block diagram of IoT at a glance.

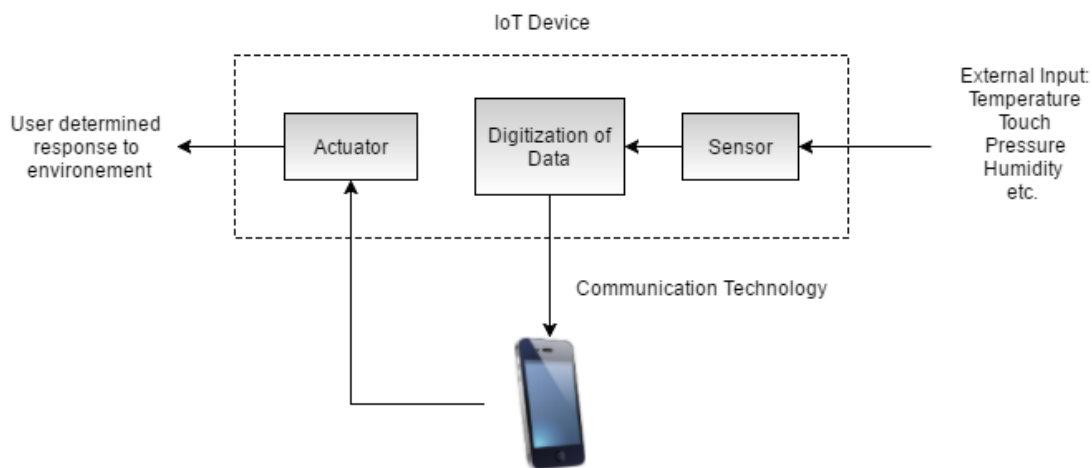


Figure 1.2: Basic IoT device model [3].

Finally, the communication technology is responsible for the communication and data transfer between the IoT object and other devices such as phones or computers. It wasn't until 15 years ago that the

concept of IoT began to fully take off owing to the work performed at MIT Auto-ID labs on networked radio-frequency identification (RFID) infrastructures [1]. RFID is a wireless communication and data collection technology which uses radio frequency to transfer data between an RFID reader and a tag consisting of a small chip or label attached to an item. The main purpose of RFID was to identify, categorize and track small movable items in an efficient manner with the reader transmitting a query signal to the tag. The received signal is then reflected back to the reader and subsequently passed to a database. The range of this technology can vary between a few centimeters to several meters and it is both cheap and energy efficient [3].

But IoT has since been expanded beyond RFID technologies and today multiple wireless communication technologies exist and each have their pros and cons based on specific applications. Following is a brief overview of some of the most popular communication technologies available for IoT:

### 1.2.0 ZigBee

ZigBee is a popular wireless network standard due to its low cost and power consumption [3]. It is targeted at radio-frequency applications that require a low data rate and common implementations include small personal area networks created using low-power digital radios for smart home automation and energy systems [7], [8]. A ZigBee radio module is often integrated with a small microcontroller.

### 1.2.1 NFC

NFC is a set of short range wireless technologies usually requiring 10 cm or less. This particular communication technology requires an initiator and a target where the initiator actively generates an RF field that can power a passive target. This mechanism allows targets to take form as small objects such as tags, stickers or cards that do not require batteries. Back and forth communication between connected devices is only possible when both are individually powered [3].

### 1.2.2 Bluetooth Low Energy

*Bluetooth Low Energy (BLE)*, also known as *Bluetooth Smart*, is the power friendly version of the *Bluetooth 4.0* specification for short range wireless communication and was specifically designed for IoT.

Its low power usage makes it suitable for devices designed to run for long periods using small coin cells or solar panels [3]. Major components of *BLE* are often implemented as small System-on-Chip (SoC) solutions with an integrated radio, making it quite convenient to integrate *BLE* into small embedded devices [9]. However, although there are similarities, classic *Bluetooth* and *BLE* not compatible [10].

### 1.2.3 WiFi

WiFi is the common name for the IEEE 802 standard of data transmission which sets up local area networks to transmit and receive data over short distances [11]. It is the staple of home and business networking and it is widely used for high data rate transfers with a max throughput of up to 54 Mbits/s. However compared with other technologies such as ZigBee and *BLE*, the implementation of a WiFi network usually requires higher power consumption and processor resources [12].

## 2 Background

### 2.0 Selecting Wireless Communication Technology

Before discussing device ideas for the project, a wireless communication technology was first selected which helped in narrowing down the specific device to be designed and its application. Table 2-1 gives an overview of the wireless communication standards discussed previously and some of their pros and cons.

Table 2-1: Overview of Wireless Communication Technologies [12], [13].

|               | Freq. Band                 | Range                    | Power Source  | Pros   | Cons  |
|---------------|----------------------------|--------------------------|---------------|--|---|
| <b>ZigBee</b> | 868MHz<br>915MHz<br>2.4GHz | Many meters              | Battery/Wired | Low power usage and available in small modules for easy development.   | Does not have as much developer or OS support as BLE.                                     |
| <b>NFC</b>    | 13.56MHz                   | ~10cm                    | Battery       | Low power and depending on application, tags don't need to be powered allowing for very small designs.                   | Much smaller range compared to all other wireless technologies.                           |
| <b>BLE</b>    | 2.4GHz                     | Up to 100m<br>(outdoors) | Battery       | Low power usage and allows for instant network setup. Plenty of developer support and almost every major OS supports it. | Lower data throughput than both WiFi and classic <i>Bluetooth</i> .                       |
| <b>WiFi</b>   | 2.4GHz<br>5GHz             | 20-140 m                 | Wired         | Standard for home and office networking and has high data throughput.  | Higher power consumption for some IoT applications and requires large hardware resources. |



We chose *BLE* because other wireless technologies were not the best for the smart cap project. For, example, WiFi consumes too much power and requires more hardware resources than are appropriate for mobile devices that can be taken out of the home. The same is true for ZigBee: although it is easier to establish a network with it than it is with WiFi, it is still a tool better suited for creating a Wireless Local Area Network (WLAN). A WLAN interconnects devices in limited areas such as a business or residence [14]. On the other hand, *BLE* is better suited for creating Wireless Personal Area Networks (WPANs) which are described by the IEEE as “networks used to convey information over short distances among a private, intimate group of participant devices.” Unlike WLANs, WPANs involve little or no infrastructure outside of the link between two devices which allows for small and power efficient wireless solutions [15].

Therefore, both WiFi and ZigBee offer too much complexity for a simple smart health device that only needs to connect to a user’s smartphone. On the other hand, although NFC contains many benefits such as its low power consumption, its incredibly small range made it not suitable for the design. On the other hand, *BLE* advertises a maximum range of 100 meters (although in practice it is usually half this) which is more than enough for this project [16]. *BLE* is also supported by both Android and iOS platforms and each have considerable developer resources. Embedded firmware development for *BLE* is also widespread and developer support is readily available.

## 2.1 *BLE* System on Chip (SoC) Solutions

There are multiple *BLE* SoC options available in market from companies such as Texas Instruments (TI), Cypress Semiconductor and Nordic Semiconductor. Most of the available SoCs contain an integrated processor and radio along with flash memory and RAM. The chips are also manufactured in two packages: QFN and WLCSP. QFN is the standard Quad Flat No Leads package and is often easier to use and results in less PCB design error due to their larger size. WLCSP or Wafer Level Chip Scale packages, on the other hand, are made with less material and are less expensive. However, their smaller size increases PCB design complexity, driving up costs and hardware failure [17].

The major factors in choosing a *BLE* SoC were ease of use and availability of development support. This meant selecting an SoC which came with significant software resources including libraries and examples, an easy to use evaluation or development kit and other useful development resources.

A set of widely used *BLE* SoCs is Nordic Semiconductor's nRF5x series with the newest iteration being the nRF52832. This was the *BLE* chip recommended by the project's advisor and was the one selected for this project. This particular SoC is built around a 32-bit ARM Cortex-M4 processor with 512kB of programmable flash plus 64kB RAM and is available in a 48 pin QFN or WLCSP package [18]. Unlike other SoCs, Nordic's nRF52832 is supplemented with an extensive set of software tools which include a pre-qualified *Bluetooth 4.2 BLE* protocol stack conveniently provided as a precompiled binary image [18]. Therefore, the developer is mostly concerned with the implementation of the higher level application. The Software Development Kit provided also contains an extensive collection of libraries and example *BLE* applications.

Although Nordic does not include a free IDE for development with nRF5 products, they do provide multiple tutorials on options for firmware development using free tools such as ARM-GCC compiler and Eclipse for IDE. In addition, the complete development kit for the nRF52832 chip is much more reasonably priced than the official kits for TI's CC2541.

The complete key features of the nRF52832 from the product specification document are included under Appendix 7.0. As for mechanical specifications, Appendix 7.1 also includes the QFN48 package specifications with dimension details along and pin assignments.

## 2.2 Smart Health IoT Device Ideas

Several ideas for a smart health IoT device were considered and the fact that *Bluetooth Low Energy* had been selected as the wireless technology made it easier to narrow down a specific application. Other factors such as cost, difficulty and previous work in the literature were also considered when reviewing ideas. Below is a brief evaluation of several ideas discussed.

### 2.2.0 Fall Detection Device - Thermal Imaging

A way to detect falls is using thermal imaging sensors with a simple method being comparing thermal imagery data of subjects behaving normally to subjects that are falling [19]. Although thermal imaging is much more accurate than other methods, small thermal cameras are expensive and are useful for detecting falls close to where they are located. In addition, *BLE* is not particularly suitable for audio, image or video applications due to its small data payload. Although Nordic Semiconductor advertises a 1Mbps to 2Mbps data rate for the nRF52, this is simply for the physical layer. The maximum throughput for the application layer is 236.7 kbps [9].

### 2.2.1 Breathing Detection Device – Monitoring Senior or Infant in Bed

There are multiple physiological conditions that can lead to the cessation of breathing, known as apnea. These conditions include respiratory diseases and other unknown reasons such as sudden infant death syndrome and sudden adult death syndrome [20]. The advisor for this project had previous experience with breathing detection using microphone arrays which capture breathing easily by subtracting background noise. However, limited sensitivity was found using omnidirectional microphones. For this project, directional microphones would be used to capture breathing. Data would first be processed using a digital signal processing chip and then sent wirelessly to a smartphone. But as mentioned previously, *BLE* is not suitable for audio applications. In addition directional microphones are expensive and issues of audio data privacy come into play.

### 2.2.2 Smart Medication Tracker – A Medicine Bottle Smart Cap

Medication adherence (i.e. taking medications as prescribed by health providers) is key in achieving optimal treatment results for most medication regimens. However, it is estimated that around 20% to 50% of patients are non-adherent to prescribed regimens with rates increasing from 40% to 80% among elderly patients. When a patient suffers from a chronic disease, non-adherence also leads to higher hospitalization rates and treatment costs [21]. There are multiple medication reminding methods ranging from text message reminders, to smart watches that vibrate at a desired time. This project would be concerned with the design

of a *BLE* smart cap that could be placed on most generic medicine bottles. This cap would serve as both a tracker for a medicine bottle and a device that, along with a smartphone application, reminds a user to take their medication and keeps track of whether it has been taken or not. This device would involve a simple pressure or contact sensor and little to no data processing would be required. The simple and functional design would also make it a cheap and easily marketable device.

### 2.2.3 Non-Invasive Glucose Monitoring System

The majority of blood glucose monitoring devices available in market use a cost-effective electrochemical biosensor that has been proven accurate in glucose detection. However, these devices employ a small needle to prick the fingertip to acquire a blood sample which leads to pain in users due to frequency of glucose checks [22]. One alternative would be using radio waves at the skin or capillary level. These waves are reflected back to a sensor and analyzed for patterns in blood characteristics [23]. However, most non-invasive technologies are not as accurate as devices that require direct access to blood, and estimates could be life threatening to diabetic patients. [22].

After carefully evaluating these ideas, it was concluded that a smart medication tracker device would be the most achievable design in the allotted project time. For this application, no difficult data processing would be required and the type of data would be small and numerical which is what *BLE* is intended for. The small size of this device would make it easily portable and would also make the project cost effective in both development and production.

## 2.3 Basics of *Bluetooth Low Energy*

This section provides a brief overview of the *BLE* wireless communication protocol and includes as much detail of the protocol stack architecture as is required for the understanding of the nRF52 SoC firmware and android application developed for this project.

### 2.3.0 What is *Bluetooth Low Energy* and how does it differ from *Bluetooth*?

As discussed in Section 1.2, *Bluetooth Low Energy (BLE)* is sometimes referred to as *Bluetooth Smart* and was first introduced as a subset of the *Bluetooth 4.0* core specification. Although there is some overlap between *BLE* and classic *Bluetooth*, *BLE* is a completely different wireless communication protocol and was originally an in-house project in Nokia known as “Wibree” [10].

Classic *Bluetooth*, which is referred to as Basic Rate/Enhanced Data Rate (BR/EDR), operates in the unlicensed industrial, scientific, and medical radio band (ISM band) at 2.4GHz [24]. This wireless communication protocol was developed in 1994 by Ericsson Mobile and was based on frequency-hopping spread spectrum technology [13], [24]. By using a frequency-hop transceiver it was possible to combat interference and fading [24]. The Basic Rate implementation of classic *Bluetooth* supports a bit rate of 1 Mbps while the Enhanced Data Rate implementation supports a bit rate of 2 Mbps [24]. In addition, classic *Bluetooth* is connection oriented which means that once a device is connected, a link is maintained indefinitely, even if there is no data transfer [13]. Therefore, classic *Bluetooth* is ideal for applications in which a relatively short ranged but continuous connection is required [25].

Like classic *Bluetooth*, *BLE* operates in the unlicensed 2.4GHz ISM band and it also employs a frequency-hopping transceiver to combat interference and fading [25]. *BLE* is often thought of as a stripped down version of classic *Bluetooth* and it is most suitable for applications that do not require large data exchanges since its application layer throughput is only 236.7 kbps [9]. These small data transactions in *BLE* allow for devices to be battery powered for longer periods. On the other hand, the high data throughput of classic *Bluetooth* means that battery life is consumed fairly rapidly [13].

However, *BLE* does not support data streaming due to its lower data rate and is therefore not suitable for applications such as audio streaming [13]. And unlike *Bluetooth*, once a connection has been established in *BLE*, the device spends most of its time in sleep mode waiting to send or receive small data packets. Once data transfers are done, the device goes back to sleep mode to conserve energy.

### 2.3.1 Platforms for *BLE* Development

Multiple wireless communication protocols are available as outlined previously and what makes *BLE* particularly attractive to product developers is that it is the easiest way to design a device that can communicate to any modern mobile platform. Especially for Apple devices, *BLE* is the only hardware design option that doesn't require developers to take complicated steps to be able to legally market their products for iOS devices [10]. Support for *BLE* is available for most major platforms as listed below:

- Android 4.3+
- iOS7+
- Apple OS X 10.6+
- Windows 8+
- GNU/Linux Vanilla BlueZ 4.93+

### 2.3.2 *BLE* Protocol Stack Overview

The *BLE* protocol stack has two main components: The Controller and the Host. Communication between them is standardized as the Host Controller Interface (HCI). The Controller includes the Physical Layer and the Link Layer and it is typically implemented as an SoC with an integrated radio. The Host on the other hand, runs on a processor and includes the upper layer functionality which is comprised of the Logical Link Control and Adaptation Protocol (L2CAP), the Attribute Protocol (ATT), the Generic Attribute Protocol (GATT), the Security Manager Protocol (SMP), and the Generic Access Profile (GAP). Other application layer functionality not defined by the *Bluetooth* specification can also be included on top of the Host [9]. Figure 2.1 shows a diagram illustrating the *BLE* protocol stack.

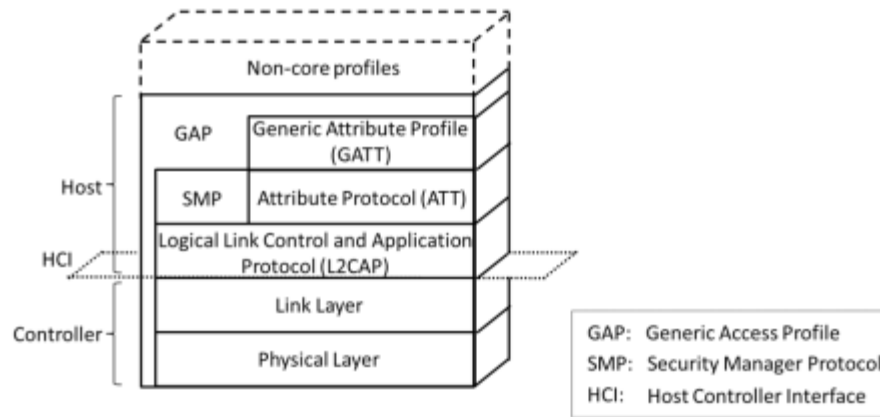


Figure 2.1: The BLE Protocol Stack [9]

As mentioned previously, there is some overlap between the *BLE* and classic *Bluetooth* protocols. However, the two are not compatible and devices that only implement *BLE* (a *single-mode* device) cannot communicate with devices that only implement classic *Bluetooth* [9]. It is possible to have *dual-mode* devices which implement both *BLE* and classic *Bluetooth* protocols [9], [25].

### 2.3.2.1 Physical and Link Layer

*BLE* operates at the 2.4GHz ISM band and it employs the frequency division multiple access (FDMA) scheme. The FDMA scheme uses 40 radio frequency (RF) channels separated by 2 MHz with three denoted as advertising channels and 37 as data channels. The advertising channels are used for device discovery, connection establishment and broadcast transmission while data channels are used for two-way communication between connected devices [25], [9].

The physical layer is divided into time units known as events and data is transmitted between connected devices in packets that are positioned into each event. Devices that transmit advertising data are known as *advertisers* while devices that receive advertising packets without the intention to connect are known as *scanners*. Transmission of data through advertising channels takes place during advertising events. Within one event, the advertiser uses each advertising channel sequentially to transmit advertising packets [9]. Figure 2.2 shows an illustration of advertising events and demonstrates the sequential use of advertising channels.

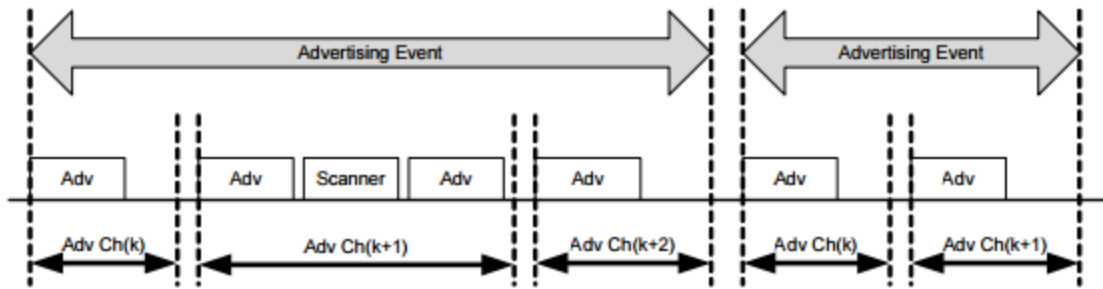


Figure 2.2: Advertising Events [25]

Devices that scan for connectable advertising packets in order to form a connection and are referred to as *initiators* [25], [9]. If the advertising device is broadcasting a connectable advertising packet, an initiator makes a connection request using the same advertising channel [25]. In the Link Layer, *initiators* and *advertisers* are assigned the respective roles of *master* and *slave*. A master device can manage multiple simultaneous connections with several slaves while a slave can only be connected to one master at once. This network is known as a “star topology” [9].

A slave device spends most of its time in sleep mode and wakes up periodically to listen for possible packets from the master [13], [9]. The master determines when the slave is required to listen and coordinates the medium access by using an FDMA scheme. Once a connection between the slave and master is established, the physical channel is divided into connection events. Within these connection events, data packets are transmitted bi-directionally using the same data channel. All connection events are initiated by the master and once the slave receives a transmission packet, it must respond. While there is packet exchange between the master and slave, the connection event is considered open. Each data packet includes a More Data (MD) bit which signals whether the sender has more data to transmit. If no more data is waiting for transmission, then the connection event will close and the slave device is no longer required to listen until the start of the next connection event [9]. Figure 2.3 shows an illustration of connection events.



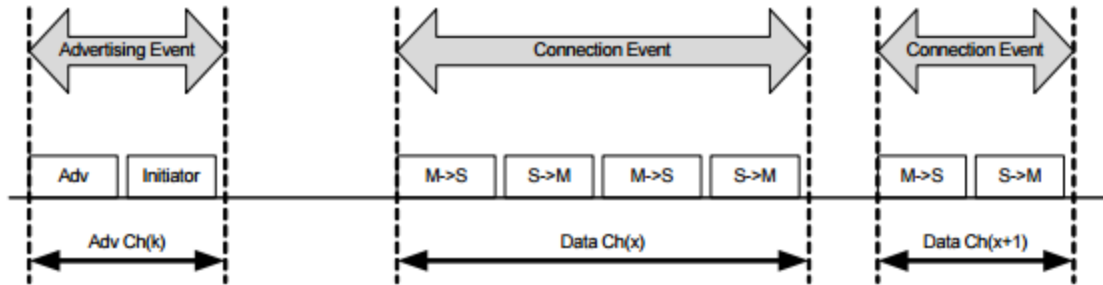


Figure 2.3: Connection Events ( $M = \text{master}$ ,  $S = \text{Slave}$ ) [25].

For every new connection event, a new data channel is selected for transmission using the frequency-hopping scheme of FDMA. For these connection events, there are three important parameters that need to be kept in mind:

- **Connection Interval:** This is the time between consecutive connection events which can range from 1.5 ms to 4 s in multiples of 1.25 ms [9].
- **Slave Latency:** This is the number of consecutive connection events in which the slave is not required to listen to the master. This parameter is an integer between 0 and 499 [9]. The longer the slave latency, the more the slave is asleep and the more power is conserved.
- **Connection Supervision Timeout:** This parameter can range between 100 ms and 32 s and it indicates when a supervision timeout occurs. The purpose of supervision timeout is to detect the loss of a connection due to signal interference or device being out of connection range [9].

It has been found in previous research that the power consumption characteristics of *BLE* can depend mainly on the connection interval and slave latency parameters since they determine how long the slave device can remain in sleep mode [9].

### 2.3.2.2 More on Advertising and Data Packets

The general structure for *BLE* over the air packets is shown in Figure 2.4.

