Master Thesis

Implantable Bluetooth Low Energy Embedded Sensor System

Kosar Nozari Mirarkolaei

Thesis submitted for the degree of
Master in Microelectronics and Sensor Technologies
60 credits

Department of Physics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2020

Master Thesis

Implantable Bluetooth Low Energy
Embedded Sensor System

Kosar Nozari Mirarkolaei

© 2020 Kosar Nozari Mirarkolaei

Master Thesis

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

High bladder pressure in patients suffering from neurogenic bladder dysfunction can have serious consequences on patients' mental and physical health. Constant monitoring of the bladder pressures allow physicians to mitigate the risk of high pressure. Two commonly used techniques in the clinical practices to monitor this pressure are catheter-based and wireless techniques. Catheter-based pressure monitoring is done by inserting a sensor into the bladder from urethra. Even though this approach has a higher accuracy compared to the wireless sensors, it poses a high risk of infection. Pressure monitoring using a wireless sensor, on the other hand, improves patients' comfort and reduces the risk of infection. Utilizing wireless communication techniques in implantable health monitoring devices has gained considerable attention in recent years.

This thesis covers the implementation of an implantable embedded sensor system using a wireless communication technique based on Bluetooth low energy. The implementation is based on the Nordic Semiconductor's nRF52840 SoC. Tests and development are conducted on Nordic's nRF52840 development kit. We additionally propose a two-layer printed circuit board. The sensory system is aimed to communicate with a smartphone application in order to monitor the pressure.

The proposed printed circuit board has a dimension of $27.43mm \times 27.58mm$. Current consumption of the implementation is $0.915mA$ in the IDLE state and $1.114mA$ in the operating mode.

Contents

Acknowledgements

This thesis is the end of my journey in obtaining the degree of Master of Science in Microelectronic and Sensor Technology at the Department of Physics, University of Oslo. This work was carried out from August 2019 to November 2020. I would like to take this opportunity to thank the people who helped me throughout this journey. First of all, I would like to thank my main supervisor Prof. Philipp Dominik Häfliger of Department of Informatics for letting me undertake this project and for his valuable feedback and support during the work on this thesis. I would also like to thank my co-supervisor Prof. Ketil Røed of the Department of Physics and Girish Aramanekoppa Subbarao for their helps along the way.

Last but not least, I would like to thank my dear husband, Hani, for always being there for me. Without his sunny optimism, I would be a much grumpier person. And to my family and friends thank you for your word of encouragements and support.

Part I

Introduction

Chapter 1

Background

1.1 Motivation

Every year, around the world, between 250 000 and 500 000 people suffer serious damage to their spinal cords [1]. The most common urologic complications following spinal cord injury (SCI) are urinary tract infection (UTI), upper and lower urinary tract deterioration, and bladder or renal stones. Nerve problems caused by diseases like multiple sclerosis (MS), Parkinson's disease, or diabetes can also lead to bladder dysfunction [2]. In most cases, having high bladder pressure could easily affect the kidneys and lead to damage that may be life-threatening.

In addition to physical complications caused by neurogenic bladder dysfunction, there can also be negative psycho-social effects present in each individual. These can include a decrease in the individual's quality of life, and feelings of embarrassment and depression that can further lead to social isolation or devastating in the case of someone with a progressive neurological condition [3].

Frequent monitoring of bladder pressure can substantially mitigate the risk caused by neurogenic bladder dysfunction. One of the technologies used for bladder pressure monitoring is catheter-based monitoring techniques. Even though these techniques are still in use in clinical practice, they can induce complications due to the usage of wires or catheters. Bacterial colonization is one of the consequences of these techniques. Bacteria can invade the bladder by migrating along the inside and the outside of the catheter [4]. With short-term catheterization, 95% of catheterized patients suffer bacterial invasion after 1 month [5]. Urinary tract infection necessitates the use of antibiotics, which are all too frequently untested against the specific bacteria and consequently often proven to be ineffective until the right one is found by a process of trial and error. This adds to the cost of clinical management, as well as being a burden for patients and carers. The use of antibiotics to control catheter induced infections contributes significantly to the development of resistant strains, about which the World Health Organization (WHO) has expressed serious concerns [6].

On the other hand, implantable devices based on wireless communication technologies are a promising enhancement to monitor bladder pressure. It increases patient comfort while preventing complications caused by wires or catheters [7].

To leverage wireless communication technologies, smartphones are good candidates. These days smartphones are an unavoidable part of our daily life. They provide different types of communication facilities such as WiFi, Bluetooth, and Near Field Communication (NFC). Using these types of communication technologies in health monitoring systems introduced the term mHealth (mobile health). These days we can see an increasing number of health related applications on smart phones. With the aid of these applications patients themselves can perform the monitoring part.

The work presented in this thesis is to implement a system to monitor bladder pressure from the sensor using Bluetooth Low Energy (BLE) technology as a means of communication. Thus, patients can easily read sensor data using their smartphones.

1.2 Previous Work

Over the past decade, there has been a growing interest among researchers in developing various architectures for implantable pressure sensors. Ali Zaher *et al.* [8] designed an implantable system that consists of dual ASIC and one FPGA with the goal of integrating everything on one single ASIC. Their implementation is relying on the NFC protocol for data and power transmission. The physical layer of the communication and the power harvester have been implemented on one ASIC, and the sensor front-end and ADC on another, while the digital circuits realizing the higher level NFC protocol have been implemented on an FPGA. The system is not capable of powering itself in absence of a power harvester, thus, the sensor will only run in the presence of the reader which delivers the power needed. A rectified output, from the signal captured across the antenna, is then forwarded to three different regulators to generate the required voltages for the system.

Tantin *et al.* [7] introduced an embedded system implantable bladder pressure sensor based on Medical Implant Communication System (MICS). MICS is an FDA-approved RF technology which is an important advantage of this implementation. This implant is utilizing a MICS transceiver, microcontroller, two amplifiers, and two sensor probes. Their implementation relies on a battery as a power supply. Their study showed that using a small battery of 600mAh an autonomy of 39 days could be achieved.

Wang *et al.* [9] designed a system for long-term bladder urine pressure measurement. Their system consists of three major components: a pressure sensor, a control ASIC, and a Radio Frequency (RF) module. The battery is the main power source for this implementation. To save battery power for a long-term observation, a control sequence was employed to turn the system on and off in specific time intervals.

Sensing systems powered by batteries cannot work for a long time because of the tradeoff between the size limitation for implantation and the battery size. Majerus *et al.* [10] improved the power supply mode. In their research, a micro-battery which can be recharged with an RF signal is used to power the implant part.

In a recent study, Zhong *et al.* [11] developed a batteryless bladder pressure monitor system that monitors bladder storage in real-time and transmits the feedback signal to the external receiver through BLE. Their design pressure measurement circuit consists of a liquid pressure sensor, an instrument amplifier, and a microcontroller. The pressure signal is amplified by an instrumentation amplifier to meet the requirement of bladder pressure measurement. This analog value is then converted to a digital value by the analog to digital converter integrated inside the microcontroller. They use a four-coil wireless energy transmission method, which supports a power transmission range of up to 7 cm. The power transmission is based on *WiTricity* method [12]. This method can achieve further transmission distance and is considered safer due to the weak interaction of magnetic fields on biological organisms.

1.3 Embedded system based design

As was mentioned above, the architectural design of an implantable bladder sensor can be categorized into two groups: ASIC and embedded system. ASIC design may give a better performance than an embedded system based design. It offers better power efficiency for high-performance applications. The flexibility of ASICs allows for the use of multiple voltages and thresholds to match the performance of critical regions to their timing constraints and hence minimize the power consumption. However, ASICs have a long design cycle that can vary from month to year and it requires higher financial resources[13].

On the other hand, with recent advances in embedded systems, factors like low power and high speed have improved significantly. The implementation of an embedded system is less time and money consuming process [14]. Due to these reasons, in this thesis, we decided to pursue an embedded system based implementation.