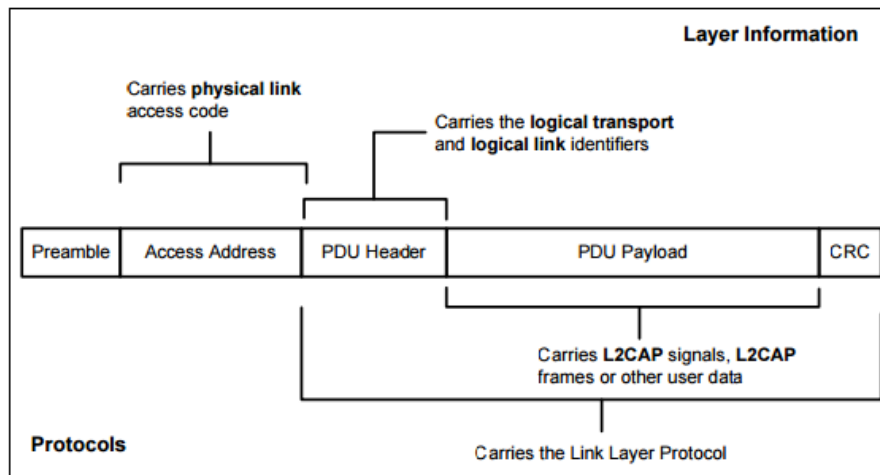


Figure 2.4: General BLE packet structure [25].

All *BLE* packets contain a 4 byte *Access Address* which is used to identify communications on a physical link. These addresses are important for ignoring all other close by packets on different physical links that are using the same physical channel [25], [26]. All packets also include a 1 byte *Preamble* and a 2-39 byte *Protocol Data Unit (PDU)* header and payload [26].

For advertising packets, the PDU consists of a 16-bit PDU header, and depending on specific type of advertising, a device address and up to 31 bytes of information [26]. An active scanner can also request up to 31 bytes of additional data from an advertiser in the form of a scan response packet [27]. On the other hand, the data packet PDU consists of the data packet PDU header and up to 37 bytes of payload. Depending on the data packet PDU, the payload can contain link layer control information or actual data for higher-level functionality [26]. As mentioned previously, data packets will also contain the MD bit to signal more data for transmission [9].

### 2.3.2.3 GATT and ATT Layers

The Generic Attribute Profile (GATT) defines a framework which uses the Attribute Protocol (ATT) for the transfer of data between connected devices using the concept of *Services* and *Characteristics*

[10]. Data related to services and characteristics are stored in the ATT layer as *attributes* in a simple lookup table [10], [9]. GATT is also responsible for determining the *client* and *server* roles in a connection and these roles are independent to the master and slave roles. Clients can access the attributes stored in the server by sending requests [9].

Since ATT is involved along with the GATT in defining a protocol for transferring attribute data, data packets in the PDU payload (Figure 2.4) are often referred to as ATT packets.

#### 2.3.2.4 GAP

The Generic Access Profile (GAP) is the highest level of the core *BLE* stack and its primary roles include controlling modes and procedures for the discovery of devices and their services, and managing connections and advertising [10], [9]. The GAP is also required for assigning the device roles of *broadcaster*, *observer*, *peripheral* and *central*. Relationships between two devices can either be *broadcaster-observer* or *central-peripheral*. A broadcaster simply broadcasts data via the advertising channels and does not support connection with other devices. The purpose of the observer then is to receive data transmitted by a broadcaster [9].

On the other hand, a central device is in charge of initiating and managing several connections [9]. In *BLE*, this central role is usually taken by devices with higher processing power such as smartphones or tablets [10]. Peripherals then are smaller and simpler devices which only support a single connection with a central device. In consequence, the controllers for central and peripheral devices need to assume the roles of master and slave respectively. Although a device may support multiple of the roles described above, only one can be assumed at a given time [9].

#### 2.3.2.5 Services, Characteristics and Profiles

According to the *Bluetooth* Core Specification, a service is a “collection of data and associated behaviors to accomplish a particular function or feature” [28]. In other words, a service is used to break up data that is transmitted between central and peripheral into logical entities [10]. Characteristics on the other hand, are “values used in a service along with properties and configuration information about how the value

is accessed and information about how the value is displayed or represented” [28]. These characteristics then, are the subcomponents of a service that contain the actual data. For example, for a wearable heart rate monitor, a heart rate service is established which contains characteristics such as the heart rate measurement and the heart rate sensor location on the body. The *Bluetooth Special Interest Group (SIG)* has predefined some standard services to ease the development of common IoT applications. However, the establishment of custom services by developers is also possible [28].

These services and characteristics are each identified by a unique number known as the *Universally Unique ID (UUID)*. Using these numbers, central devices can discover what kinds of services are being broadcast by the peripheral [28]. These UUIDs can also come in two sizes: 16-bit and 128-bit. Sixteen bit UUIDs are reserved for standard SIG services while 128-bit UUIDs are used for custom services and are often referred to as *vendor specific* [10], [28]. Although they take more space in advertising packets, 128-bit vendor specific UUIDs are recommended to significantly decrease the chance that no other service in the world shares the same UUID [28].

A vendor specific UUID is comprised of a *base UUID* that look as follows:

```
6C48xxxx-2EE9-E8C7-A7C7-F1532BB023E8
```

The four x’s in the base UUID represent a smaller 16-bit identifier that is attached directly to a custom service or a specific characteristic. Therefore, a single 128-bit base UUID can be used for multiple custom services with their respective characteristics which are each identified with a unique 16-bit number [28].

In the context of the *BLE* stack, the ATT contains a table in which each row is considered an attribute, and each attribute has a handle, a type, a set of permissions, and a value. Certain standard UUIDs are used to identify attribute types which can be Service Declarations (0x2800) for declaring services and Characteristic Value Declarations (0x2803) for declaring characteristics [29]. Once a Characteristic Declaration is made, a Characteristic Value Declaration can be made which is specific to the application

and is identified by a developer determined UUID. These characteristics will include access properties such as read, write or notify. Depending on the specific properties, they may also include Descriptor Declarations which hold additional information on the characteristic [29].

The concept of GATT then is to group attributes in an attribute table in a logical order forming *profiles*. These *profiles* are a pre-defined collection of services that are compiled by *Bluetooth* SIG or by product firmware developers [29]. Figure 2.5 shows the hierarchical structure of the GATT profile.

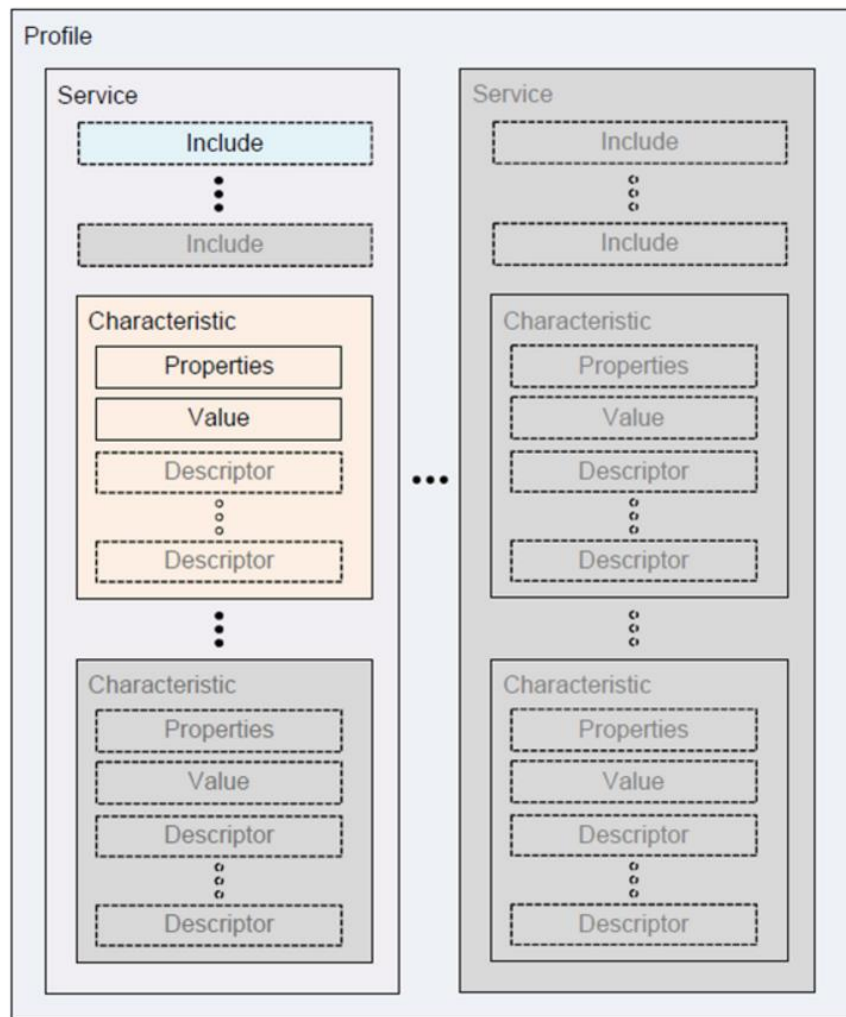


Figure 2.5: GATT profile hierarchy [30].

## 2.4 nRF52832 SoftDevice and API

For the development with Nordic’s *BLE* SoCs, Nordic provides a precompiled, linked binary image implementing a *BLE* protocol stack known as the SoftDevice. For the nRF52, the S132 version or above are the SoftDevice revisions that are recommended. As mentioned in Section 2.1, the SoftDevice enables the firmware developer to write their program as a standard ARM Cortex-M4 project without needing to integrate with proprietary Nordic software frameworks. This means that any ARM Cortex-M4 compatible toolchain can be used to develop with the nRF52 SoC [31]. A block diagram of the nRF52 software architecture is shown in Figure 2.6:

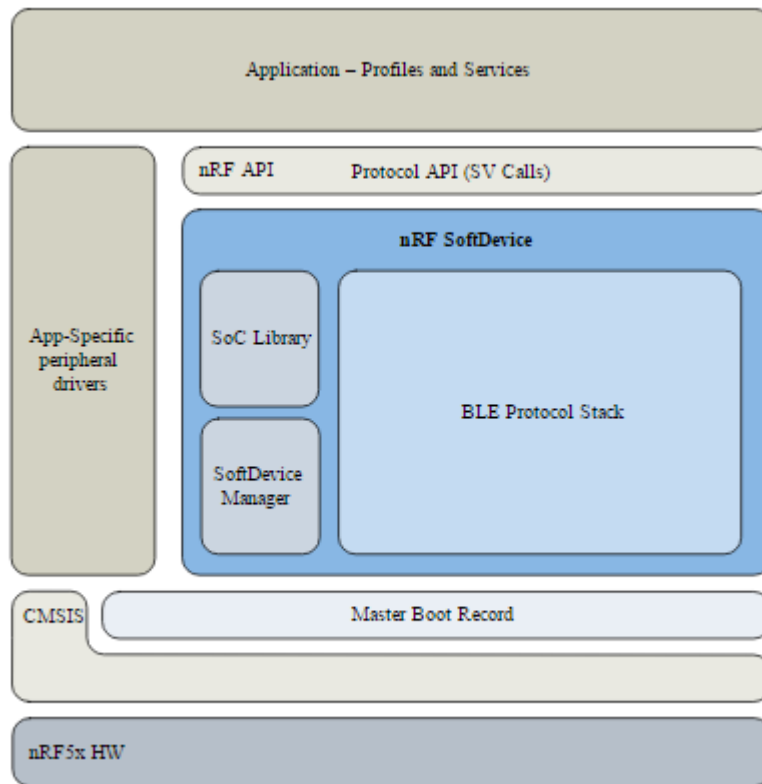


Figure 2.6: Block diagram of nRF52 SoC software architecture [31].

The nRF52 architecture includes an ARM Cortex Microcontroller Software Interface Standard (CMSIS) for interfacing with chip hardware, a master boot record, application specific peripheral drivers, the SoftDevice firmware module, and the profile and application code. The SoftDevice module then is composed of the binary image *BLE* protocol stack, the SoftDevice Manager and the SoC Library. The SoC

Library includes API for shared hardware resource management between the higher level application and the SoftDevice while the SoftDevice Manager includes all the API for *BLE* stack management such as enabling/disabling etc. [31].

The API (Application Programming Interface) is a set of standard C language data types and functions provided in the Software Development Kit (SDK) as a series of header files which give the higher level application code complete compiler and linker independence from the SoftDevice implementation [31]. In addition to the SoftDevice Manager and SoC Library, there are other API modules providing common definitions and functions for the SoftDevice, GAP, GATT client and server, and L2CAP layers [31].

## 3 Methodology

### 3.0 Project Design Overview – Medicine Bottle Smart Cap

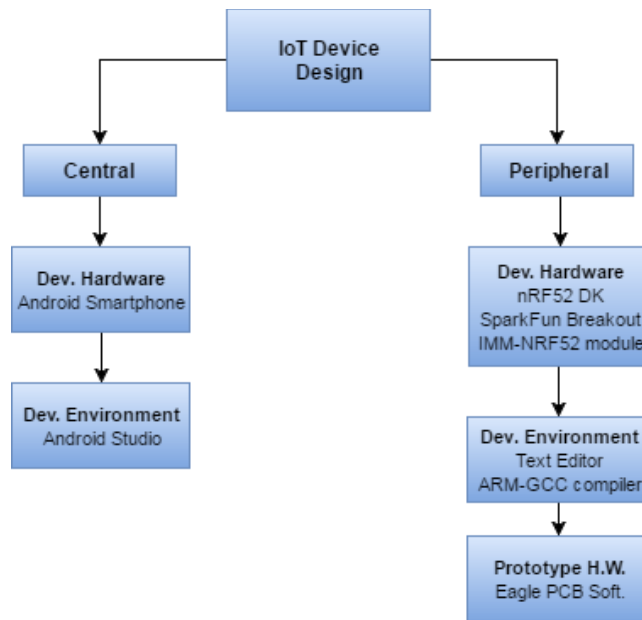
The goal of this project was to create a small IoT device for a smart health application that is small, portable and efficient enough to be powered with a coin cell battery. The device must have a sensor that collects data and it must wirelessly connect to a smartphone. As described in Section 2.2, a smart medicine bottle cap was the idea selected for the IoT device. The main purpose of this device is for patients, or even physicians, to keep a clear record of if and when their medication is taken with the aid of a companion smartphone application.

This device can play a positive role in the treatment management of patients since, as mentioned previously, it is estimated that around 20% to 50% of patients are non-adherent to prescribed regimens. For patients with chronic illnesses, nonadherence can lead to a higher number of hospital stays, which can lead to higher costs of health care.

As touched upon in Section 2.3.2, when forming a bidirectional connection between two devices, one must assume the roles of central and master and the other of peripheral and slave. The peripheral and slave roles are taken by the smart medicine cap device since it is small and only needs to support one connection to a central device. Hence, the smartphone takes on the central and master roles since it should be able to manage multiple connections to several slaves to form a star topology network if required. For instance, the smartphone can connect to multiple medicine bottle smart caps.

After developing the embedded firmware of the nRF52 chip for the smart medication cap device, a prototype hardware was designed. Figure 3.1 shows a chart demonstrating the project design flow. This included using firmware development hardware and a development environment to write central/profile device applications, and then using a PCB layout software for fast turnaround prototyping.

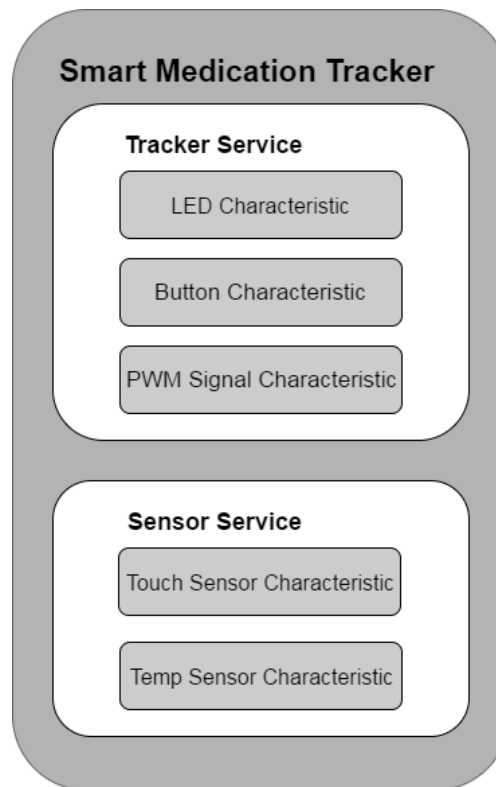




*Figure 3.1: Project Design Flow*

## 3.1 Overview of Profiles and Services

### 3.1.0 Smart Medication Tracker Profile



*Figure 3.2: Smart Medication Tracker Profile with its respective services and characteristics.*

From Figure 3.2, it can be seen that the smart medication cap application has a Smart Medication Tracker Profile which consists of the Tracker and Sensor Services. The Tracker service encompasses all the proximity tracking functionality for the smart medication cap and it consists of the LED, button and PWM signal characteristics. If a user cannot find his or her medication, they tap a button on their smartphone application that will flash an LED or play a repeating audio signal driven by one of the nRF52's pulse width modulated (PWM) channels. Through the button characteristic, a user can also locate their missing smartphone by pressing a button on the smart medication cap which will trigger the phone to ring.

The sensor service on the other hand, encompasses the functionality of the smart cap that will keep track of whether the user has taken his or her medication. This specific data is contained in the Touch Sensor Characteristic while data from the on-chip temperature sensor is contained in the Temp Sensor

Characteristic. A temperature characteristic was included as part of the Smart Medication Tracker Profile in case it is crucial that a user store their medication at a particular temperature.

Through these services and characteristics, the complete functionality of the smart medication tracker cap was established and a user is able to both locate their medication when misplaced and keep a record of if and when medication was taken. More detail on each characteristic such as their data types and GATT permission properties will be discussed in later sections. Details on the specific sensor used to collect data for the Touch Sensor Characteristic will also be discussed in later sections as well.

## **3.2 Firmware Development Hardware**

### **3.2.0 nRF52832 Development Kit**

The nRF52832 development kit is the official evaluation board sold by Nordic Semiconductor. The board has all GPIO pins and interfaces available at edge connectors. It can be powered using a 3V coin cell battery, a power supply, or a micro USB cable. Since programs cannot be flashed directly onto the nRF52832, the board also contains an on-board interface microprocessor unit (MCU) that is factory preloaded with SEGGER J-Link OB to program and debug firmware on the SoC.

The official development kit was selected for initial firmware development since it can be easily programmed and taken on the go with a battery. The on-board buttons and LEDs were also useful for fast development and testing. In addition, the kit came with five sample nRF52832 QFN48 chips which were originally intended for soldering onto a prototype PCB. Figure 3.3 shows the physical appearance of the nRF52832 development kit with the most relevant components pointed out:

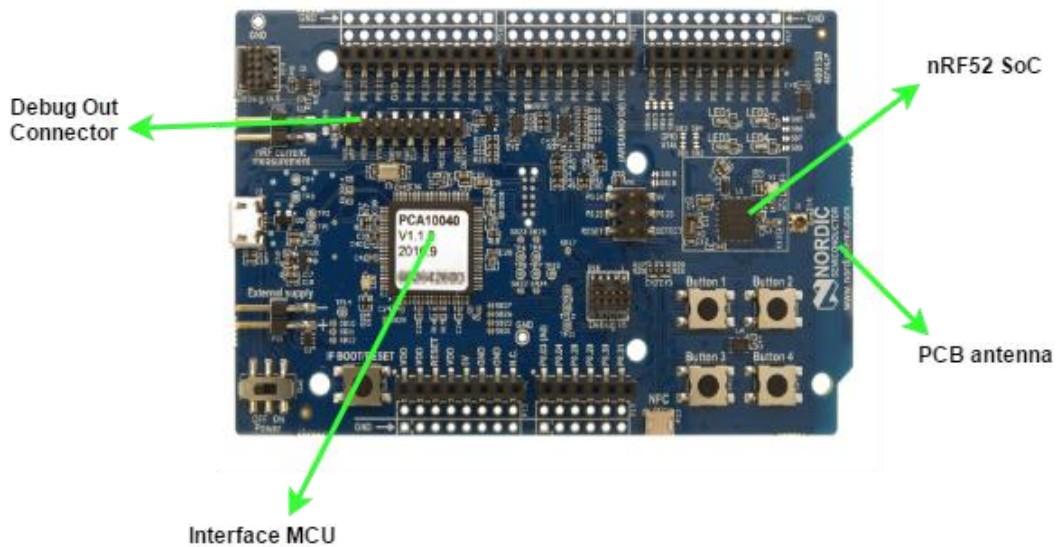


Figure 3.3: Official nRF52832 Development Kit used for initial firmware development and breakout board/prototype programming [31].

### 3.2.1 SparkFun nRF52832 Breakout

After developing the majority of the project firmware using the official development kit, it was important to explore the hardware behavior using a smaller breakout board. The nRF52 SoC is small enough that bread boarding it using a socket was not the best approach when designing prototype hardware. Instead, a breakout board could be used for circuit design to catch any voltage or current issues that could arise when powering the *BLE* transceiver in parallel to other circuit components such as the touch sensor and audio transducer in the medication smart cap.

However, the intention was still to develop a PCB prototype for both the smart cap and the item tracker without the use of a third party module or breakout board. Designing these custom prototypes would entail hand soldering the nRF52 SoC. Since experience with antenna design was limited, it was desirable to find an open source third party breakout with available schematic and board files for a free PCB design software. This breakout was chosen since it comes with Eagle schematic and PCB files. SparkFun also maintains a significant Eagle library of RF components including different style PCB antennas and RF transceiver footprints.

As explained in more detail in later sections, it was possible to program this breakout board using the debug output header on the official development kit shown in Figure 3.3. Figure 3.4 shows the SparkFun nRF52832 breakout board:

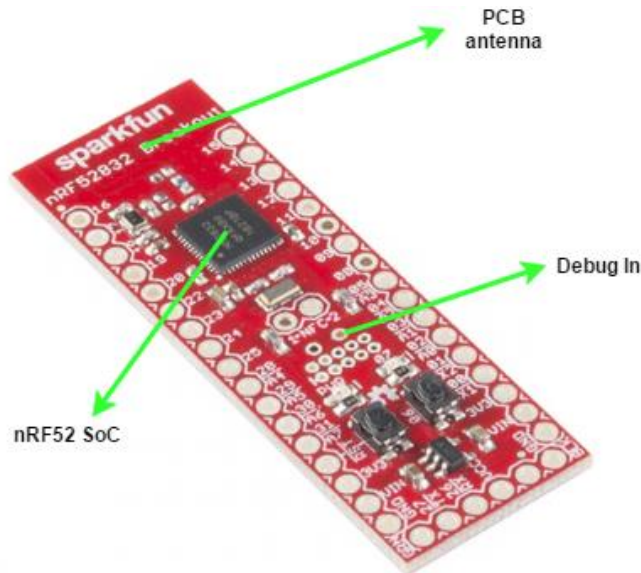


Figure 3.4: SparkFun nRF52832 Breakout board [32].