1.4 Bluetooth Low Energy

The communication protocol which was used in this thesis is Bluetooth Low Energy (BLE). This section gives a brief history of BLE technology and elaborates on why we chose this communication protocol.

BLE is a low power wireless technology used to establish a connection between a pair of devices. BLE operates in the 2.4 GHz ISM (Industrial, Scientific, and Medical) band, and is targeted towards applications that need to consume less power to be powered by a battery for a long period [15].

BLE was introduced in the year 2010 as part of the Bluetooth specification 4.0 release. The original Bluetooth defined in the previous versions is referred to as the Bluetooth Classic. BLE is not an upgrade to the original Bluetooth, but rather it is a new technology that utilizes the Bluetooth brand but focuses on the Internet of Things (IoT) applications where small amounts of data are transferred at lower speeds. It is important to note that there is a big difference between Bluetooth Classic and BLE in terms of technical specifications, implementations, and the types of applications they are each suitable for. For instance, Bluetooth Classic can be used for streaming applications such as audio streaming and file transfers. On the other hand, BLE is used for sensor data, control of devices, and low-bandwidth applications.

Bluetooth Classic, however, is not compatible with BLE. Therefore, the BLE device cannot communicate directly with a Bluetooth Classic device. Nevertheless, some devices implement both BLE and Bluetooth Classic and allow talking to these devices independently.

In BLE there are two kinds of devices: a Central device, and a Peripheral device. A central device is usually a more capable device in terms of CPU power, memory, and battery capacity. Whereas, a peripheral device is more resource-constrained, especially when it comes to battery life. BLE is an asymmetric technology which means that most of the heavy lifting and the processing responsibility is put on the central device versus on the peripheral. This allows the peripheral device to sleep for a longer period, turn off the radio, and consume less power.

To mention some of the major benefits of BLE over the available communication technologies in the market first thing that comes to mind is its low power characteristic. As it was mentioned, BLE has low power consumption. Even when we compare BLE to other low power technologies, BLE is one of the lowest power technology available [16]. It is also free of cost to access the original specification documents. BLE module and chipset have low cost and it makes it suitable for the budget-constrained applications. The last and probably the most important advantage of BLE over other technologies is the existence of this protocol in most smartphones of the market.

The BLE range can be up to 50 meters. This however significantly decreases to just several meters in the presence of obstacles or walls. Antenna design and device orientation are also some other factors that can limit the range[17].

BLE 5.0 added two new Physical Layer (PHY) variants to the PHY specification used in Bluetooth 4 : Coded PHY (long-range) and 2MBPS (2Mbps bit rate). The former is used for long-range communication and the latter is used for faster communication. For example, in applications where long-range communication is required it is possible to increase the BLE range up to 500 meters with BLE 5 long-range mode [18].

For a relatively new standard, BLE has seen an uncommonly rapid adoption rate, and the number of devices that use BLE is much more than other wireless technologies. Compared to other wireless standards, the rapid growth of BLE is relatively easy to explain. Because its fate is so intimately tied to the phenomenal growth in smartphones, tablets, and mobile computing.

Even though BLE is not FDA approved for use in implantable devices, especially due to BLE security vulnerability [19] , there are already several health monitoring applications that utilize BLE communication. With the availability of BLE on smartphones and rapid growth in health monitoring applications, it is likely to see implantable devices using BLE protocol soon.

1.5 Tools

This section will give a brief description of the tools and frameworks that have been used throughout this implementation.

1.5.1 nRF52840 Development Kit

The Nordic Semiconductor nRF52840 DK is a versatile single-board development kit for BLE, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT, and 2.4 GHz proprietary applications on the nRF52840 SoC. It facilitates development by exploiting all features of the nRF52840 SoC. All GPIOs are available via edge connectors and headers, and 4 buttons and 4 LEDs simplify input and output to and from the SoC. The development board comes with an on-board Segger J-Link debugger allowing programming and debugging of onboard SoC. Programming files are written to SoC using a USB interface and the nRFgo Studio windows application [20]. The development kit also has dedicated current measurement pins which enables us to measure the SoC's current consumption. We will use these pins to measure the current consumption of our implementation later on.