### 3.2.2 IMM-NRF52832 Micro-Module

In the process of designing and assembling a PCB prototype, it was extremely difficult to hand solder the QFN48 even when using a stencil and solder paste. Therefore, the prototype PCB transitioned from just having the bare chip to having a small breakout or module that could be easily soldered onto it. However, the previously tested 52 mm x 17 mm SparkFun board was too large and contained too many components that were not required for the final product. Therefore, a smaller module was sought that could be easily hand soldered and that already contained an antenna.

The IMM-NRF52832 micro-module was perhaps one of the best documented and supported modules available in market and is also officially recognized as a reliable third party hardware by Nordic Semiconductor. At its small size of 23 mm x 17 mm, the module would not take too much space on the final prototype PCB and could be easily hand soldered with its 1.27 mm pitch surface mount pads. This module

also exposes all 32 GPIO pins along with pins necessary for program flashing. Figure 3.5 shows the IMM-NRF52832 Micro-module.



*Figure 3.5: IMM-NRF52832 Micro-Module [33].*

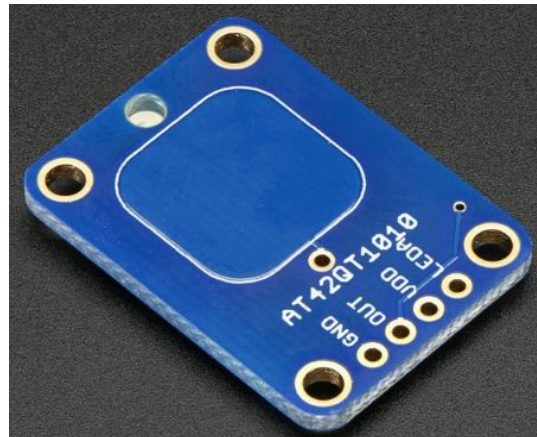
### **3.2.3 Standalone Momentary Capacitive Touch Sensor Breakout – AT42QT1010**

When evaluating the kind of sensor that would be most appropriate for the smart medication cap application, several ideas arose. The first was to have a conductive surface on the parts of the cap and bottle that make contact with each other. The conductive surface on the cap would be connected to a general purpose I/O pin on the nRF52 while the conductive surface on the bottle would be grounded. This way, opening and closing the medication cap would be equivalent to pressing a button. However, this concept was dismissed since extensive modifications to the materials of the bottle would have to be made and production costs would increase.

The second idea was to turn a portion of the cap into a touch sensor. Therefore, touch input is recorded every time a user removes the cap. Achieving this functionality was easiest using a simple capacitive touch sensor whose sensitivity can be easily altered with parameters such as electrode size, capacitance or thickness of overlying panel materials [34]. The SDK for the nRF52 provides a capacitive touch sensor library using the SoC's analog input pins and the comparator module. However, as explained in errata 84 [31] for the nRF52832, capacitive sensing using this method is not viable in real products since

it will only work reliably at room temperature. This issue is related to the faulty programmable current source ISOURCE, which is used by the comparator for capacitive touch sensing [35].

Temperature performance is particularly important for a medication device since medicine could be stored in specific environments outside room temperature. An easy solution was encountered with Adafruit's momentary capacitive touch sensor breakout pictured in Figure 3.6.



*Figure 3.6: Adafruit's Momentary Capacitive Touch Sensor Breakout - AT42QT1010*

This capacitive touch breakout uses ATMEL's AT42QT1010 digital burst mode, charge-transfer sensor IC, which is capable of detecting near-proximity or touch. With a proper electrode and circuit design, the digital IC will sense a touch or proximity field for several centimeters through any dielectric and it is designed for any interface where a button or switch would be used [34]. Therefore, with the AT42QT1010, it would be relatively easy to integrate capacitive touch into the smart medication cap device without the use of the nRF52's comparator. Since it is analogous to a button, using simple library button functions from the SDK would also be sufficient to read in touch input, simplifying firmware development. Further detail on the specifications of the AT42QT1010 and how it works will be discussed in the hardware design section for the smart medication cap prototype.

### 3.3 Setting up the Development Environment

Given that expensive licenses for IDEs such as Keil or IAR Embedded workbench were not available at this academic setting, it was desirable to set up a free development environment such as ARM Eclipse with ARM GCC compiler. As briefly mentioned, in order to achieve such a setup, Nordic provides what seems to be a comprehensive tutorial [36]. However, after multiple attempts at properly setting up ARM Eclipse with ARM GCC compiler and J-Link Debugger, no success was achieved. It was thought appropriate to continue with firmware development using just a text editor along with ARM GCC compiler tools.

Although this was not the most elegant solution for development, it still provided the capability required to develop the firmware for both the smart medication cap. Yet the setup was limited since no comprehensive debugging was available. Important debugging capabilities such as direct access to registers or variable tracking were not possible which increased troubleshooting time when programs would compile properly but malfunction.

Nonetheless, using a text editor with ARM GCC compiler proved sufficient throughout the project. Below are described the several tools that were required for firmware development and board programming.

#### 3.3.0 GNU Toolchain for ARM Cortex-M (ARM GCC)

The GNU Embedded Toolchain for ARM is an open source set of tools for C, C++ and Assembly programming targeted to ARM Cortex-M and Cortex-R processors. It includes the GNU compiler (GCC) and it is available free of charge for embedded firmware development on Windows, Linux and Mac OS X operating systems. After downloading and installing the latest version, the path to the toolchain was added to the PATH system variables of a Windows machine in order to run toolchain executables from any directory using a bash command line shell.



### 3.3.1 nRF5x Software Development Kit v12.0.0

The nRF5 Software Development Kit (SDK) is a zip file provided by Nordic which has extensive resources for firmware development including SoC drivers, libraries, proprietary radio protocols and several versions of the SoftDevice required for *BLE* applications. It also includes multiple example projects tailored to run on Nordic's nRF5x Development Kits. Each example folder contains a *makefile* for compiling projects using ARM GCC. The inclusion of makefiles in example and template projects provided all directives necessary for compiling and linking applications and flashing the precompiled SoftDevice onto development and prototype boards.

In order to program using makefiles and the ARM GCC toolchain, it was first necessary to edit the *Makefile.windows* file on the `<SDK>/components/toolchain/gcc` folder path in the SDK. This file includes the path to the ARM GCC toolchain, its version and prefix as shown in Figure 3.7:

```
1 GNU_INSTALL_ROOT := C:/GNU_Tools_ARM_Embedded/5.4_2016q3
2 GNU_VERSION := 5.4.1
3 GNU_PREFIX := arm-none-eabi
```

*Figure 3.7: Contents of Makefile.windows file.*

### 3.3.2 GNU Make

Another required tool for compiling and building projects based on makefiles is the GNU *make* tool. This tool controls the generation of executable files from a program's source files by using the detailed instructions for building and linking included under a makefile. Once GNU make is downloaded, installed and included as a PATH system variable, it can be used to compile SDK examples and template projects on the bash terminal. The output given after compiling the smart medication cap software using make is shown in Figure 3.8.

```
MINGW64:/c:/Users/Maria/Documents/MQP/nRF5_SDK_12.0.0_12f24da (1)/examples/ble_periphe...
Maria@Strider MINGW64 ~/Documents/MQP/nRF5_SDK_12.0.0_12f24da (1)/examples/ble_periphe...
/ble_app_smartcap2/pca10040/s132/armgcc
$ make
makefile:222: Cannot find include folder: ../../../../config/ble_app_template_pca10040_s132
makefile:222: Cannot find include folder: ../../../../config
Compiling file: nrf_log_backend_serial.c
Compiling file: nrf_log_frontend.c
Compiling file: app_gpiote.c
Compiling file: app_button.c
Compiling file: app_error.c
Compiling file: app_error_weak.c
Compiling file: app_timer.c
Compiling file: app_util_platform.c
Compiling file: hardfault_implementation.c
Compiling file: nrf_assert.c
Compiling file: nrf_drv_clock.c
Compiling file: nrf_drv_common.c
Compiling file: nrf_drv_gpiote.c
Compiling file: nrf_drv_pwm.c
Compiling file: main.c
Compiling file: lb_service.c
Compiling file: sensor_service.c
Compiling file: ble_advdata.c
Compiling file: ble_advertising.c
Compiling file: ble_conn_params.c
Compiling file: ble_conn_state.c
Compiling file: ble_srv_common.c
Assembling file: gcc_startup_nrf52.S
Compiling file: system_nrf52.c
Compiling file: softdevice_handler.c
Linking target: _build/nrf52832_xxaa.out

  text   data    bss     dec     hex filename
 15708   164    708   16580   40c4 _build/nrf52832_xxaa.out

Preparing: _build/nrf52832_xxaa.hex
Preparing: _build/nrf52832_xxaa.bin
```

Figure 3.8: Terminal output of make showing linked source files and output hex file.

As shown in Figure 3.8, the final output of the make command is the hex file *nrf52832\_xxaa.hex* which can be loaded onto the development, breakout and prototype boards using the programming commands described next.

### 3.3.3 nRF5x Command Line Tools

The nRF5x Command Line tools are freely provided by Nordic Semiconductor and are used for development, programming and some limited debugging of nRF5x SoCs. The most relevant component of the command line tools is the *nrfjprog* executable. This command line tool is used to program the nRF52 through SEGGER J-Link programmers and debuggers. As mentioned in Section 3.2.0, the development board includes an interface MCU that is factory preloaded with SEGGER J-Link OB. The command line tools can be installed and added to the PATH system variables for use in all directories.

The specific commands required to program the nRF52 SoC are shown in Figure 3.9 in sequence. For erasing memory, programming and running, the -f command is used to specify the nRF5x SoC family used.

```
// erasing all user available program flash memory
nrfjprog -f nRF52 -e

// flashing the precompiled binary image of SoftDevice onto SoC flash memory
make flash_softdevice

// programing the compiled HEX file into device
nrfjprog -f nRF52 --program _buld/nrf52832_xxaa.hex

// resetting the SoC and running the loaded program
nrfjprog -f nRF52 -r
```

Figure 3.9: nrfjprog command line tools for programming the nRF52 SoC with BLE applications.

### 3.3.4 Programming nRF52 Modules and Prototype Boards: Serial Wire Debug

Since the official nRF52 Development Kit includes an on-board interface MCU pre-programmed with SEGGER J-Link software, it is sufficient to program the SoC using a micro-USB cable. However, both the SparkFun breakout and the IMM-NRF52832 module contain no such interface MCU due to their size. Fortunately, it is possible to easily program modules and prototype boards using the interface MCU and the Debug Out Connector on the nRF52 Development Kit. The Debug Out Connector is shown in Figure 3.3 and contains the Serial Wire Debug (SWD) pins outlined in Figure 3.10.

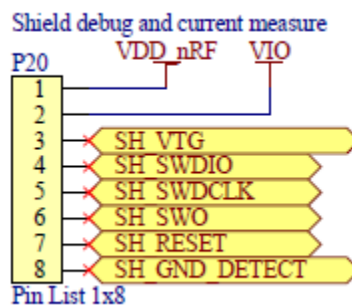
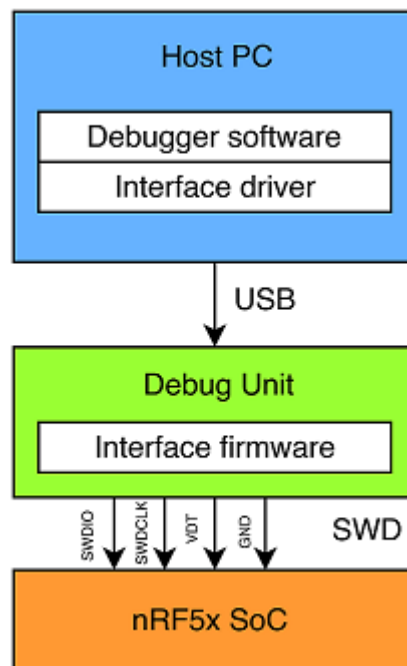


Figure 3.10: Serial Wire Debug (SWD) pins on Debug Out Connector P20 for SWD programming [37].

Of the eight shown in Figure 3.10, only the SH\_VTG, SH\_SWDIO, SH\_SWDCLK, and SH\_GND\_DETECT pins are required for SWD programming of external nRF52 SoCs. The SH\_VTG pin is connected to the external board's power supply while SH\_GND\_DETECT shares a common ground with the external board. The SH\_SWDIO and SH\_SWDCLK pins are the data line and clock line respectively. It is important to note that all pins in header P20 correspond to the interface MCU on the nRF52 Development Kit.

When the external board to be programmed is powered, the interface MCU will detect an external supply through the SH\_VTG pin and will then program the target SoC on the external board. The only voltage supported by external programming is 3.0 V [31]. Figure 3.11 shows a graphical description of the SWD interface between the nRF52 Development Kit (Debug Unit) and an external nRF52 SoC.



*Figure 3.11: SWD programming interface between nRF52 Development Kit (Debug Unit) and external nRF52 SoC [37].*

Figure 3.12 shows the physical setup for programming the IMM-NRF52832 module on a breakout board using SWD programming. The red jumper wire connects the external 3.0 V supply to SH\_VTG while the black wire ties SH\_GND\_DETECT to ground. The yellow and green wires connect the SH\_SWDIO

and SH\_SWCLK pins between the kit and module breakout respectively. The same setup was used when programming the SparkFun breakout and prototype boards for the smart medication cap.

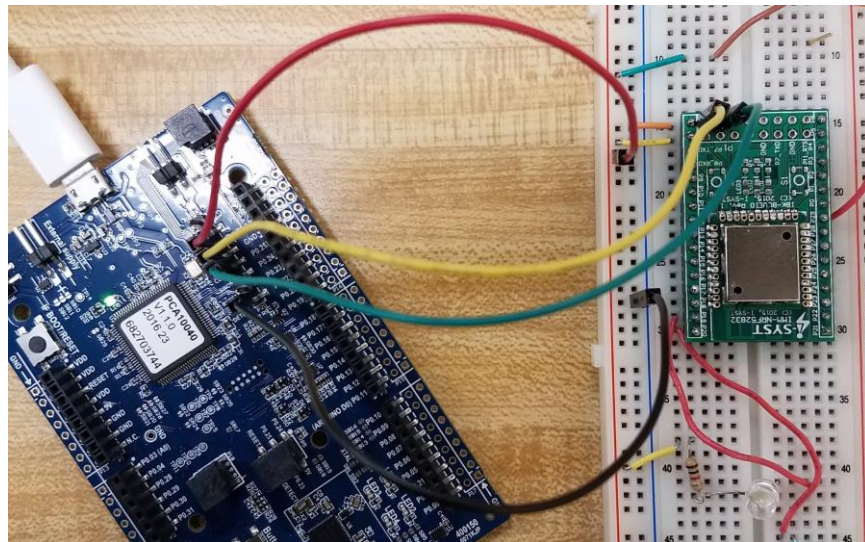


Figure 3.12: Physical setup for SWD programming of the IMM-NRF52832 micro-module breakout board.

### 3.4 Overview of Smart Medication Cap Firmware

This section contains an overview of the most relevant components of the smart medication cap firmware including *BLE* GAP, GATT, advertisement and connection parameters, and device functionality through services and characteristics. The source files for the firmware are attached in a zip file to the project's main title page. However, important sections of code discussed in this section will be included here.

#### 3.4.0 Initializing the SoftDevice in main.c

To initialize and enable the SoftDevice, several modules from the SoftDevice Manager API are required which allow the developer to set parameters such as memory isolation, clock source (internal RC oscillator or external crystal oscillator) and power management. Below is the segment of code that initializes and enables the SoftDevice for *BLE* applications:

```
// Function for initializing the BLE stack.  
// Initializes the SoftDevice and the BLE event interrupt.  
static void ble_stack_init(void)  
{
```

```

uint32_t err_code;

nrf_clock_lf_cfg_t clock_lf_cfg = NRF_CLOCK_LFCLKSRC;

// Initialize the SoftDevice handler module.
SOFTDEVICE_HANDLER_INIT(&clock_lf_cfg, NULL);

ble_enable_params_t ble_enable_params;
err_code = softdevice_enable_get_default_config(CENTRAL_LINK_COUNT,
                                                PERIPHERAL_LINK_COUNT,
                                                &ble_enable_params);

APP_ERROR_CHECK(err_code);

//Check the ram settings against the used number of links
CHECK_RAM_START_ADDR(CENTRAL_LINK_COUNT, PERIPHERAL_LINK_COUNT);

// Enable BLE stack.
#if (NRF_SD_BLE_API_VERSION == 3)
    ble_enable_params.gatt_enable_params.att_mtu = NRF_BLE_MAX_MTU_SIZE;
#endif
err_code = softdevice_enable(&ble_enable_params);
APP_ERROR_CHECK(err_code);

// Subscribe for BLE events.
err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
APP_ERROR_CHECK(err_code);
}

```

The clock source `NRF_CLOCK_LFCLKSRC` is defined in a library header file specific to the nRF52832 development kit circuit revision (`pca10040.h`) and is set to `NRF_CLOCK_LF_SRC_XTAL` which is the external 32.768 kHz crystal oscillator. Use of the external crystal oscillator is recommended since the use of the internal RC oscillator consumes higher current when necessary calibrations are performed [18], [38]. Once a clock source is defined, the `SOFTDEVICE_HANDLER_INIT` macro is used to initialize the softdevice handler module (`softdevice_handler.c`). This macro takes as input the `CENTRAL_LINK_COUNT` and `PERIPHERAL_LINK_COUNT` parameters and an empty `ble_enable_params` structure for fetching the default SoftDevice configuration. The central and peripheral link count parameters are defined in `main.c` and are set to 0 and 1 respectively since this device is meant to take the role of peripheral only. With the nRF52 it is possible for a device to switch between central and peripheral roles, but for this application, that was not necessary.

`SOFTDEVICE_HANDLER_INIT` will then return the same `ble_enable_params` structure containing the default SoftDevice configuration which will work for a majority of applications. The most relevant default parameters in `ble_enable_params` are described below [31]:

- ***gatts\_enable\_params.attr\_tab\_size***: this parameter determines the attribute table size in bytes. The default size is 0x580 bytes which is more than enough for this particular application.
- ***gap\_enable\_params.central\_conn\_count***: determines the number of connections acting as central (set to `CENTRAL_LINK_COUNT = 0` for this application).
- ***gap\_enable\_params.periph\_conn\_count***: determines the number of connections acting a peripheral (set to `PERIPHERAL_LINK_COUNT = 1` for this application).
- ***common\_enable\_params.vs\_uuid\_count***: determines the maximum number of 128-bit, vendor specific UUID bases to allocate. This parameter is set to one in this application since both services in the Smart Medication Tracker profile (Figure 3.2) share the same 128-bit base UUID.
- ***gatt\_enable\_params.att\_mtu***: determines the maximum size of an ATT packet the SoftDevice can send or receive. The default size is set to 23 bytes which is the maximum data transfer allowed in *BLE*.

Before enabling the SoftDevice, the `CHECK_RAM_START_ADDR` macro is used to check the specific RAM start address requirements for the SoftDevice. Since the number of central and peripheral links affects RAM resources used by the SoftDevice, the parameters `CENTRAL_LINK_COUNT` and `PERIPHERAL_LINK_COUNT` are passed to the macro [31].

Finally, the SoftDevice can be enabled by passing *ble\_enable\_params* structure to the function *softdevice\_enable()*. The application must then be subscribed to *BLE* events by calling the function *softdevice\_ble\_evt\_handler\_set(ble\_evt\_dispatch)*. The function *ble\_evt\_dispatch()* contains several *BLE* event handlers and must be passed to the event subscription function.

### 3.4.1 Initializing GAP and Advertisement Parameters

As mentioned in Section 2.3.2, GAP controls connections and advertising in *BLE* and it is essentially the component of the protocol stack that makes a device visible to scanning devices. The following segment of code is used to set the necessary GAP parameters used by the smart medication cap application.

```

// Function for the GAP initialization.
// This function sets up all the necessary GAP (Generic Access Profile) parameters of the
// device including the device name, appearance, and the preferred connection parameters.
static void gap_params_init(void)
{
    uint32_t          err_code;
    ble_gap_conn_params_t  gap_conn_params;
    ble_gap_conn_sec_mode_t  sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode,
                                          (const uint8_t *)DEVICE_NAME,
                                          strlen(DEVICE_NAME));

    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;

    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}

```

The memory reference to the parameter *sec\_mode* is passed to the macro `BLE_GAP_CONN_SEC_MODE_SET_OPEN()` to establish the security of connections made by this device as open. The updated *sec\_mode* parameters is then passed to the function *sd\_ble\_gap\_device\_name\_set()* along with parameter `DEVICE_NAME` to set the GAP device name. Finally, after setting appropriate memory for the structure *gap\_conn\_params* of type *gap\_con\_params\_t*, the parameters for maximum and min connection interval, slave latency, and connection supervision timeout are set. These parameters are defined in `main.c` as `MIN_CONN_INTERVAL`, `MAX_CONT_INTERVAL`, `SLAVE_LATENCY`, and `CONN_SUP_TIMEOUT` and are set to the values shown in table 3-1.



**Table 3-1: GAP initialization parameters for smart medication cap.**

| Parameter         | Value |
|-------------------|-------|
| MIN_CONN_INTERVAL | 0.2 s |
| MAX_CONN_INTERVAL | 0.4 s |
| SLAVE_LATENCY     | 5     |
| CONN_SUP_TIMEOUT  | 5 s   |

As seen in Table 3-1, the connection interval range was set to be between 0.2 s and 0.4 s. The interval is therefore in the faster end of the allowed 1.5 ms to 4 s range. However, due to the slave latency parameter, the *effective* connection interval is in fact longer. For this application, the slave latency was set to five, i.e., the slave device can ignore five consecutive connection events from the master. The equation below is used to determine the *effective* connection interval [39]:

$$Eff. Conn. Interval = Conn. Interval \times (1 + Slave Latency)$$

Therefore, with a slave latency of five, and assuming that the connection interval is at its minimum value of 200 ms, the effective connection interval is 1.2 s. In a situation in which no data is being sent from the slave, the slave will only transmit every 1.2 s. Having the effective connection interval longer than a second can significantly reduce the power consumption of both slave and master. Finally, the connection supervisory timeout, which must be longer than the effective connection interval, was set to 5 s. This means that checking for disconnection events due to poor range or signal interference will happen every 5 seconds. Once these parameters are set, the SoftDevice handler function `sd_ble_gap_ppcp_set(&gap_conn_params)` is used to set the GAP parameters.

Next, the advertising parameters for the smart cap application were set using the code segment below:

```

// Function for initializing the Advertising functionality.
static void advertising_init(void)
{
    uint32_t          err_code;
    ble_advdata_t     advdata;
    ble_adv_modes_config_t options;

    // Build and set advertising data.
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type          = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance = false;
    advdata.flags              = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;;
    advdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids)/sizeof(m_adv_uuids[0]);
    advdata.uuids_complete.p_uuids = m_adv_uuids;

    // Declaring and Instantiating Scan Response
    // Adding scan response increases advertising space
    ble_advdata_t srdata;
    memset(&srdata, 0, sizeof(srdata));

    // addign UUID to scan respose
    srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids_sr)/sizeof(m_adv_uuids_sr[0]);
    srdata.uuids_complete.p_uuids = m_adv_uuids_sr;

    memset(&options, 0, sizeof(options));
    options.ble_adv_fast_enabled = true;
    options.ble_adv_fast_interval = APP_ADV_INTERVAL;
    options.ble_adv_fast_timeout = APP_ADV_TIMEOUT_IN_SECONDS;
    options.ble_adv_slow_enabled = true;
    options.ble_adv_slow_interval = APP_ADV_SLOW_INTERVAL;
    options.ble_adv_slow_timeout = APP_ADV_SLOW_TIMEOUT_IN_SECONDS;

    err_code = ble_advertising_init(&advdata, &srdata, &options, NULL, NULL);
    APP_ERROR_CHECK(err_code);
}

```

First, the advertising data is saved into the declared structure *advdata* of type *ble\_advdata\_t*. The first parameters set in this struct is the *name\_type* which is set to `BLE_ADVDATA_FULL_NAME` to include the full name of the peripheral. The appearance of the device is not included by setting *include\_appearance* to `false`. Then the advertising data *flags* is set to `BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE` to indicate that the device only supports *BLE* and is set to general discoverable mode. A complete count of the UUIDs and the UUIDs themselves are then included in the advertising data. Only the 128-bit UUID for the tracker service is included in the advertising data due to size constraints.

In order to advertise the 128-bit sensor service UUID as well, the scan response data is used by declaring the *srdata* structure of type *srdata*. The only parameters set in *srdata* were the UUID count and

actual UUID for the sensor service. Then, the actual advertising parameters were set using the *options* struct of type *ble\_adv\_modes\_cofrig\_t*. The parameters set in the *options* struct are outlined below:

- ***options.ble\_adv\_fast\_enabled***: Set to true in order to enable the peripheral device to advertise in fast mode. This mode allows for smaller advertising intervals.
- ***options.ble\_adv\_fast\_interval***: Set to APP\_ADV\_INTERVAL which is defined in main.c as 187.5 ms.
- ***options.ble\_adv\_fast\_timeout***: Set to APP\_ADV\_TIMEOUT\_IN\_SECONDS which is defined in main.c as 30 s.
- ***options.ble\_adv\_slow\_enabled***: Set to true in order to enable the device to advertise in slow mode. This mode allows for longer advertising intervals.
- ***options.ble\_adv\_slow\_interval***: Set to APP\_ADV\_SLOW\_INTERVAL which is defined in main.c as 1 s.
- ***options.ble\_adv\_slow\_timeout***: Set to APP\_ADV\_SLOW\_TIMEOUT\_IN\_SECONDS which is defined in main.c as zero. Setting this parameter to zero allows for indefinite advertising in slow mode.

As described in the advertising parameters, both fast and slow advertising are enabled for this peripheral device. When the device first advertises, it will do so in fast mode at intervals of 187.5 ms to enable a faster connection. After 30 s, fast advertising will timeout and the device will switch to slow advertising at intervals of 1 s. Since it is required that the smart cap device advertise indefinitely, increasing the length of the advertising interval can significantly decrease current consumption by the device. The parameters for fast and slow advertising are summarized in Table 3-2.

Table 3-2: Fast and slow advertising parameters for smart medication cap.

|                             | Fast Advertising | Slow Advertising |
|-----------------------------|------------------|------------------|
| <b>Advertising Interval</b> | 187.5 ms         | 1 s              |
| <b>Advertising Timeout</b>  | 30 s             | No timeout       |

In order to initialize fast advertising, the SoftDevice handler function *sd\_ble\_gap\_adv\_star()* function is called with fast advertising interval and timeout as arguments. Once fast advertising timeout occurs, the function returns the *BLE* event *BLE\_GAP\_EVT\_TIMEOUT*, which signals the start of slow advertising. To start slow advertising, the same *sd\_ble\_gap\_adv\_start()* function is used but with slow advertising interval and timeout as arguments.

### 3.4.2 Overview of Services and Characteristics

The *track\_service.h* and *sensor\_service.h* files included in the attached zip file contain the necessary definitions and data structures required in the files *track\_service.c* and *sensor\_service.c* to initialize the tracker and sensor services and add the appropriate characteristics. As shown in Figure 3.2, the tracker service characteristics include an *LED pin state* characteristic, a *PWM signal* characteristic, and a *Button state* characteristic while the sensor service includes the *Sensor state* and *Temperature value* characteristics. The operation properties of each characteristic are shown in Table 3-3.

**Table 3-3: Operation properties for characteristics in tracker and sensor services.**

| Characteristic Value      | Operation Properties |
|---------------------------|----------------------|
| <b>Tracker Service</b>    |                      |
| <i>LED PIN STATE</i>      | read/write           |
| <i>PWM SIGNAL</i>         | read/write           |
| <i>BUTTON STATE</i>       | read/notify          |
| <b>Sensor Service</b>     |                      |
| <i>TOUCH SENSOR STATE</i> | read/notify          |
| <i>TEMPERATURE VAL.</i>   | read/notify          |

Therefore, a client device is able to read and write the characteristic value for *LED pin state* and *PWM signal* characteristics while it is able to read the characteristic value and receive notifications from the *BUTTON state*, *TOUCH SENSOR state* and *TEMPERATURE VAL.* characteristics. The segment of code below from *track\_service.c* shows the function *led\_char\_add()* for declaring the *LED pin state* characteristic:

```
// Function for adding LED characteristic to Service
// param[in] p_track_service Tracker Service structure.
// param[in] p_lb_init LED Button PWM Service init structure.
static uint32_t led_char_add(ble_track_t * p_track_service, const ble_track_init_t *
p_track_init)
{
    // Add a custom characteristic UUID
    uint32_t err_code;
    ble_uuid_t char_uuid;
    ble_uuid128_t base_uuid = BLE_UUID_BASE_UUID;
    char_uuid.uuid = BLE_UUID_CHARACTERISTIC_LED;
    err_code = sd_ble_uuid_vs_add(&base_uuid, &char_uuid.type);
    APP_ERROR_CHECK(err_code);

    // Add read/write properties to our characteristic
    ble_gatts_char_md_t char_md;
    memset(&char_md, 0, sizeof(char_md));
    char_md.char_props.read = 1;
    char_md.char_props.write = 1;
}
```

```

// Configuring Client Characteristic Configuration Descriptor metadata and add to char_md
structure
ble_gatts_attr_md_t cccd_md;
memset(&cccd_md, 0, sizeof(cccd_md));

// Configure the attribute metadata
ble_gatts_attr_md_t attr_md;
memset(&attr_md, 0, sizeof(attr_md));
attr_md.vloc = BLE_GATTS_VLOC_STACK;
attr_md.rd_auth = 0;
attr_md.wr_auth = 0;
attr_md.vlen = 0;

// Set read/write security levels to our characteristic
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);

// Configure the characteristic value attribute
ble_gatts_attr_t attr_char_value;
memset(&attr_char_value, 0, sizeof(attr_char_value));
attr_char_value.p_uuid = &char_uuid;
attr_char_value.p_attr_md = &attr_md;

// Set characteristic length in number of bytes
attr_char_value.max_len = sizeof(uint8_t);
attr_char_value.init_len = sizeof(uint8_t);
attr_char_value.p_value = NULL;

// Add led characteristic to the service
err_code = sd_ble_gatts_characteristic_add(p_track_service->service_handle,
                                           &char_md,
                                           &attr_char_value,
                                           &p_track_service->led_char_handles);

APP_ERROR_CHECK(err_code);

return NRF_SUCCESS;
}

```

In this segment of code, the function *sd\_ble\_uuid\_vs\_add()* is first used to add the vendor specific characteristic UUID (BLE\_UUID\_CHARACTERISTIC\_LED) to the *BLE* attribute table based on the application's base UUID. Then the characteristic metadata structure, *char\_md* (of type *ble\_gatts\_char\_md\_t*), is used to set the characteristic properties for read and write by setting the *char\_props.read* and *char\_props.write* data fields to true. Next, the attribute metadata structure (of type *ble\_gatts\_attr\_md\_t*) field *attr\_md.vloc* is used to set the attribute value location. Setting this field to BLE\_GATTS\_VLOC\_STACK saves the attribute value in stack memory and no user memory is required. Next, the macro BLE\_GAP\_CONN\_SEC\_MODE\_SET\_OPEN sets the open security levels of the characteristic as read and write.

The structure *attr\_char\_value* (of type *ble\_gatts\_attr\_t* for setting GATT attribute) then contains a pointer to the characteristic UUID (*attr\_char\_value.p\_uuidi*) and a pointer to the previously defined attribute metadata (*attr\_char\_value.p\_attr\_md*). Its three other fields also contain an initial characteristic value offset and its maximum value in bytes, and a pointer to the actual value. Since the attribute value is saved in stack memory, a user is not allowed to specify location and its pointer must be set to NULL. Finally, the SoftDevice handling function *sd\_ble\_characteristic\_add()* is used to add the characteristic to the attribute table using the defined structures for attribute metadata and characteristic attribute values, and a pointer to the structure where the assigned values will be stored (declared in *track\_service.h*).

The function for declaring the *PWM signal* characteristic, *pwm\_char\_add()* is in essence identical to *led\_char\_add()* since they have the same read and write operation properties and read and write open security levels. On the other hand, the functions for adding characteristics in both services that send a notification to the central device contain a small difference. For example, the function *temp\_char\_add()* for adding the *TEMPERATURE value* characteristic contains the additional lines of code:

```
// Configuring Client Characteristic Configuration Descriptor metadata and add to char_md structure
ble_gatts_attr_md_t cccd_md;
memset(&cccd_md, 0, sizeof(cccd_md));
cccd_md.vloc = BLE_GATTS_VLOC_STACK;
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.write_perm);

// Add read/write properties to our characteristic
ble_gatts_char_md_t char_md;
memset(&char_md, 0, sizeof(char_md));
char_md.char_props.read = 1;
char_md.char_props.notify = 1;
char_md.p_cccd_md = &cccd_md;
```

This segment of code sets the memory data field for the *cccd\_md* structure of type *ble\_gatts\_attr\_md\_t* to configure the Client Characteristic Configuration Descriptor (CCCD) metadata. CCCD is required for a GATT client to control what kinds of packets the GATT server can send to it. Therefore, the peripheral device can only send a notification if the client has written a one to the CCCD for the respective characteristic [40]. The macro *BLE\_GAP\_CONN\_SEC\_MODE\_SET()* is then used to set the open security permissions of the CCCD metadata as read and write. The characteristic metadata for this

*TEMPERATURE* value characteristic will then set possible operations to read and notify and will also contain a pointer to the *cccd\_md* structure. This same procedure is used for adding the *SENSOR state* and *BUTTON state* characteristics since they too are required to send the client notifications.

Finally, the *track\_service.c* and *sensor\_service.c* files contain functions that are called in main for initializing the services. For example, the function *sensor\_srv\_init()*, from *sensor\_service.c*, is shown below:

```
// Function for initiating the sensor service.
// param[in] p_sensor_service    sensor Service structure.
uint32_t sensor_srv_init(ble_sensor_t * p_sensor_service)
{
    uint32_t err_code;    // return error codes from softdevice and library functions

    //Declare 16 bit service and 128 bit base UUIDs and add them to BLE stack table
    ble_uuid_t service_uuid;
    ble_uuid128_t base_uuid = BLE_UUID_BASE_UUID;
    service_uuid.uuid = BLE_UUID_SENSOR_SERVICE;
    err_code = sd_ble_uuid_vs_add(&base_uuid, &service_uuid.type);
    APP_ERROR_CHECK(err_code);

    // Add service to the stack
    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
                                        &service_uuid,
                                        &p_sensor_service->service_handle);

    APP_ERROR_CHECK(err_code);

    // Calling the function sensor_char_add() to add new characteristic to the service
    err_code = sensor_char_add(p_sensor_service);
    APP_ERROR_CHECK(err_code);

    // Calling the function temp_char_add() to add a new characteristic to the service
    err_code = temp_char_add(p_sensor_service);
    APP_ERROR_CHECK(err_code);

    return NRF_SUCCESS;
}
```

This function first calls *sd\_ble\_uuid\_vs\_add()* to add the service UUID to the *BLE* stack attribute table based on the applications base UUID. Again both the service UUID (*BLE\_UUID\_SENSOR\_SERVICE*) and the base UUID (*BLE\_UUID\_BASE\_UUID*) are defined in the *sensor\_service.h* file. Next, the SoftDevice handling function *sd\_ble\_gatt\_service\_add()* is called to add the service to the *BLE* stack by specifying the memory location of the service UUID and the service handle defined in *sensor\_service.h*. The service adder function also requires the developer to specify the GATT server service type. Both the sensor and tracker services for the smart medication device are primary which



is denoted by `BLE_GATTS_SRVC_TYPE_PRIMARY`. After adding the service to the stack, the functions for adding the different service characteristics described previously are then called. Therefore, calling `sensor_srv_init()` in main initializes the service and its respective characteristics.

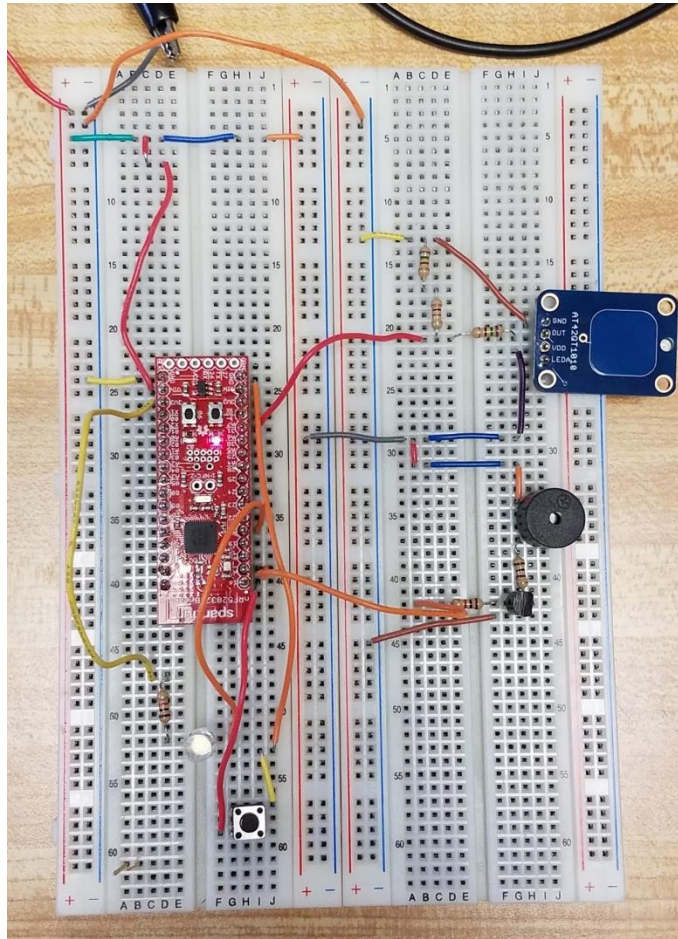
Apart from the service and characteristic adding functions, the `track_service.c` and `sensor_service.c` files also contain other functions for *BLE* event handling including connection, disconnection and writing events. For certain characteristics, the service files also contain functions which, with the help of the SoftDevice handling function `sd_ble_gatts_hvx()`, send the client notifications. The `main.c` file also contains multiple write handlers for characteristics that can be written to by the client. For brevity, these functions will not be discussed in detail but can still be inspected in the attached zip file.

### 3.5 Smart Medication Cap Prototype Hardware

This section gives an overview of the steps taken to design and test the prototype hardware for the smart medication cap device. Initially, the hardware was prototyped using a standard breadboard. The tested design on the breadboard was then translated to a PCB prototype using Eagle PCB design software. Two iterations of hardware design were performed for this device: one in which the nRF52 SoC would be hand soldered onto the prototype board and another in which an nRF52 module with attached antenna would be used.

#### 3.5.0 Initial Hardware Design with Hand Soldered nRF52832

The initial hardware design was tested with the nRF52 Development Kit but then was eventually tested with SparkFun's nRF52 breakout board since it was initially intended for the final PCB prototype to be completely custom with a hand soldered nRF52 SoC. As mentioned in Section 3.2.1, the SparkFun breakout was chosen since it was provided with comprehensive Eagle schematic and PCB files that could be closely followed for circuit design accuracy. This was particularly attractive since the student working on this project had very little experience with antenna design. The breadboarded circuit for the smart medication cap using the SparkFun breakout is shown in Figure 3.14.



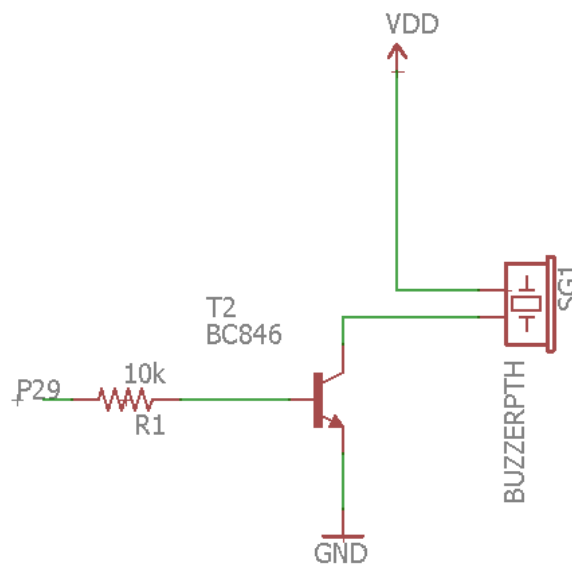
*Figure 3.13: Breadboarded prototype circuit using SparkFun's nRF52 breakout board.*

The breakout board was powered using 3.0 V from the power supply to imitate a 3.0 V coin cell. It is extremely important to note that the nRF52 general purpose I/O (GPIO) pins cannot drive more than 15 mA of current combined. In fact, it is never recommended to drive any components directly from GPIO pins in any microcontroller since they will most likely not provide enough current. Therefore, the LED, PWM buzzer, and capacitive touch sensor IC are all powered with the same 3.0 V input voltage rail as the nRF52 breakout board.

The LED whose state is controlled by the *LED state* characteristic value was a standard T-1 ¼ LED package whose maximum forward current is 20 mA and max forward voltage is 4 V. When powered with 3.0 V, it was possible to achieve around half of the LED's light intensity with a forward current of around 6 mA. Therefore, the LED is connected to  $V_{CC}$  with a current limiting resistor of around 475  $\Omega$  which was

simply calculated using Ohm's laws. Although this only allows for 50% intensity, the LED used in the hardware has an intensity of 7,500 millicandela (mcd) at a 32 degree viewing angle. Hence, the LED is still incredibly bright with this setup and cannot be looked upon directly from above without discomfort.

Next, the PWM buzzer, whose state is controlled by the *PWM signal* characteristic value is also connected to the 3.0 V  $V_{CC}$  rail in parallel to the LED and the nRF52 breakout board. In order to drive the buzzer with a PWM signal generated by the nRF52 SoC, a transistor driver circuit was used. This driver circuit is shown in Figure 3.15.



*Figure 3.14: NPN transistor driver circuit for piezoelectric buzzer/magnetic transducer audio elements.*

In this simple driver circuit, the transistor behaves like a switch which is controlled by the voltage at its base pin. Hence, the control signal for the audio buzzer (a PWM signal from GPIO pin 29) is connected to the base of the transistor. Once the control signal's voltage increases above the transistor's threshold voltage, the transistor will behave as a short circuit between its collector and emitter pins, tying the buzzer straight to ground. However, when the control signal's voltage is below the threshold voltage, the transistor is in cutoff mode and no current flows between collector and emitter. This is a simple technique that allows for the audio buzzer to be powered by the  $V_{CC}$  rail for higher current drive while still being controlled by an nRF52 GPIO pin signal.

Finally, the Adafruit capacitive touch sensor breakout using ATMEL's sensor IC was also powered with the 3.0 V  $V_{CC}$  rail in parallel to the rest of the components. Its digital output pin is then connected to an nRF52 GPIO pin that is controlled using SDK button library functions. This pin was initialized to use an internal pull up resistor, therefore, when the sensor output pin is low, the GPIO pin is shorted to ground, registering an input value of one. However, when the capacitive sensor pad is touched and the output pin goes high, the input GPIO pin is no longer tied to ground and is pulled high, registering a zero. Hence, when the sensor is touched, the smart medication cap will send the central device a value of zero. In addition, when the sensor IC output pin goes high, it does so at the same input voltage of 3.0 V. Therefore, for protection measures, this sensor output voltage was stepped down to the nRF52 GPIO pin's logic high voltage of 2.5 V using a simple voltage divider.

This breadboarded circuit was then translated to a PCB prototype design using Eagle software. Again, this particular design involved hand soldering the nRF52 SoC and was therefore complex since it involved not just the hardware for the external components described previously, but also the specific supporting hardware for the proper functioning of the nRF52 SoC, including a PCB antenna. Fortunately, it was possible to closely follow the provided Eagle schematic and layout provided for the SparkFun breakout board to minimize design flaws. The detailed schematic for this design is included under Appendix 7.2. Figure 3.16 shows the final layout design for this particular PCB prototype.

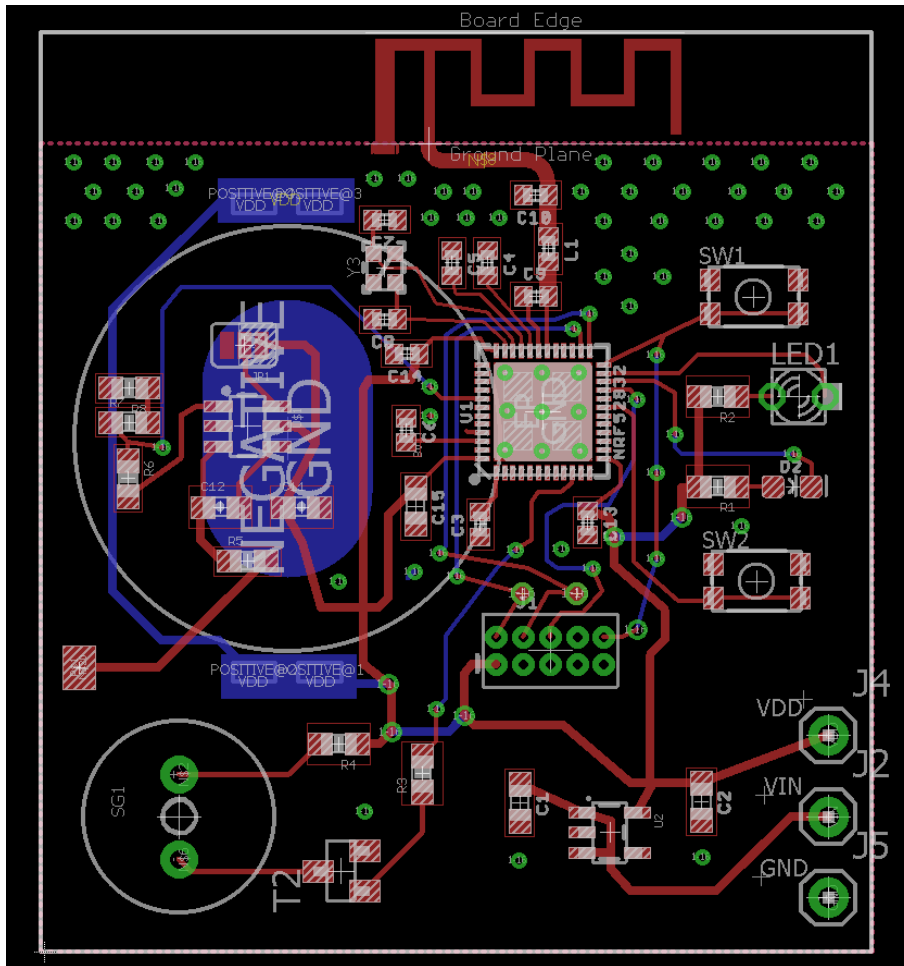


Figure 3.15: PCB layout for complete custom prototype with hand soldered nRF52 SoC.