1.5.2 Segger Embedded Studio

Segger embedded Studio is a complete all-in-one solution for managing, building, testing, and deploying embedded applications. It is easy to use especially with common ARM microcontrollers and also has an included C/C++ compiler. Segger embedded Studio uses a style similar to Microsoft's Visual Studio [21].

It has been recommended for the development of Nordic nRF52 series and free to use for all Nordic Semiconductor customers. Thus we decided to use this as a development tool for our project.

1.5.3 nRFgo Studio

This is a test and programming tool available for Nordic Semiconductor products. This tool has been used to configure Firmware (SoftDevice) and applications on the development board. It also provides a means to evaluate the radio performance and functionality of the device.

1.5.4 nRF Toolbox smartphone application

The nRF Toolbox is a container app that stores Nordic Semiconductor apps for BLE in one location. The nRF Toolbox works with a wide range of BLE accessories. It contains applications for Health Thermometer Monitor, Glucose Monitor, Proximity Monitor, Nordic UART, etc. It is compatible with Nordic Semiconductor nRF5 Series devices that have the SoftDevice and bootloader enabled.

In this implementation reading and writing to the implanted device is performed using Nordic UART application. So the implanted device

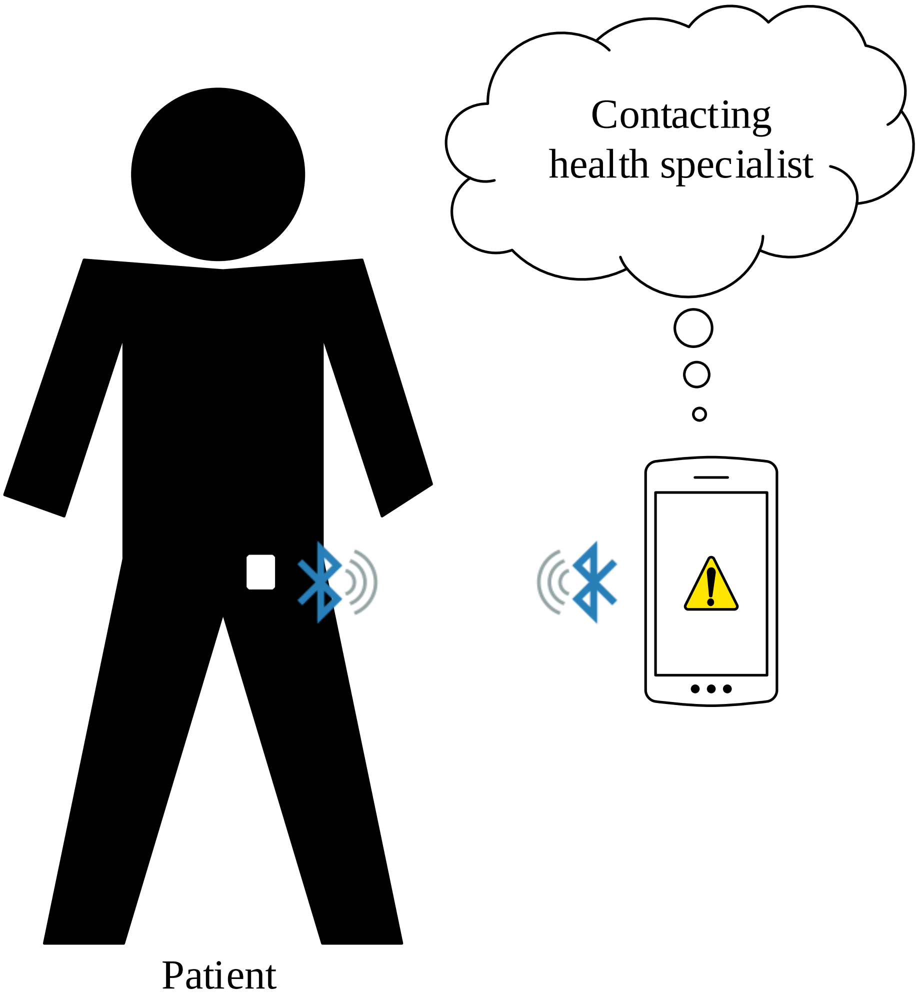


Figure 1.1: Usage illustration

and smartphone application will have a peripheral and central role respectively.