From the schematic under Appendix 7.2 and the PCB layout shown in Figure 3.15, it can be seen that many more components are included in the design than were originally tested on the breadboarded circuit in Figure 3.14. However, the placement of these components in the final design was carefully mimicked from the SparkFun breakout board to insure proper functionality. This was especially true for the impedance matching network between the SoC antenna pin and the PCB trace antenna. The 15.2 mm trace antenna itself was also provided as a library component under the SparkFun Eagle files. In addition, the extra components include a 3.3 V AP2112K voltage regulator in case the custom board was to be powered using higher voltage through the  $V_{IN}$  pin.

Otherwise, the 39.32 mm by 43.40 mm custom board contains all other components described previously required for the smart medication cap application. In Figure 3.16, the location of the audio

transducer and the driver circuit is clearly visible as is the T-1 ¾ LED and buttons. The ATMEL AT42QT1010 sensor IC and its supporting circuitry were also placed on the top layer of the custom board, opposite the battery holder on the bottom layer. The design for this supporting circuitry for the sensor IC was readily available in the IC’s data sheet as shown in Figure 3.17.

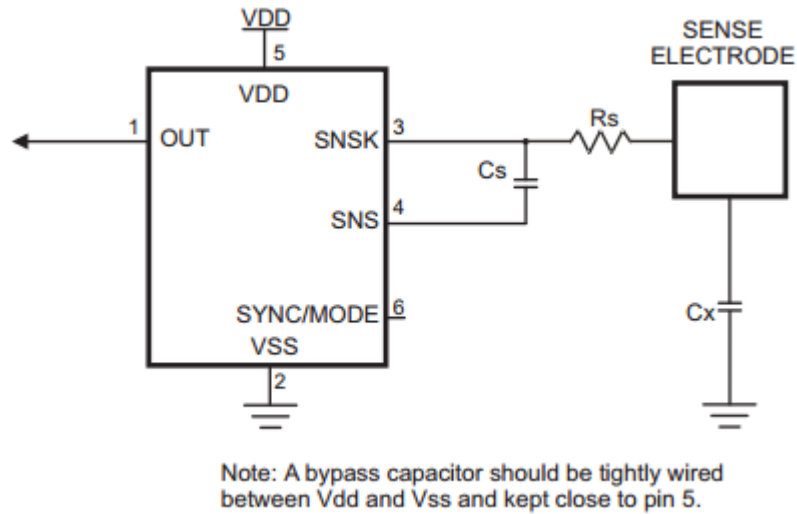


Figure 3.16: ATMEL AT42QT1010 sensor IC circuitry [34].

A few things to keep in mind for the sensor IC circuit in the above figure is the fact that a 0.1 uF bypass capacitor must be wired between  $V_{DD}$  and  $V_{SS}$ . In addition, the value of  $C_s$ , which is typically between 2 and 50 nF, can affect the overall sensitivity of the sensor IC. For example, decreasing the value of this capacitance can significantly reduce sensitivity. For this particular prototype, the value was kept at 10 nF, which is the same used in the Adafruit breakout sensor utilized during testing.

Another important aspect to keep in mind for the PCB design shown in Figure 3.16 is the fact that area where the PCB antenna is traced is completely out of reach for other traces and ground planes. This particular section of the board is also only one layer instead of two. Having ground planes or other traces near the antenna can have a major negative impact on the RF performance of the antenna.

### 3.5.1 Second Hardware Design Using IMM-NRF52832 Micro-Module

The previous PCB design shown in Figure 3.16 had to be discarded since attempting to hand solder the 48 pin QFN package for the nRF52 SoC did not result in success. The idea of outsourcing prototype assembly to an outside company was considered, however, time and budget constraints did not make this solution feasible. The fastest and simplest solution was to redesign the PCB layout in Figure 3.16 to use an nRF52 module instead. As mentioned in Section 3.2.2, the IMM-NRF52832 micro-module was chosen since it already has a PCB antenna and was one of the best documented third party modules available in market. Figure 3.18 shows the breadboarded prototype circuit for the smart medication cap using the IMM-NRF52832 micro-module.

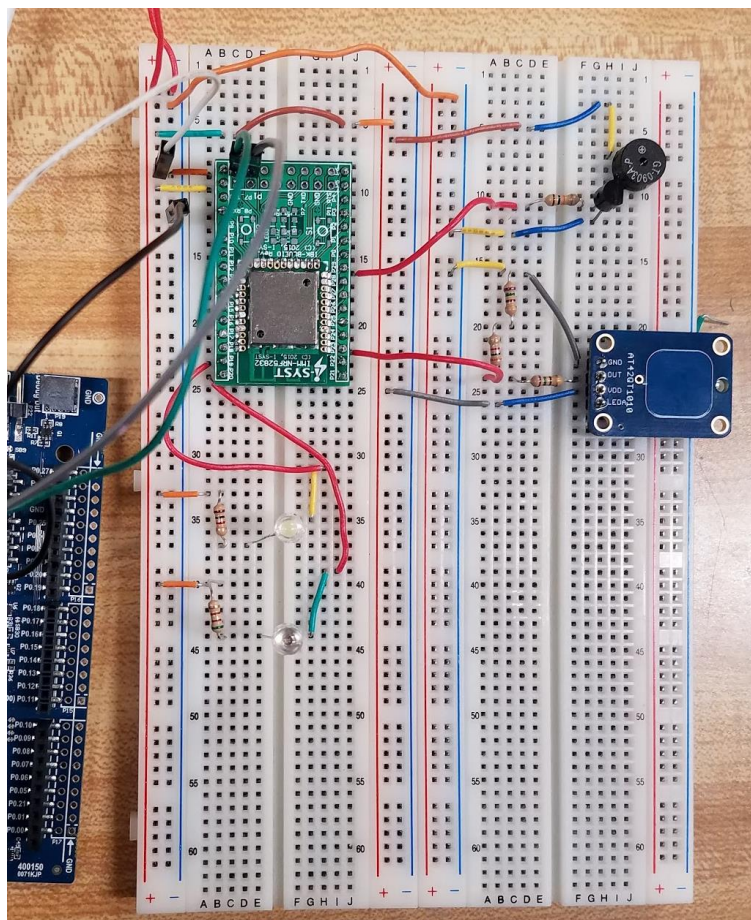


Figure 3.17: Breadboarded prototype circuit with IMM-NRF52832 micro-module.

The specific design for all the external components such as the LED, the audio buzzer and the ATMEL sensor IC circuitry were as described in the previous section. However, the use of the IMM-

NRF52832 module significantly reduced the complexity of the PCB layout design as seen in the schematic included under Appendix 7.3 and the PCB layout shown in Figure 3.19.

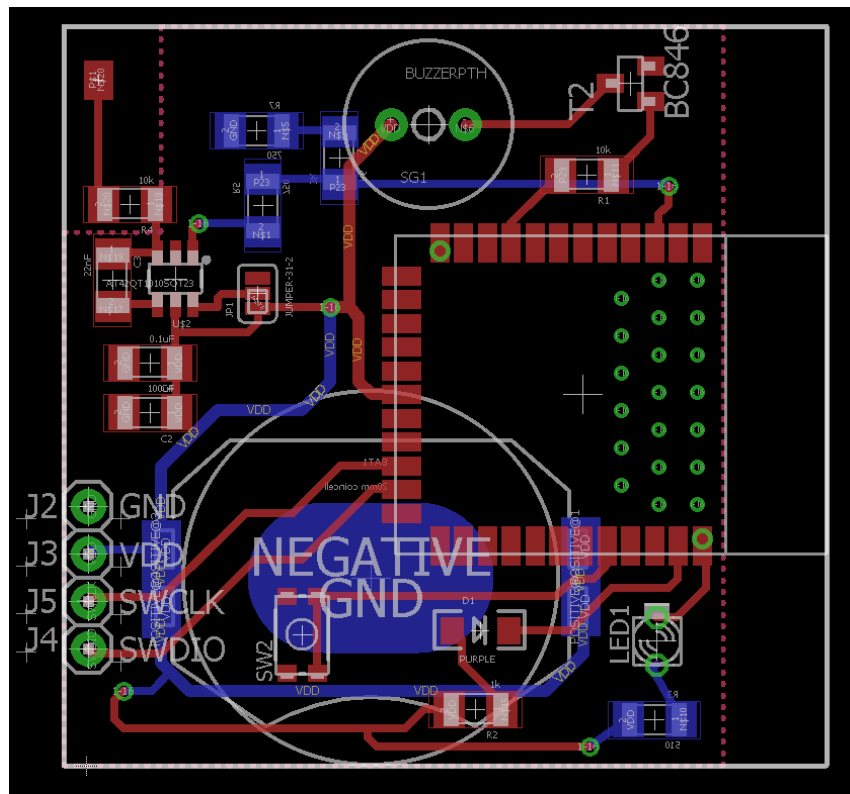


Figure 3.18: PCB layout for final revision of smart medication cap prototype using IMM-NRF52832 module.

This 40.85 mm by 39.78 mm revision of the PCB prototype is less crowded than the one shown in Figure 3.16 and involved tracing the external components shown in Figure 3.18 to their respective GPIO pins on the nRF52 module. Again, it is important to note that there is only one layer on the right side of the board where the nRF52 module antenna rests. There is also no ground plane or other traces in this antenna keep out area. In addition, the SWCLK and SWDIO pins were traced to two through-hole connectors for SWD programming.

In this prototype revision, further care was also taken when placing and tracing the components of the sensor IC's supporting circuit as detailed in the data sheet [34]. For example, the sensor IC was placed to minimize the SNSK (see Figure 3.17) trace length to reduce low frequency pickup. In addition, the  $C_S$  and  $R_S$  components were placed as close as possible to the chip's body to reduce trace length between



SNSK and  $R_S$  as much as possible. Keeping this trace short reduces its antenna like ability to pick up high frequency signals and feed them to the sensor IC. To reduce loading as well, no ground plane was included near the  $R_S$  resistor and electrode pad. Finally, the electro trace and electro pad were kept as far away from other signal and power traces. Switching signals adjacent to this trace can induce significant noise onto the sensing signal, causing the IC to misbehave.

### 3.6 Testing BLE Applications with *nRF Connect* Mobile Application

When developing firmware for nRF5x family SoCs, Nordic Semiconductor offers a valuable tool for testing and troubleshooting known as the *nRF Connect*. This tool is in the form of a smartphone application which behaves as central device and is programmed to scan and connect to BLE peripherals. Figure 3.13 shows a screenshot of the application scanner which has discovered multiple BLE peripherals including the smart medication cap. The scanner is set to list all discovered devices based on their *device name* included under GAP parameters. A shortened version of the smart medication cap device name, MQP\_SMART, is shown on the list.

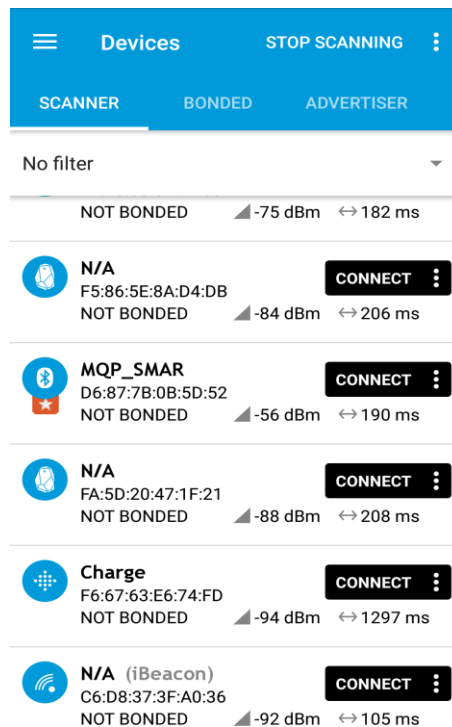
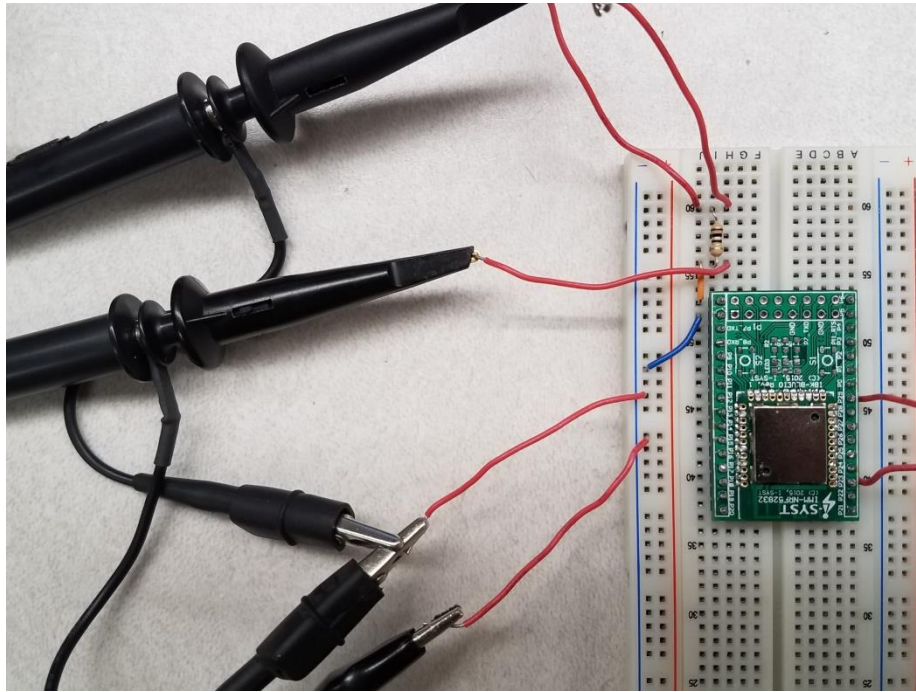


Figure 3.19: *nRF Connect* BLE device scanner listing discovered devices, including the MQP\_SMART IoT device.

The *nRF Connect* application can also discover services and allow the reading and writing of discovered characteristics. It can display real time graphs of a device's received signal strength indicator (RSSI) and allow the enabling or disabling of *BLE* notifications. But perhaps the most important functionality of the *nRF Connect* application is parsing advertisement packet data. Parsed advertisement data displays the connection modes for a scanned device, a complete list of 128-bit/16-bit UUIDs, device name, and the exact advertising interval in ms for each consecutive packet. In addition, the *nRF Connect* application logs events and method calls in enough detail to facilitate debugging during development. This feature was particularly useful for determining when and why certain connection errors occurred.

### 3.7 nRF52832 TX and RX Current Measurements

For both the smart medication cap firmware, while the devices are not undergoing an advertisement or connection event, the CPU will be in low power mode as dictated by the SoftDevice power management function *sd\_app\_evt\_wai()* which is called in an infinite loop in *main()*. Therefore, the major current consumption by the nRF52 SoC will occur when the CPU is active and undergoing an advertising and connection event. To evaluate the SoC's current performance during advertisement and connection events, oscilloscope measurements were performed using a test circuit which was similar to the one in Figure 3.18. This involved placing a 10  $\Omega$  resistor in series with the 3.0 V voltage source. Then two oscilloscope probes are placed on either side of the resistor as shown in Figure 3.21.



*Figure 3.20: Ten ohm resistor and oscilloscope probe placement for current measurement.*

Placing the oscilloscope probes at either end of the resistor allows for measuring the voltage drop across it by subtracting the two voltage signals using a math setting included in most oscilloscopes. The current consumed is then given by the following equation.

$$I_{event} = \frac{V_{drop}}{R}$$

It is possible to use this equation to calculate current consumed at each different step of the connection or advertising events. The average of these currents would then equal the total current consumed. However, performing these measurements were difficult using oscilloscopes available in the department's laboratories since they are not sensitive enough to accomplish accurate current measurements due to limitations in resolution. In addition, they were not sophisticated enough to capture advertising and connection events as a smooth waveform and would contain significant discontinuity between gathered samples.

Therefore, it was best to only accurately measure the current consumed during peak *TX* and *RX* since these two sections of the advertising or connection events resulted in the largest change in voltage. The *TX* and *RX* current results were compared to estimates given by Nordic's nRF52 Online Power Profiler [41].

### 3.8 Overview of Smart Medication Cap Android Application

Setting up the android development studio was simple compared to the steps taken in Section 3.3 to set up a firmware development environment for the nRF52 SoC. This consisted of downloading the latest version of Android Studio and installing it onto a computer along with the latest SDK and building tools. Since android development is done using Java, it was also important to have the latest version of Java's Standard Edition Development Kit (JDK) installed. The path for the JDK was then added as a new system environment variable named `JAVA_HOME`. Android Studio searches for the JDK using this specific name, hence the system variable name cannot be changed.

In order to speed up the implementation of the smart cap smartphone application, the android example project *BluetoothLEGatt* was used as a starting point. This example already contained source files for implementing a *BLE* scanner that can discover devices, connect to them, and discover available services. Further functionality and user interface pertaining to the smart medication cap was built upon this example application.

Starting with the example application was also convenient since the student working on this project had no previous experience writing in Java or developing smartphone applications. After becoming familiar with the *BluetoothLEGatt* example, it was sufficient to simply navigate the Android Developers Documentation to learn how to include further functionality and user interface to the app. The available documentation is thorough and easy to understand, making it a perfect resource for beginners. A zip file of the android project is attached to the project's files in the MQP title page.

## 4 Results

### 4.0 Results for Device Firmwares using *nRF Connect* Application

#### 4.0.0 Smart Medication Cap Device Firmware

Figure 4.1 shows the scanner on the *nRF Connect* application having discovered the smart medication cap device. Once *nRF Connect* has discovered a device, it lists it by name followed by device address. The listed information also includes whether the respective device is *single-mode* or *dual-mode* along with other flags. The information also includes all custom 128-bit service UUIDs discovered for the particular device. The UUIDs listed for the smart medication cap device are for the tracker service and sensor service as described previously.

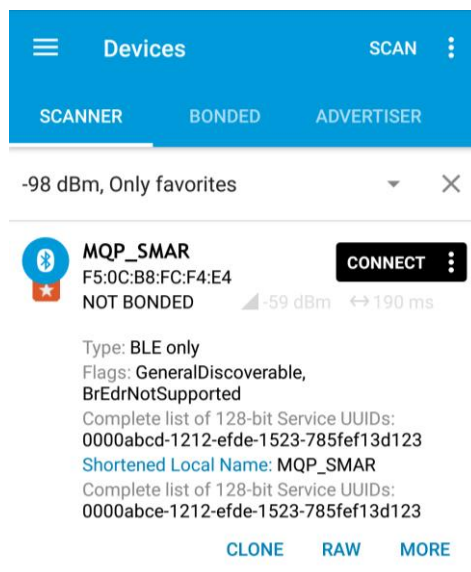


Figure 4.1: *nRF Connect* Scanner listing the discovered smart medication cap device by name and listing important properties of advertising/scan packets.

As can be seen in Figure 4.1, the smart medication cap is a *single-mode* device that only supports the *BLE* protocol. The device is also set to be a general discoverable device. The line “BrEdrNotSupported” is included to explicitly state that this device does not support classic *Bluetooth*.

Figure 4.2 shows the advertising event tracker on the *nRF Connect* application. In this particular figure, the smart medication cap is advertising in fast mode and its interval is around 191 ms. This feature

of the app allows the user to keep track of the varying advertising interval lengths for the most recent 100 events. The advertising interval can vary slightly and keeping track of this information is necessary for debugging and for evaluating the performance of the nRF52 SoC. It can be seen that all recorded interval times are close to the originally set value of 187.5 ms.

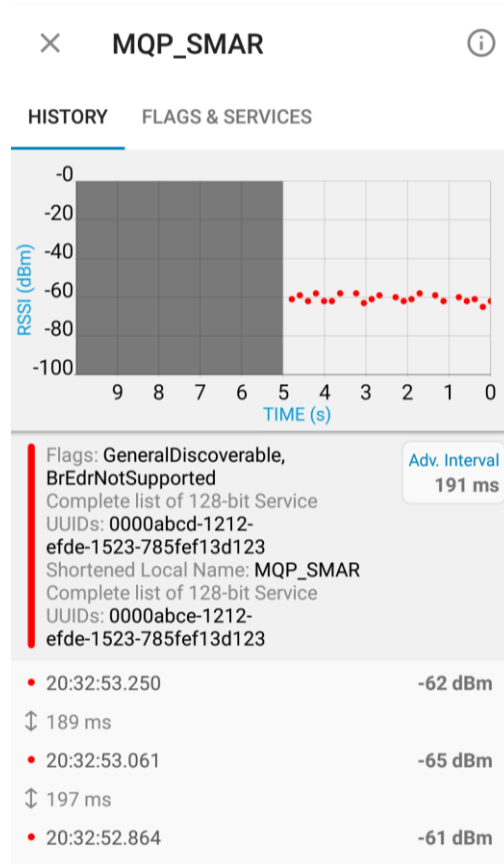


Figure 4.2: Variations in advertising interval duration for the smart medication cap in fast advertising mode along with RSSI value plot.

The plot shown in Figure 4.2 also allows the programmer to visually track the received signal strength indicator (RSSI) value for the device. This value is reported in dB and becomes less negative as the range between the central and peripheral devices becomes shorter. Figure 4.3 shows the same tracker of advertising events but for when the smart medication cap is advertising in slow mode. Again, although the advertising interval length can vary slightly, it is still close to the set value of 1 s as shown in table 3.2. The current value in this case was 994 ms.

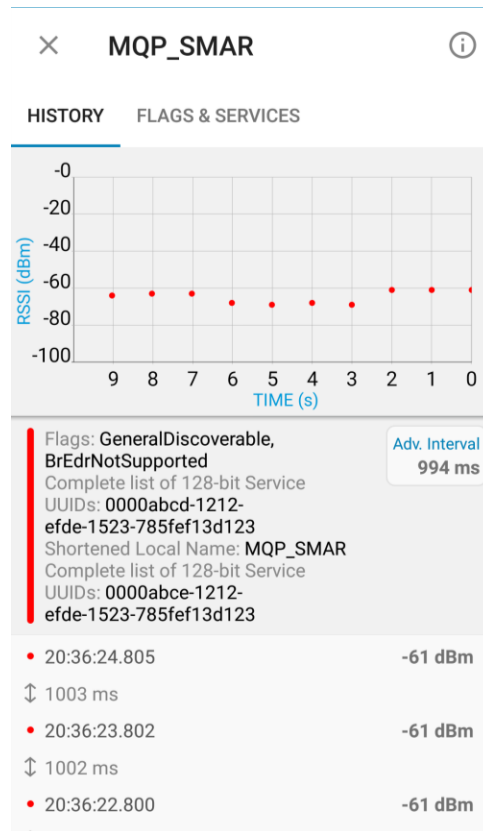


Figure 4.3: Variations in advertising interval duration for the smart medication cap in slow advertising mode along with RSSI value plot.

Figures 4.2 and 4.3 served to verify that the device was indeed switching between fast and slow advertising modes. The fact that the device could be discovered by the application scanner at any point in time also confirms that the smart medication cap is capable of advertising indefinitely while in slow mode. The device will only stop advertising when not powered. In addition, once connected, the device will only disconnect when out of range or when there is signal disruption as desired. If disconnected for any reason, the device will advertise in fast mode to connect as quickly as possible.

After connecting to the smart medication cap device, the *nRF Connect* application lists the available services and their characteristics as shown in Figures 4.4 and 4.5. It was confirmed that both the tracker and sensor services were initialized properly in the smart medication cap firmware since they were discovered and listed by *nRF Connect*. The same was true for all characteristics inside the services. If these were not initialized properly, they would not be listed as they are in Figures 4.4 and 4.5.

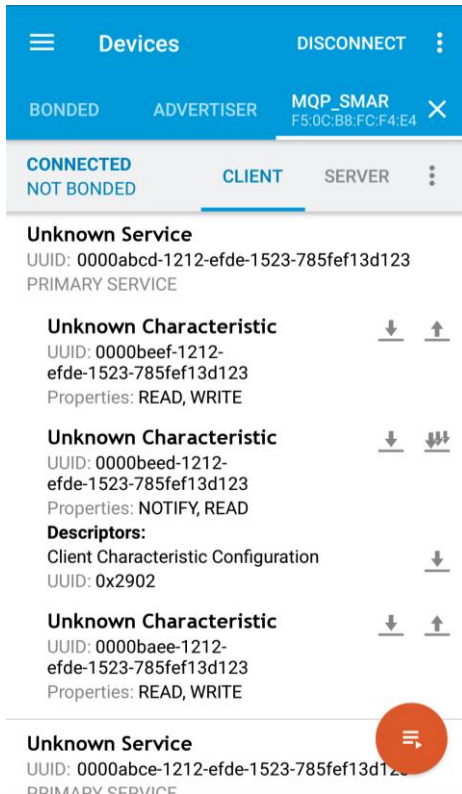


Figure 4.4: nRF Connect listing tracker service and its characteristics upon connection to the smart medication cap device.

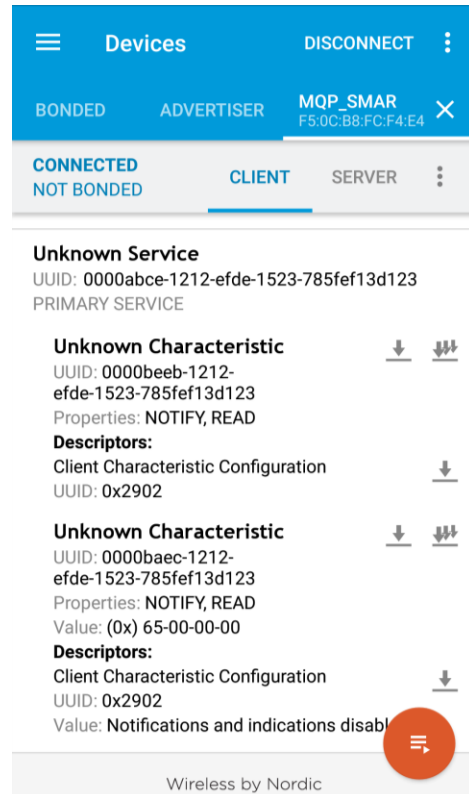


Figure 4.5: nRF Connect listing sensor service and its characteristics upon connection to the smart medication cap device.

Testing the writing, reading and notification properties of each characteristic was also successful. For example, when writing 0x01 to the *LED state* characteristic on the tracker service, the following is observed in the debug window of the nRF Connect application:

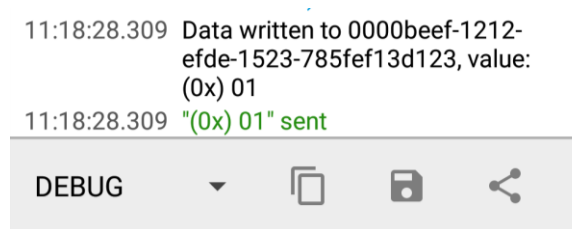


Figure 4.6: Debug window output on nRF Connect upon writing 0x01 to LED state characteristic on tracker service.



As expected, writing a 0x01 to the *LED state* characteristic triggers the LED on the smart medication cap device to flash. The LED stops flashing once 0x00 is written to the *LED state* characteristic. The same is true for the *PWM signal* characteristic on the tracker service. Writing a 0x01 to this characteristic triggers the PWM signal on the nRF52 pin which is fed to the magnetic transducer on the smart medication cap hardware for audio. Playback of the PWM signal is stopped when 0x00 is written to the characteristic.

Reading data from the touch sensor is also achieved by enabling notifications on the *nRF Connect* application. It is important to note on Figures 4.4 and 4.5 that all characteristics in the tracker and sensor services that provide the central device with a notification have the descriptor *Client Characteristic Configuration* (CCCD) set to the UUID value of 0x2902. This descriptor was set when initializing the characteristics in the *track\_service.c* and *sensor\_service.c* files as explained in Section 3.5.2. After enabling notifications and sending sensor data, the following is observed in the debug window of the *nRF Connect*:

```
12:06:39.923 "Notifications enabled" sent
12:06:39.932 gatt.setCharacteristicNotification(0000beeb-1212-efde-1523-785fef13d123, true)
12:06:39.936 Notifications enabled for 0000beeb-1212-efde-1523-785fef13d123
12:06:51.255 Notification received from 0000beeb-1212-efde-1523-785fef13d123, value: (0x) 00
12:06:51.255 "(0x) 00" received
12:06:52.823 Notification received from 0000beeb-1212-efde-1523-785fef13d123, value: (0x) 01
12:06:52.823 "(0x) 01" received
```

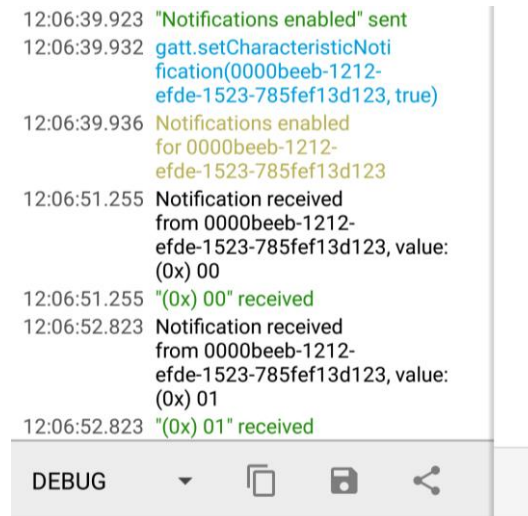
The image shows a screenshot of the nRF Connect application's debug window. The window has a light gray background and a dark gray header bar with the word "DEBUG" on the left and three icons (a dropdown arrow, a copy icon, and a share icon) on the right. The main area of the window contains a list of log entries with timestamps and messages. The messages are color-coded: green for status messages, blue for function calls, yellow for informational messages, and black for notification events. The log shows the process of enabling notifications for a specific characteristic and then receiving two notifications with values 0x00 and 0x01.

Figure 4.7: Debug window output upon enabling notifications for sensor characteristic and receiving sensor data.

The same output on the debug window is observed when enabling notifications for the *BUTTON state* and *TEMPERATURE value* characteristics. This confirms that all characteristics that send the central device a notification were initialized properly in the smart medication cap firmware and that their respective data were handled and sent appropriately. By verifying the complete functionality of the smart medication

cap using the *nRF Connect* app, it was concluded that both the firmware and the hardware prototype were functioning as expected.

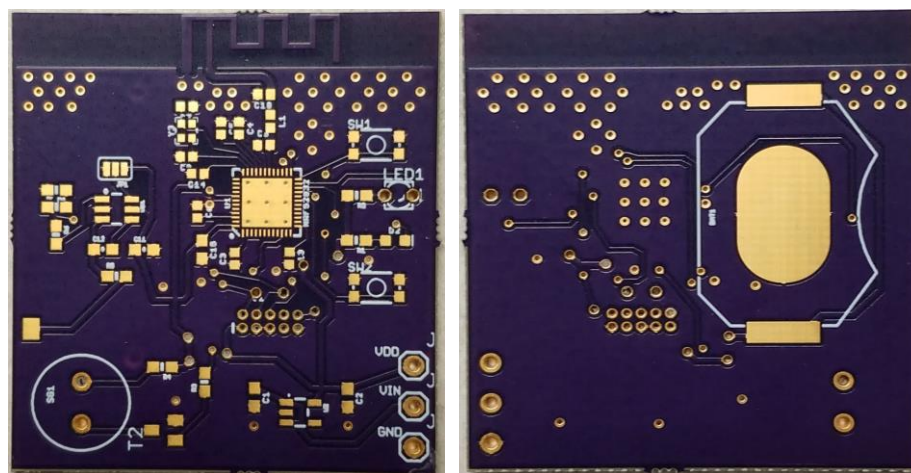
## 4.1 Prototype Hardware Results for Smart Medication Cap

In this section, the prototype hardware results for the smart medication cap are discussed. As mentioned previously, during the completion of the project there was only enough time to manufacture the two different prototype PCBs for the smart medication cap device shown in Section 3.7.

The PCB prototypes that were ordered for the smart medication cap were manufactured by OSH Park. This manufacturer was selected for its “Super Swift” service which guarantees shipping of PCB prototypes within five business days at a price of ten dollars per square inch. Due to the small size of the designed boards, ordering three boards had a cost of around thirty dollars for each prototype.

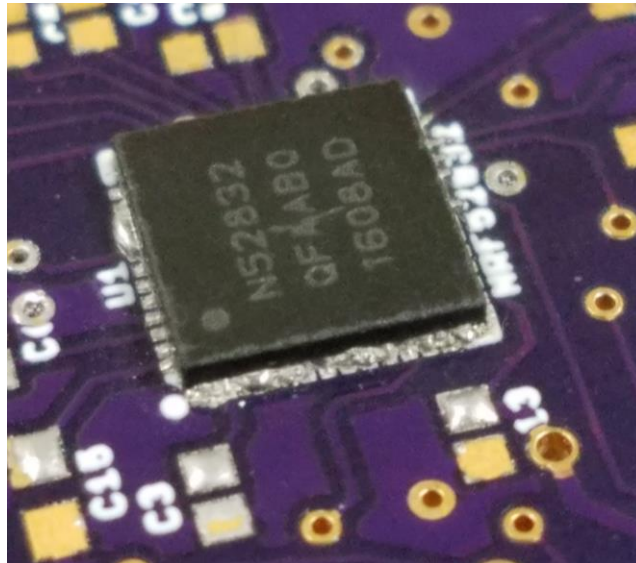
### 4.1.0 Prototype Hardware Results for Initial Design Revision

Figure 4.11 shows the front and back of the manufactured PCB for the smart medication cap prototype shown in Figure 3.16.



*Figure 4.8: Front and back of manufactured PCB for smart cap using hand soldered nRF52 SoC.*

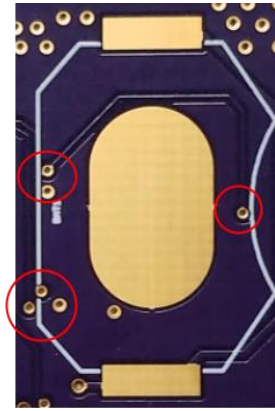
The first step in attempting to assemble this PCB prototype was to hand solder the nRF52 SoC. Figure 4.12 demonstrates the hand soldered result of the QFN package using solder paste and a stencil.



*Figure 4.9: Hand soldered nRF52 SoC on prototype PCB.*

Despite a small number of pins on the SoC that were properly soldered to their pads, there were still multiple solder bridges on each side of the chip. The most visible damage can be seen on the row containing pin number one in Figure 4.12. On this side of the chip, all pins were bridged together. In addition, since there was no reflow oven available for surface mount assembly, the chip and solder were set using a hot air soldering rework station. However, using a hot air gun is not recommended for sensitive ICs such as the nRF52 since the gun is channeling air at much higher than 200°C directly onto it. It is very likely that even if the chip was soldered properly, it would not be functional due to exposure to peak package temperature for longer than the recommended 30 seconds.

This PCB prototype contained another significant defect as shown in Figure 4.13. The footprint shown in the figure is for a 20 mm coin cell battery holder. The small circled holes are known as *vias* which, connect signal traces between the top and bottom layer of the PCB board. In the assembled prototype, the negative side of the battery would be sitting on top of these vias causing multiple shorts. It would not have been a major issue if these vias were connected to the ground plane, but they are connecting other signals, including  $V_{CC}$ , between PCB layers.

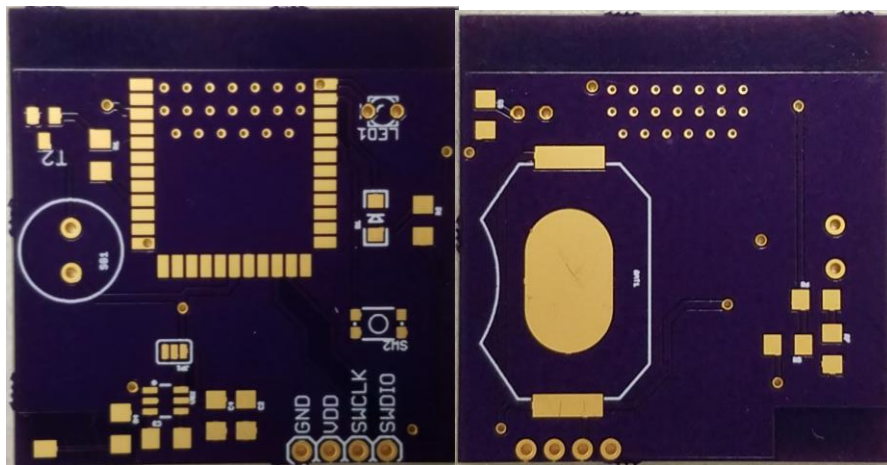


*Figure 4.10: Incorrectly placed signal and VCC vias in coin cell battery area which would lead to multiple shorts.*

Therefore, powering this smart medication cap prototype using a battery would not have been possible. The power and ground pins shown in Figure 4.11 would have provided a powering alternative.

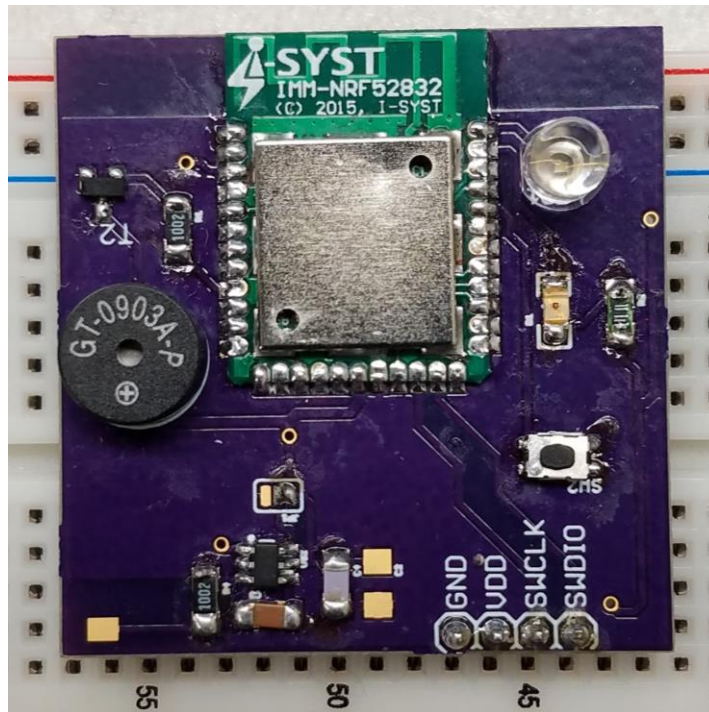
#### **4.1.1 Prototype Hardware Results for Second Design Revision**

Figure 4.14 shows the the manufactured PCB board for the second smart medication cap prototype using the IMM-NRF52832 micro-module.



*Figure 4.11: Front and back of manufactured PCB for smart cap using IMM-NRF52832 micro-module*

The assembled prototype is shown in Figure 4.15. The coin cell battery pack was not soldered initially for easy breadboarding of the board for SWD programming using the SWCLK and SWDIO pins. However, the prototype could easily be powered using the power and ground pins shown in Figures 4.14 and 4.15.



*Figure 4.12: Assembled prototype PCB for smart medication cap device.*

The capacitive touch sensor IC and its supporting components can be seen on the lower left corner of the prototype board in Figure 4.15. A wire attached to a conductive copper surface on a medicine bottle cap was soldered to the small pad on the bottom left corner of the prototype. The finished assembly along with the medicine bottle is shown in Figure 4.16. Although not clearly visible in the figure, a 20 mm coin cell battery pack is soldered to the bottom of the prototype PCB.



*Figure 4.13: Finished prototype with PCB hardware and medication bottle.*

#### **4.1.2 Notes on Hardware Design**

After having ordered the prototype PCB shown in Figures 4.14 and 4.15, a design error was observed in the breadboarded test circuit shown in Figure 3.18 concerning the AT42QT1010 touch sensor IC breakout module. Occasionally, when the PWM signal was turned on, the sensor module would behave erratically and record false input. This of course was undesirable.

After closely reading through the AT42QT1010 datasheet once more it was found that if the IC's power supply is shared with another electronic system, then care should be taken so that the supply is free of digital spikes since the IC can be negatively affected by rapid input voltage fluctuations [34]. This was indeed the case when the PWM signal was turned on. The best hardware fix is to regulate the input voltage of the IC using a Low Dropout (LDO) regulator. However, as the prototype shown in Figure 4.14 had already been ordered, the addition of the LDO regulator was not possible. It was best to assemble the prototype to look out for the same error and, if it occurred, a new design would be included using an LDO regulator. An additional software fix to bypass the issue was also added to the firmware. This was to stop reading input from the sensor when the PWM signal is currently on.

Fortunately, when the assembled prototype in Figure 4.15 was tested, the same issue was not observed. It was then concluded that the misbehavior of the touch sensor IC module in the breadboard circuit could be related to other noise not in the input voltage signal as well. But the best practice is to isolate the IC input voltage from the rest of the circuit and thus the LDO regulator should be included in future designs.

## 4.2 nRF52832 TX and RX Current Measurement Results

Figure 4.17 shows an oscilloscope capture of an advertisement event when the nRF52 SoC is advertising at 1 s intervals. The green signal (CH4) corresponds to the oscilloscope probe connected between the input voltage and the 10  $\Omega$  resistor as shown in Figure 3.21. The purple signal (CH4) corresponds to the oscilloscope probe connected between the 10  $\Omega$  resistor and the rest of the test circuit. The red signal is the difference between the CH4 and CH3 signals, i.e. the voltage drop across the resistor.

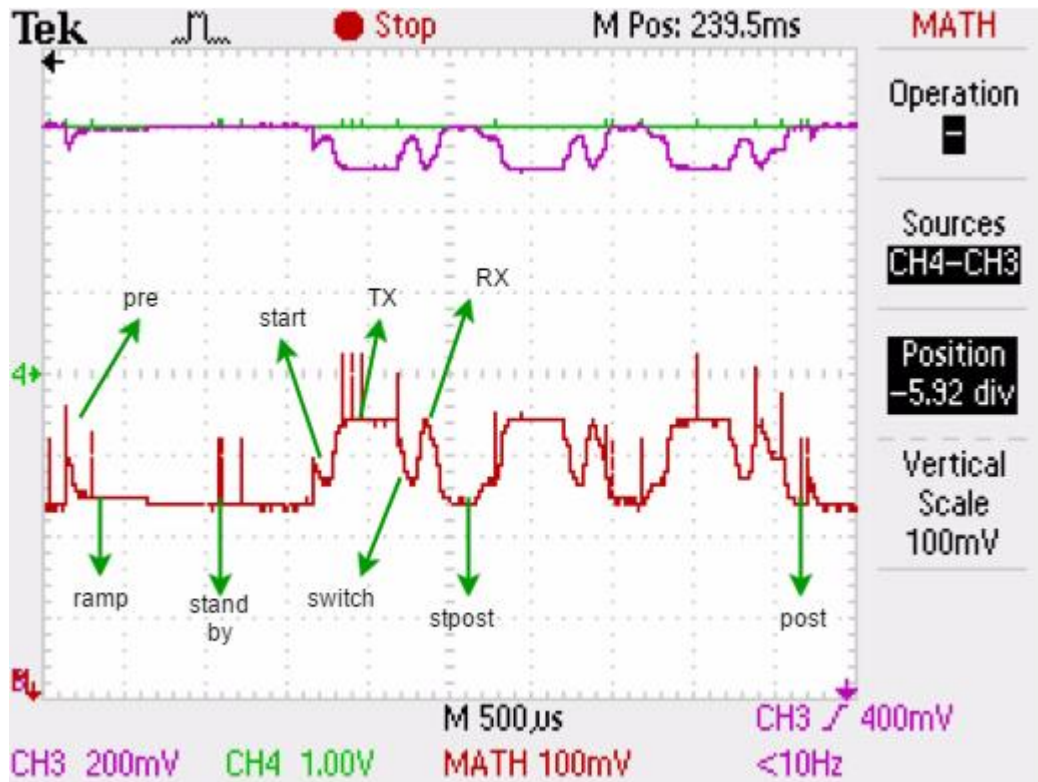


Figure 4.14: Advertising event for nRF52 SoC when advertising interval is 1 s (red signal).