1.6 Usage Illustration

Figure 1.1 shows how a complete system would work. The goal is to design a small implantable device, which can communicate with smartphone applications through BLE. In this thesis, we are focusing on the design and implementation of the implanted device.

Part II

The Project

Chapter 2

Hadrware System

In this section, we will discuss the hardware requirement for the proposed system. We will give a brief description of our hardware choice and a demonstration of the system's block diagram.

2.1 System On a Chip based Solution

As it was discussed in the previous chapter (section 1.3), in this thesis, we are interested in embedded system based implementation. Preliminarily, it was microcontrollers that are mainly used in the embedded system applications, but nowadays SoCs are rising to prominence in the embedded systems market. Some of the reasons behind this are that SoCs offer better reliability, smaller footprint, and lower cost [22]. They are also much more efficient as systems since their performance is maximized per watt [23].

An SoC is essentially an integrated circuit or an IC that takes a single platform and integrates an entire electronic or computer system onto it. The components that an SoC generally looks to incorporate within itself include a central processing unit, input and output ports, internal memory, as well as analog input and output blocks among other things [23].

Our design is based on the nRF52840 SoC which is a member of Nordic Semiconductor nRF52 Series SoC family. The nRF52 Series of SoC devices embed a powerful yet low-power ARM® Cortex™-M4 processor with 2.4 GHz RF transceivers. This family series enables us to make ultra-low power wireless solutions.

nRF52840 contains 1 MB of flash and 256 Kb of RAM that can be used for code and data storage. Having non-volatile memory is a big advantage in this chip. It will make it impervious to memory loss during a power-down event. It also contains an integrated 12-bit ADC which will save us from adding additional components which could have a negative effect on the size of the design. It also contains a full-speed 12Mbs USB device controller for data transfer and a power supply for battery recharging.

2.2 Amplifier

The voltage swing of the pressure sensor is in the 100mv range. To increase the accuracy of ADC, sensors output should match its input range. So the data must be amplified before entering the ADC. To eliminate input impedance matching, using an instrumentation amplifier can be a good choice. These kinds of amplifiers are also widely used in measurement and test equipment. In this design, Texas Instruments INA333 has been chosen. It is a low noise, low distortion instrumentation amplifier. The gain of this amplifier can be tuned with an external resistor and can reach up to 1000.

2.3 Components Summery

The main components needed for this design are nRF52840 SoC and an amplifier. But we eventually need to add some other components such as power source and voltage regulators. Table 2.1 shows the proposed component, their power consumption, and prices. Amplifier power consumption is extracted from its datasheet. nRF52840 power estimation has been calculated using the Nordic Semiconductor online power profiler. This power profiler gives an estimation based on some chip settings like connection interval, supply voltage, and transmitted payload per event. Figure 2.1 shows an example of proposed system implementation.

Component	**Manufacturer**	**Name**	**Power Consumption**	**Price (unit of 100)**
SoC	Nordic Semiconductor	nRF52840	6.4 μA	41.63 NOK
Amplifier	Texas Instrument	INA333	50 μA Quiescent	26.92 NOK

Table 2.1: Utilized components to be used in the project

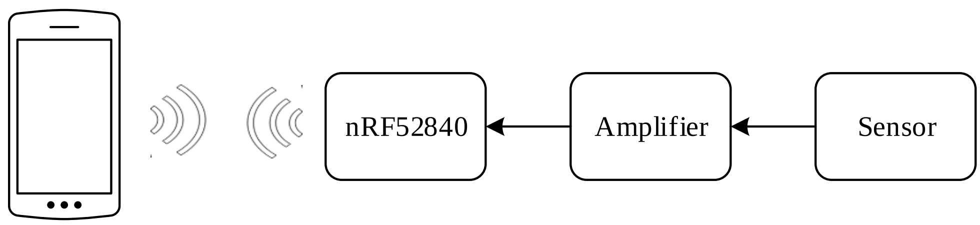


Figure 2.1: Block Diagram of proposed system

Chapter 3

Software implementation

3.1 System Overview

Figure 3.1 is a top-level overview of the proposed system. Each block outside of nRF52840 is a separate discrete component. Inside the nRF52840 block, it is demonstrated how different peripheral are connected to each other and to the main processing unit. In this diagram, only peripherals that were in use in this implementation are illustrated. A complete block diagram of nRF52840 is presented in Appendix A.

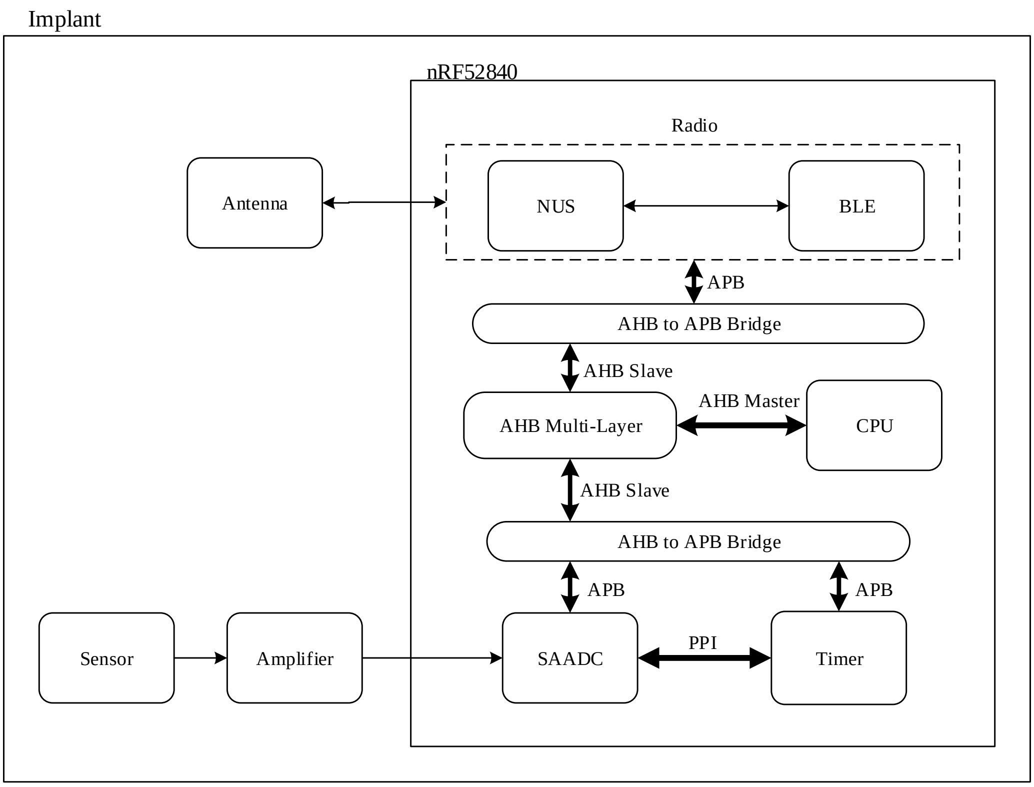


Figure 3.1: System Diagram

Sensor's data is amplified to reach the input range of Successive Approximation Analog to Digital Converter (SAADC). Different peripherals inside the nRF52840 can communicate with each other through Parallel Peripheral Interface (PPI). In this way events from a peripheral can trigger a task in another peripheral. For instance, in this design, the Timer module will trigger the sampling task in the SAADC module. Peripherals also can have access to memory without CPU intervention, since nRF52840 uses Direct Memory Access (DMA) feature. This feature gives peripherals direct access to RAM whenever they are needed. SAADC module also uses the DMA feature to store conversion result inside the buffers without CPU intervention. The conversion result will then feed to the NUS unit which will send these data through BLE channels to the smartphone application.