The specific steps of the advertising event are also pointed out in Figure 4.17. After the *pre*, *ramp* and *stand by* events, data exchange starts followed by *TX*, *switch* and then *RX*. As seen in the figure, the complete advertising event has three consecutive data exchange intervals separated by a small *stpost* period. The max *TX* payload for the advertisement event of 31 bytes is being transmitted in this case.

Using amplitude cursors on the oscilloscope, it was possible to measure the peak *TX* and *RX* voltages as 104 mV. Therefore, using equation 3.1, the peak *TX* and *RX* currents were calculated as 10.4 mA. This measured value is around 4 mA larger than the estimate value given by Nordic's Online Power Profiler. The estimate online profiler values were 6.6 mA for *TX* and 6.7 mA for *RX*. This discrepancy could be due to the fact that measuring a change in voltage on the poor resolution wave resulted in a higher estimate than expected. But of course, the values reported on the online profiler are modeled estimates and it is possible that the value measured here is appropriate due to device variations or other aspects of the test circuit.

Figure 4.18 shows the advertising event (red signal) when the nRF52 is advertising at a 100 ms interval. The waveform is similar to the one in Figure 4.17 but is shown here using a larger time scale.



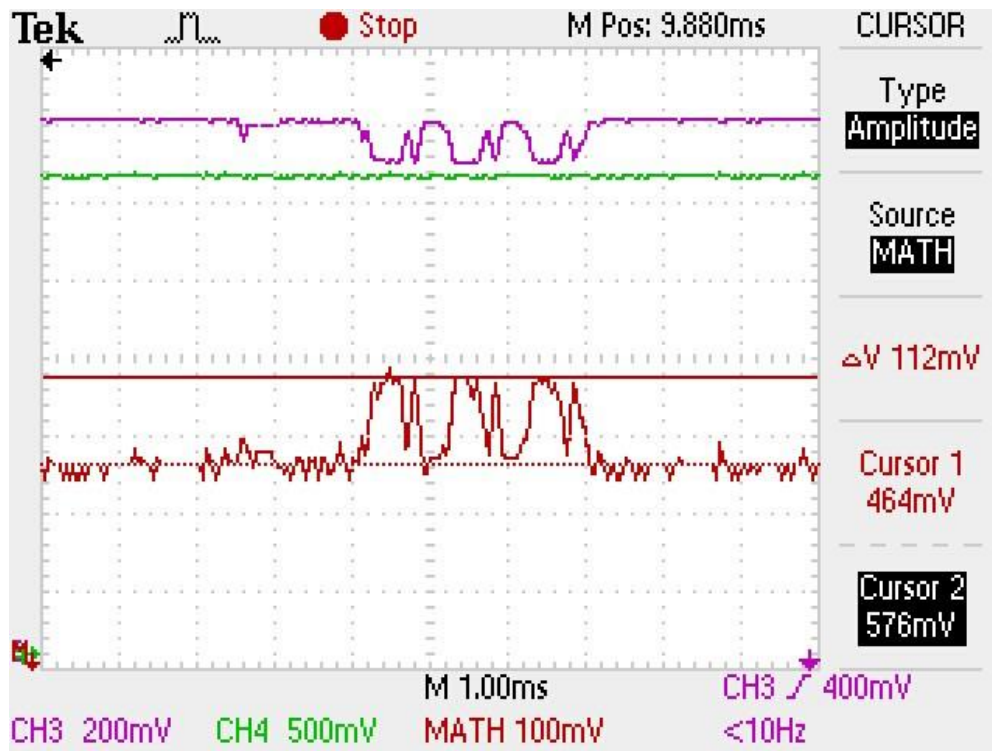


Figure 4.15: Advertising event for nRF52 SoC when advertising interval is 100 ms (red signal).

It is clear by looking at this particular signal why doing a complete average current estimate for the event using the available oscilloscopes was almost impossible. There is too much noise in the signal and the specific events pointed out in Figure 4.18 are not clearly visible. Attempts at capturing the signal at smaller time scales did not improve its overall appearance. However, it was possible to measure peak TX voltage at around 112 mV. Peak RX voltage was measured at around 108 mV. Therefore, peak current for TX and RX were 11.2 mA and 10.4 mA respectively. These peak values should not change based on advertising interval and while TX was higher than 10.4 mA, it was not by a significant amount. In fact, the average current for each part of the event will remain constant no matter the advertising interval. What makes a longer interval preferable is the fact that, since these events will take place less often, average current consumption will be lower.

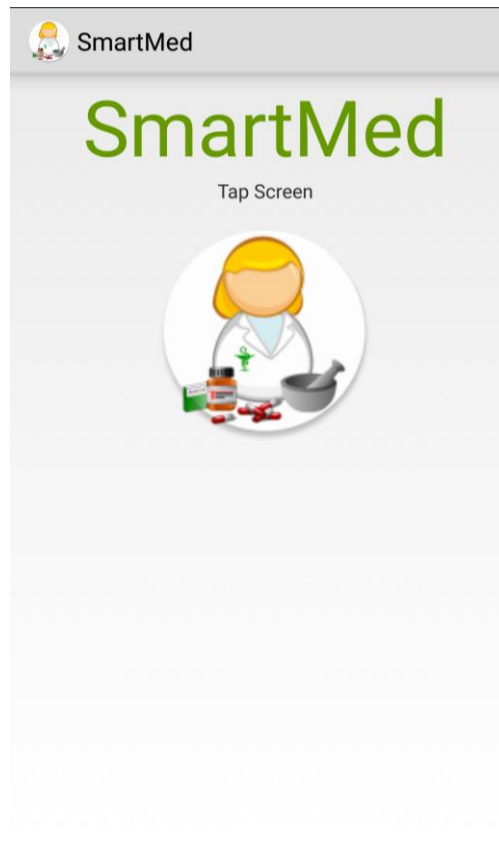
However, advertising interval is not the only factor that affects current consumption. The size of data being transmitted also plays a major role. Since 128-bit UUIDs were advertised in this case, the full 31 bytes were used. If less data, such as 20 bytes, would have been transmitted instead, the length of each



and the lower data transmission (with a maximum of 27 bytes) makes connection more power efficient than advertising.

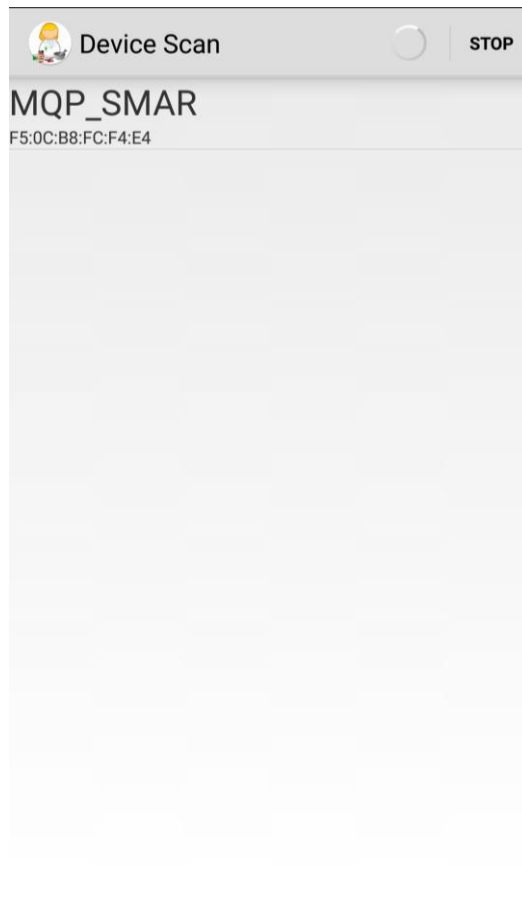
### 4.3 Smart Medication Cap Android Application Results

Figure 4.20 shows the welcome screen of the *SmartMed* android application. This screen first appears after the application is launched.



*Figure 4.17: Welcome screen for SmartMed android application, a companion app to the smart medication cap.*

At the welcome screen, the user is instructed to tap the screen to continue. The activity that follows is shown in Figure 4.21. Here, the application starts scanning for devices that only support the sensor and tracking services shown in Figure 3.2 and described in Section 3.5.2. Once a device with these services has been discovered, it will be listed by name and device address as shown in the figure.



*Figure 4.18: SmartMed application scanner which has discovered the prototype device “MQP\_SMART.”*

The STOP button on the upper right corner of the screen can be pressed to stop the application from scanning for devices. The STOP button is then replaced with a SCAN button which can be pressed to restart scanning. To establish a connection between the smartphone and the peripheral device, the listed device in Figure 4.21 can be pressed. Once pressed, the following activity automatically triggers the central device to negotiate a connection with the smart medication cap. The screen shown in Figure 4.22 is also shown.

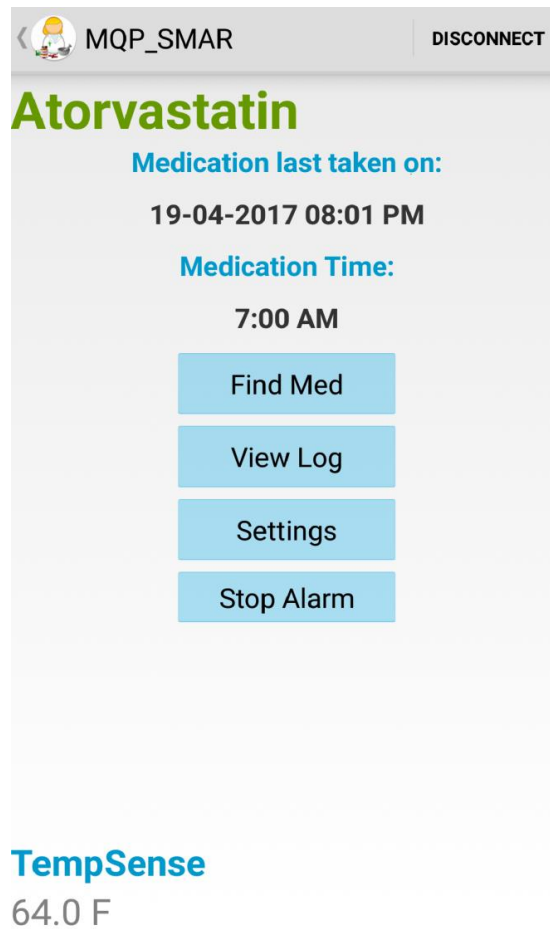
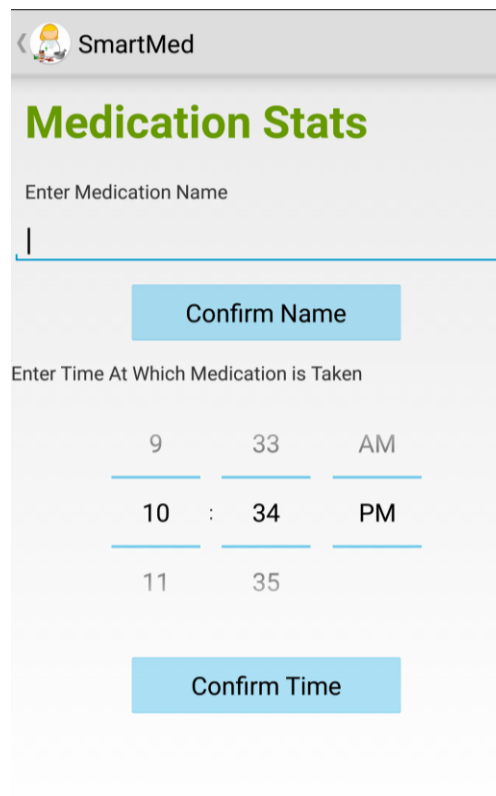


Figure 4.19: Control panel for the SmartMed android application.

The screen shown in Figure 4.22 serves as the main control panel for the *SmartMed* application. The page is titled with the medication name which is user determined. This is followed by the date and time in which the medication was last taken. The time at which the medication is supposed to be taken, which is user determined, is also listed. Three user buttons then allow for adjusting settings, locating medication, or viewing a medication consumption log. Under “TempSense” in the lower left corner of the screen is the on-chip temperature sensor reading in degrees Fahrenheit.

When the settings button is pressed, the screen in Figure 4.23 is shown. Here the user can set the name of the medication and at what time they will take it. These settings are shared preferences in the application and will remain saved even if the application is killed. The user can then press the back button on the upper left corner to return to the control panel. An alarm will go off at the time determined by the

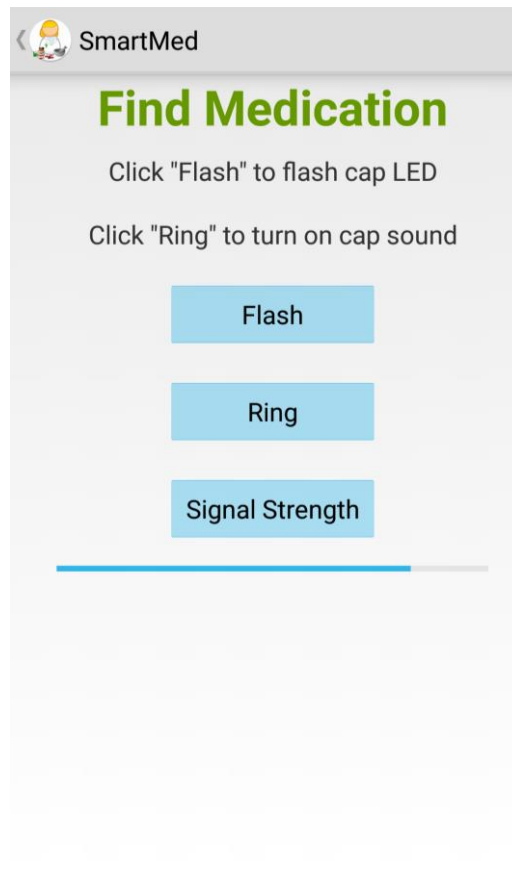
user and a message appears on the control panel instructing the user to take their medication. By clicking the “Stop Alarm” button in Figure 4.22, the user can dismiss the alarm tone.



The image shows a mobile application interface for 'SmartMed'. At the top, there is a back arrow, a small icon of a person with a pill, and the text 'SmartMed'. Below this is a green heading 'Medication Stats'. Underneath the heading is a text input field labeled 'Enter Medication Name' with a vertical cursor. A blue button labeled 'Confirm Name' is positioned below the input field. Below the button is another text input field labeled 'Enter Time At Which Medication is Taken'. This field contains a digital clock display with three columns: the first column shows '9', the second shows '33', and the third shows 'AM'. Below this row are two horizontal lines. The second row shows '10' in the first column, a colon ':' in the second, '34' in the third, and 'PM' in the fourth. Below this row is another horizontal line. The third row shows '11' in the first column and '35' in the second. A blue button labeled 'Confirm Time' is located at the bottom of the form.

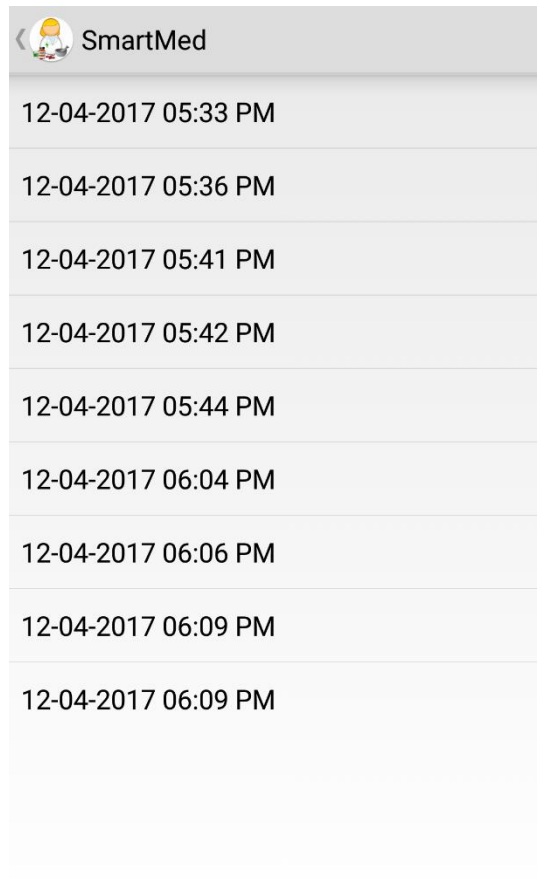
*Figure 4.20: Settings screen where the user can input medication stats such as name and time at which medication is taken.*

When the “Find Med” button is pressed, the screen in Figure 4.24 is shown. Here the user can flash the LED on the smart medication cap by pressing “Flash.” The user can also toggle the audio signal on the smart cap by pressing “Ring.” Finally, the user can get an estimate of the device’s signal strength by pressing “Signal Strength.” This feature of the app takes the device’s RSSI value and maps it to a progress bar percentage. The percentage of progress bar filled depends on how close the smartphone is to the smart medication cap. In Figure 4.24, the devices are close to each other and the progress bar is mostly filled with blue.



*Figure 4.21: Find medication screen in which the user can flash or ring the device and get approximate distance through signal strength.*

When the user takes the medication, the touch sensor on the smart cap will read input and send a *BLE* notification to the smartphone. Once the smartphone application receives notification data from the touch sensor UUID, it will record a timestamp string that is printed onto the control panel screen as shown in Figure 4.22. This timestamp includes the date in day-month-year and the time in which the sensor was touched. This string is a shared preference in the application and will remain saved even if the application is killed. The timestamp string is also saved into a simple SQLite database in order to maintain a log of timestamps. This log can be viewed when the user presses “View Log” in Figure 4.22. The log screen is shown in Figure 4.25.



*Figure 4.22: Touch sensor timestamp log in SmartMed application.*

The log is a simple list of all the timestamp strings that have been saved into the SQLite database. As the list keeps increasing in size, the user will be able to scroll down to view it completely. By evaluating this log, a patient is able to see at which specific time the medication was taken each day. The user can also see whether they have skipped days or if medication was taken twice in one day. If the user cannot remember if they have taken their medication in a particular day, they can check the date on the control panel or they can check the log.

Other miscellaneous features of the *SmartMed* application involve the small user button on the smart medication cap. If this button is pressed, a notification is sent to the smartphone and once the notification is received, the default ringtone on the phone will play. This is intended in case the user has the medication cap in hand and they cannot locate their phone.



## 5 Conclusion

The goal of this Major Qualifying Project was to design and implement an IoT device related to the field of smart health. The device requirements established at the beginning of the project are listed in Section 1.0. The device is small and for mobile applications, the device has a sensor that collects data, and the device wirelessly connects to a smartphone using *BLE*.

The implemented device is relevant in the field of smart health since it addresses the issue of non-adherence with medication regimens. As mentioned in Section 2.2, medication adherence is key in achieving optimal treatment results. However, around 20% to 50% of patients do not take their medication as indicated by trained professionals. In the long term, this can lead to higher hospitalization rates and treatment costs, especially for patients with chronic illness. The touch sensor integrated into the smart medication cap prototype registers input when a user opens the medicine bottle to take their medication. This triggers a notification which is received by the *SmartMed* android application which will record a detailed timestamp saved into a database. This way, the smartphone application keeps a detailed record of when medication has been taken (i.e. when the touch sensor registers input).

The developed firmware for the smart medication cap was fully functional as expected and implemented the Smart Medication Tracker and Item Tracker profiles successfully. In addition, the second hardware design revision for the smart medication cap prototype functioned properly during testing and did not exhibit the false input read problems by the sensor due to noise as observed in the breadboarded circuit.

The *SmartMed* android application also demonstrated all required functionality during testing. It can scan for and connect to devices that support the Smart Medication Tracker Profile and will remain connected to the peripheral device indefinitely as required. In addition, it allows the user to manipulate the data for the characteristics in the tracker service in order to locate the device using light, sound and signal strength. Most importantly, the app keeps a clear log of the date and times at which the touch sensor on the medication cap registers input, indicating that medication has been taken.

Of the requirements established at the beginning of the project, the only one that was not met was power efficiency. The developed prototype for the smart medication cap was not efficient enough to be powered using a 3.0 V coin cell battery and had to be powered using 3.0 V worth of AA batteries during testing and demonstration. The final prototype consumed around 40 mA during advertisement and during connection. When attempting to power the smart medication cap using a coin cell, most functionality still worked except the AT42QT1010 sensor IC. Since the sensor is not getting enough current, it malfunctions and registers input continuously. However, the sensor IC behaves as expected when supplied with enough current.

Even when using AA batteries, the device will not stay powered for very long. For example, when using Duracell Quantum batteries with a current capacity of around 3000 mAh, the device will only stay powered for around 75 hours. This is only around three days of battery life.

Otherwise the project results were successful and throughout the process, valuable experience was gained in the fields of embedded programming, wireless protocols, hardware prototyping and android development.

## 5.0 Future Recommendations

Although the touch sensor integrated to the smart medication cap is sufficient for the purpose of tracking when medication is taken, it is too sensitive to be completely functional in practice. If the user were to grab the bottle by the cap and not take medication, input would still be recorded. For future revision of the device, it would be desirable to explore alternate methods for determining that the medication bottle has been opened. A better approach would be using a pressure sensor along with capacitive touch sensing. If other revisions of the nRF5x SoCs are used, the capacitive touch sensor libraries would allow the user to adjust for electrode sensitivity using software.

In addition, the prototype hardware should be re-evaluated in order to achieve optimal current efficiency and allow for the device to be powered using a coin cell battery. This could involve further adjusting the connection interval and latency parameters listed under Table 3-1. However, current

measurements for the *BLE* SoC running the smart cap program were on the order of micro amps on average when it is in sleep mode. Therefore, the problems are most likely due to surrounding peripherals in the smart cap prototype such as the touch sensor IC.

Further modification could be done to the *SmartMed* android application as well. For example, recorded timestamps are saved to a local database using SQLite. SQLite implements self-contained, serverless databases that are saved into the host device's memory. This means that only the owner of the smartphone has access to the medication tracker log. Although this might be desirable in certain applications, it would still be useful to save the database to an external server. This way medical professionals, or even guardians, can have access to patient's logs if required.