There is a unit inside the nRF52840 named Advanced High-performance Bus (AHB) Multi-Layer. This unit enables parallel access paths between multiple masters and slaves in the system. Access is resolved using priorities. The CPU and all of the DMAs are AHB bus masters on the AHB multilayer, while the RAM and various other modules are AHB slaves.

The software implementation is done in the Segger Embedded Studio development platform [24]. After building the application, the application file together with Softdevice is flashed to SoC through a USB interface and onboard Segger J-Link debugger.

In the following sections, we will give a detailed description of the role of each unit in Figure 3.1 and how these units are working together to shape the final product.

3.2 Handling Radio Communication

To handle the radio (here Bluetooth) communication, several components need to work together to create a successful data channel. These components are SoftDevice, BLE event handler, and Nordic UART Application. In the following sections, we will give a detailed description of each component's task.

3.2.1 SoftDevice

SoftDevice is an Application Program Interface (API) that handles the BLE stack and radio events [25]. In this thesis, Nordic's nRF SoftDevice was utilized to handle the wireless communication part of the design. nRF SoftDeive provides a wrapper to BLE stack protocols which facilitates transferring and receiving data through BLE.

To use SoftDevice's functionality in our system, the first step is to initialize the SoftDevice. A separate function has been implemented to achieve this goal. Inside this function, a request is sent to enable the device. After enabling the SoftDevice, the BLE stack is configured. This helps the application to have access to the BLE stack functionality through SoftDevice. Now the SoftDevice and BLE stack is enabled but an important step still remains. For our modules to be notified about SoC events we need to register an event handler inside the SoftDevice. Thus, the SoftDevice can perform proper action based on generated events. *NRF_SDH_BLE_OBSERVER* macro is used to register an observer for BLE stack events.

```
NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO,
    ble_evt_handler, NULL);
```

ble_evt_handler is an observer in this implementation that monitors the BLE stack events. In the next subsection, we will give a detailed description of this event handler.

3.2.2 BLE Events Handler

It was mentioned in the previous section that a BLE event handler is registered inside the SoftDevice to handle the events coming from the BLE stack. This event handler catches different events that happen during the communication and sends a proper response concerning that event. Figure 3.2 shows how this event handler responds to each event. This event handler does not provide a specific response to all received events from the BLE stack. Basic events to handle a BLE connection have been considered.

There are two events in this event handler that we would like to elaborate more in details. The first event is PHY update request. It was mentioned earlier, the newer version of BLE supports two new PHY variants. For instance, the default PHY bit rate is 1Mbps, but if the central supports 2Mbps bit rate, it will send a request to update the PHY variant. The event handler will catch this request and perform a proper action.

Another event is related to security key exchange. In this design, the Low Energy Secure Connections (LESC) pairing model with the Just Works pairing method was used during the implementation. This pairing method is based on the Diffie-Hellman key exchange method. In this method, the parties provide each other with their public keys and there is a private key for each party which is not shared. When the package is sent each party uses its own private key and the other party's public key to calculate the message. Therefore, this model is secure against pas-