## 6 References

- [1] F. Wortmann and K. Flutcher, “Internet of Things”, *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221-224, 2015.
- [2] Voas, “Demystifying the Internet of Things”, *Computer*, vol. 49, no. 6, pp. 80-83, 2016.
- [3] M. Joshi and N. Rao, “Study on Internet of Things”, *International Journal of Latest Trends in Engineering and Technology*, vol. 7, no. 2, pp. 475-483, 2016
- [4] A. Whitmore, A. Agarwal and L. Da Xu, “The Internet of Things - A survey of topics and trends”, *Information Systems Frontiers*, vol. 17, no. 2, pp. 261-274, 2014.
- [5] “3 Ways the Internet of Things is Improving Healthcare”, *HIMSS*, 2015. [Online]. Available: <http://www.himss.org/news/3-ways-internet-things-improving-healthcare>.
- [6] A. Nia and N. Jha, “A Comprehensive Study of Security of Internet of Things”, *IEEE Transactions on Emerging Topics in Computing*, 2016.
- [7] “ZigBee Wireless Standard - Digi International”, *Digi*, 2017. [Online]. Available: <http://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard>.
- [8] “What is Zigbee?”, *Zigbee*, 2017. [Online] Available: <https://www.zigbee.org/what-is-zigbee/>
- [9] C. Gomez, J. Oller and J. Paradells, “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology”, *Sensors*, vol. 12, pp. 11734-11753, 2012.
- [10] K. Townsend, “Introduction to Bluetooth Low Energy”, *Adafruit Learning System*, 2014. [Online]. Available: <https://learn.adafruit.com/introduction-to-bluetooth-low-energy>
- [11] “WiFi – Oxford Reference”. *Oxfordreference.com*, 2017. [Online]. Available: <http://www.oxfordreference.com/view/10.1093/acref/9780191800986.001.0001/acref-9780191800986-e-3017>.
- [12] “Industrial Wireless: Selecting a Wireless Technology”. *B & B Electronics*, 2017. [Online]. Available: <http://www.bb-elec.com/Learning-Center/All-White-Papers/Wireless-Cellular/Industrial-Wireless-Selecting-a-Wireless-Technolog.aspx>
- [13] C. F. Hughes, “Bluetooth Low Energy for Use with MEM Sensors”, Masters, Arizona State University, 2015.
- [14] “ZigBee compared with Bluetooth Low Energy”. *Green Peak*, 2017. [Online]. Available: <http://www.greenpeak.com/Company/Opinions/CeesLinksColumn19.pdf>
- [15] *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific Requirements*, IEEE Standard 802.15.1, 2005
- [16] J. Sponas, “Things you should know about Bluetooth range”, *Bolg.nordicsemi.com*, 2016. [Online]. Available: <http://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range>

- [17] "A Guide to Selecting a Bluetooth and BLE Chipset – Argenox Technologies", *Argenox.com*, 2016. [Online]. Available: <http://www.argenox.com/bluetooth-low-energy-ble-v4-0-development/library/a-guide-to-selecting-a-bluetooth-chipset/>
- [18] "nRF52832 Product Specification v1.3", *infocenter.nordicsemi.com*, 2017. [Online]. Available: [http://infocenter.nordicsemi.com/pdf/nRF52832\\_PS\\_v1.3.pdf](http://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.3.pdf)
- [19] S. Kido, T. Miyasaka, T. Tanaka, T. Shimizu and T. Saga, "Fall detection in toilet rooms using thermal imaging sensors", *2009 IEEE/SICE International Symposium on System Integration (SII)*, 2009.
- [20] P. Corbishley and E. Rodriguez-Villegas, "Breathing Detection: Towards a Miniaturized, Wearable, Battery-Operated Monitoring System", *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 1, pp. 196 – 204, 2008.
- [21] K. Foreman, K. Stockl, L. Le, E. Fisk, S. Shah, H. Lew, B. Solow and B. Curtis, "Impact of a Text Messaging Pilot Program on Patient Medication Adherence", *Clinical Therapeutics*, vol. 34, no. 5, pp. 1084-1091, 2012.
- [22] S. Vashist, "Continuous Glucose Monitoring Systems: A Review", *Diagnostics*, vol. 3, no. 4, pp. 385-412, 2013.
- [23] "GlucoWise™ : Meet the new non-invasive glucose monitor that helps you take control of your life", *Glucowise.com*, 2017. [Online]. Available: <http://www.glucowise.com/>
- [24] "Bluetooth BR/EDR", *Bluetooth.com*, 2017. [Online]. Available: <https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/br-edr>
- [25] "Specification of the Bluetooth System: Covered Core Package version 4.2", *Bluetooth*, vol. 0, 2014.
- [26] M. Galeev, "Bluetooth 4.0: An introduction to Bluetooth Low Energy – Part 1", *EE Times*, 2011. [Online]. Available: [http://www.eetimes.com/document.asp?doc\\_id=1278927](http://www.eetimes.com/document.asp?doc_id=1278927)
- [27] M. Bors-Lind, "Tutorials - Nordic Developer Zone", *Devzone.nordicsemi.com*, 2015. [Online]. Available: <https://devzone.nordicsemi.com/tutorials/5/>
- [28] M. Bors-Lind, "BLE Services, a beginner's tutorial", *Devzone.nordicsemi.com*, 2015. [Online]. Available: <https://devzone.nordicsemi.com/tutorials/8/>
- [29] M. Bors-Lind, "BLE Characteristics, a beginner's tutorial", *Devzone.nordicsemi.com*, 2016. [Online]. Available: <https://devzone.nordicsemi.com/tutorials/17/>
- [30] "Generic Attribute (GATT) and the Generic Attribute Profile", *Bluetooth.com*, 2017. [Online]. Available: <https://www.bluetooth.com/specifications/generic-attributes-overview>
- [31] "Nordic Semiconductor Infocenter", *Infocenter.nordicsemi.com*, 2017. [Online]. Available: <https://infocenter.nordicsemi.com/index.jsp>
- [32] "nRF52832 Breakout Board Hookup Guide - learn.sparkfun.com", *Learn.sparkfun.com*, 2017. [Online]. Available: [https://learn.sparkfun.com/tutorials/nrf52832-breakout-board-hookup-guide?\\_ga=1.185151593.1315312906.1488226688](https://learn.sparkfun.com/tutorials/nrf52832-breakout-board-hookup-guide?_ga=1.185151593.1315312906.1488226688)
- [33] "Hardware Reference IMM-NRF52832 Micro-module", *i-syst.com*, 2016. [Online]. Available: [http://i-syst.com/pdf/IMM-NRF52832\\_RefMan.pdf](http://i-syst.com/pdf/IMM-NRF52832_RefMan.pdf)

- [34] “One-channel Touch Sensor IC AT42QT1010”, *ATMEL*, 2017. [Online]. Available: <https://cdn-shop.adafruit.com/datasheets/AT42QT1010.pdf>
- [35] E. Thorsrud, “Capacitive Touch on the nRF52”, *Devzone.nordicsemi.com*, 2016. [Online]. Available: <https://devzone.nordicsemi.com/tutorials/30/>
- [36] V. Berg, “Development with GCC and Eclipse”, *Devzone.nordicsemi.com*, 2015. [Online]. Available: <https://devzone.nordicsemi.com/tutorials/7/>
- [37] A. Strand, “Programming and debugging custom nRF5x devices”, *Devzone.nordicsemi.com*, 2000. [Online]. Available: <https://devzone.nordicsemi.com/blogs/803/programming-and-debugging-custom-nrf5x-devices/>
- [38] “What low-frequency clock source can I use?”, *Devzone.nordicsemi.com*, 2013. [Online]. Available: <https://devzone.nordicsemi.com/question/953/what-low-frequency-clock-sources-can-i-use/>
- [39] “Technical Considerations”, *Bluetooth*, 2017. [Online]. Available: <https://www.bluetooth.com/specifications/bluetooth-core-specification/technical-considerations>
- [40] “What does CCCD mean?”, *Devzone.nordicsemi.com*, 2013. [Online]. Available: <https://devzone.nordicsemi.com/question/1765/what-does-cccd-mean/>
- [41] “nRF52 Online Power Profiler”, *Devzone.nordicsemi.com*, 2016. [Online]. Available: <https://devzone.nordicsemi.com/blogs/903/>

## 7 Appendix

### 7.0 nRF52832 SoC Specifications

| Key features  | Applications   |
|---|--|
| <ul style="list-style-type: none"> <li>2.4 GHz transceiver           <ul style="list-style-type: none"> <li>-96 dBm sensitivity in <i>Bluetooth</i><sup>®</sup> low energy mode</li> <li>2 Mbps <i>Bluetooth</i><sup>®</sup> low energy mode</li> <li>1 Mbps, 2 Mbps supported data rates</li> <li>TX power -20 to +4 dBm in 4 dB steps</li> <li>Single-pin antenna interface</li> <li>5.3 mA peak current in TX (0 dBm)</li> <li>5.4 mA peak current in RX</li> <li>RSSI (1 dB resolution)</li> </ul> </li> <li>ARM<sup>®</sup> Cortex<sup>®</sup>-M4 32-bit processor with FPU, 64 MHz           <ul style="list-style-type: none"> <li>215 EEMBC CoreMark<sup>®</sup> score running from flash memory</li> <li>58 <math>\mu</math>A/MHz running from flash memory</li> <li>51.6 <math>\mu</math>A/MHz running from RAM</li> <li>Data watchpoint and trace (DWT), embedded trace macrocell (ETM), and instrumentation trace macrocell (ITM)</li> <li>Serial wire debug (SWD)</li> <li>Trace port</li> </ul> </li> <li>Flexible power management           <ul style="list-style-type: none"> <li>Supply voltage range 1.7 V–3.6 V</li> <li>Fully automatic LDO and DC/DC regulator system</li> <li>Fast wake-up using 64 MHz internal oscillator</li> <li>0.3 <math>\mu</math>A at 3 V in OFF mode</li> <li>0.7 <math>\mu</math>A at 3 V in OFF mode with full 64 kB RAM retention</li> <li>1.9 <math>\mu</math>A at 3 V in ON mode, no RAM retention, wake on RTC</li> </ul> </li> <li>Memory           <ul style="list-style-type: none"> <li>512 kB flash/64 kB RAM</li> <li>256 kB flash/32 kB RAM</li> </ul> </li> <li>Nordic SoftDevice ready</li> <li>Support for concurrent multi-protocol</li> <li>Type 2 near field communication (NFC-A) tag with wakeup-on-field and touch-to-pair capabilities</li> <li>12-bit, 200 ksp/s ADC - 8 configurable channels with programmable gain</li> <li>64 level comparator</li> <li>15 level low power comparator with wakeup from System OFF mode</li> <li>Temperature sensor</li> <li>32 general purpose I/O pins</li> <li>3x 4-channel pulse width modulator (PWM) units with EasyDMA</li> <li>Digital microphone interface (PDM)</li> <li>5x 32-bit timers with counter mode</li> <li>Up to 3x SPI master/slave with EasyDMA</li> <li>Up to 2x I2C compatible 2-Wire master/slave</li> <li>I2S with EasyDMA</li> <li>UART (CTS/RTS) with EasyDMA</li> <li>Programmable peripheral interconnect (PPI)</li> <li>Quadrature decoder (QDEC)</li> <li>AES HW encryption with EasyDMA</li> <li>Autonomous peripheral operation without CPU intervention using PPI and EasyDMA</li> <li>3x real-time counter (RTC)</li> <li>External system           <ul style="list-style-type: none"> <li>Single crystal operation</li> <li>On-chip balun (single-ended RF)</li> <li>Few external components</li> </ul> </li> <li>Package variants           <ul style="list-style-type: none"> <li>QFN48 package, 6 × 6 mm</li> <li>WLCSP package, 3.0 × 3.2 mm</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>Internet of Things (IoT)           <ul style="list-style-type: none"> <li>Home automation</li> <li>Sensor networks</li> <li>Building automation</li> <li>Industrial</li> <li>Retail</li> </ul> </li> <li>Personal area networks           <ul style="list-style-type: none"> <li>Health/fitness sensor and monitor devices</li> <li>Medical devices</li> <li>Key fobs and wrist watches</li> </ul> </li> <li>Interactive entertainment devices           <ul style="list-style-type: none"> <li>Remote controls</li> <li>Gaming controllers</li> </ul> </li> <li>Beacons           <ul style="list-style-type: none"> <li>A4WP wireless chargers and devices</li> <li>Remote control toys</li> <li>Computer peripherals and I/O devices</li> </ul> </li> <li> <ul style="list-style-type: none"> <li>Mouse</li> <li>Keyboard</li> <li>Multi-touch trackpad</li> <li>Gaming</li> </ul> </li> </ul> |

# 7.1 nRF52832 Mechanical Specifications

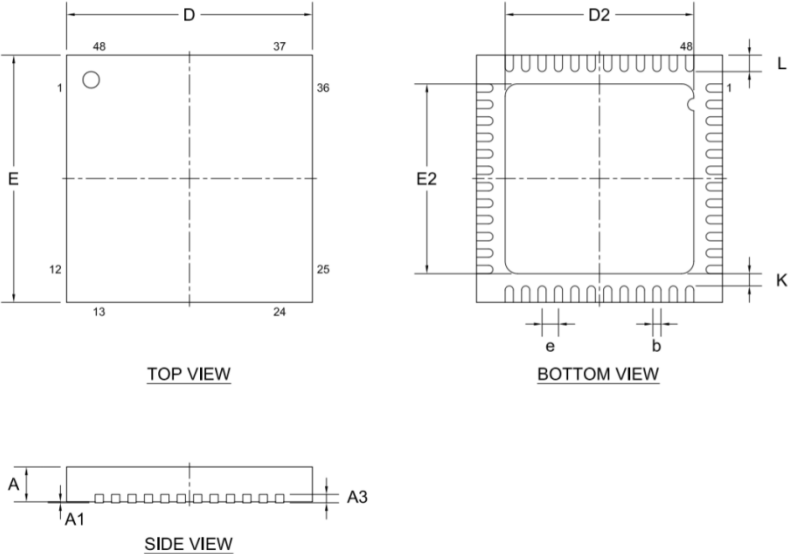


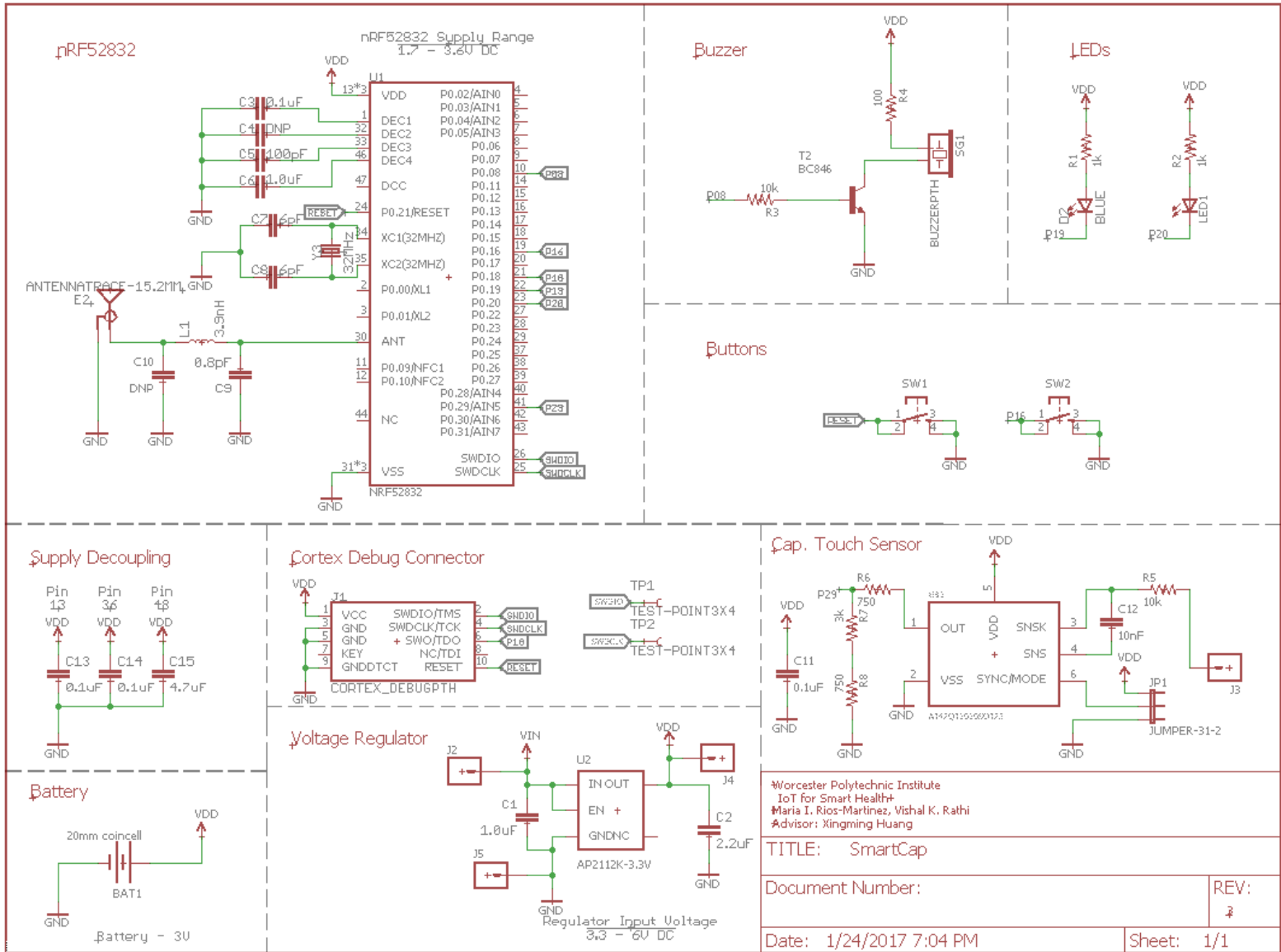
Figure 7.1: 6mm x 6mm QFN48 package and dimensions [18]:

| Package     | A    | A1   | A3  | b    | D, E | D2, E2 | e   | K    | L    |      |
|-------------|------|------|-----|------|------|--------|-----|------|------|------|
| QFN48 (6x6) | 0.80 | 0.00 |     | 0.15 |      | 4.50   |     | 0.20 | 0.35 | Min. |
|             | 0.85 | 0.02 | 0.2 | 0.20 | 6.0  | 4.60   | 0.4 |      | 0.40 | Nom. |
|             | 0.90 | 0.05 |     | 0.25 |      | 4.70   |     |      | 0.45 | Max. |

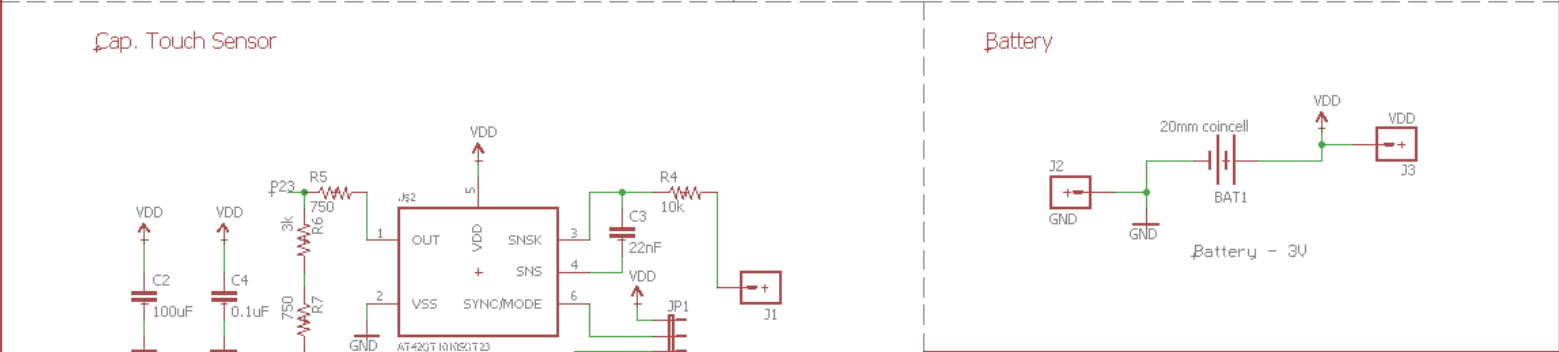
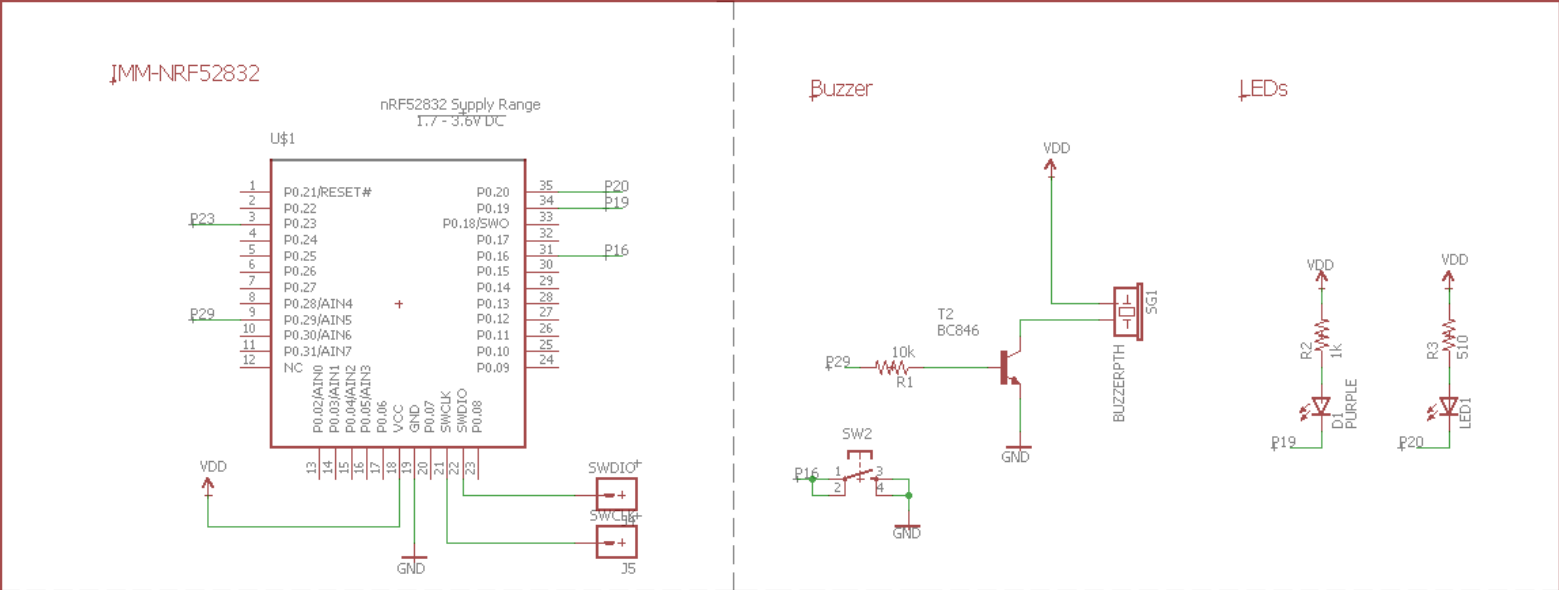
Figure 7.2: QFN48 package dimensions (mm) [18].



## 7.2 Smart Medication Cap First Hardware Design Schematic



### 7.3 Smart Medication Cap Second Hardware Design Schematic



|  |            |
|--|------------|
| Worcester Polytechnic Institute<br>IoT for Smart Health+<br>Maria I. Rios-Martinez<br>Advisor: Xinming Huang |            |
| TITLE: MQP-SmartCap-IMM-NRF52  |            |
| Document Number:   | REV:       |
| Date: 4/2/2017 2:24 PM   | Sheet: 1/1 |