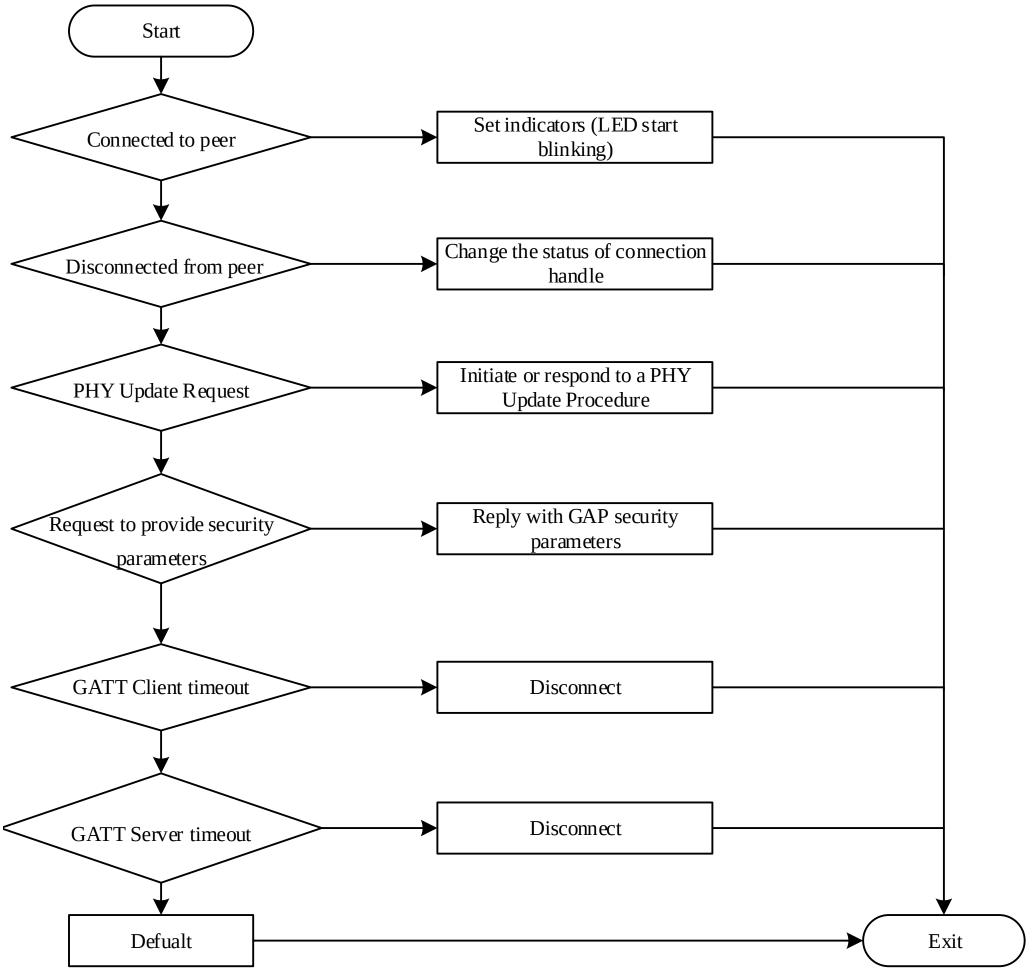


Figure 3.2: Overview of BLE event handler module

sive eavesdropping [26]. Nevertheless, this method does not provide an authentication method, thus it is likely to be vulnerable to Man In The Middle attacks(MITM).

In order to guard against the MITM attack, one can use the passkey pairing method. This method, however, requires a monitor and an onboard keyboard which is not possible to include in this implementation due to it is being an implantable device.

3.2.3 Nordic UART Service

We need our software to support both the peripheral (Development board) and the central (Android/PC/iOS) sides. One of the Nordic services that provides this capability is the Nordic UART Service (NUS). NUS Application emulates a serial port over BLE. To support the basic UART communication requirements, it sets up two RX and TX data channels with write and notify properties respectively. Thus, data received from a peer through BLE (sensor's data for instance) is passed to the NUS application and subsequently to the module that concern with these data. The same will be applied when sending data to peer, data first will be passed to the NUS application and then from NUS to the peer.

3.3 ADC Implementation

We start the ADC configuration by calling *nrf_drv_saadc_calibrate_offset()* function. This function is available in Nordic Software Development Kit (SDK) and will trigger the ADC offset calibration. It has two return values: *NRF_SUCCESS*, when calibration is started successfully, and *NRF_ERROR_BUSY*, when the ADC driver is busy or calibration is already in progress. We can find out if the calibration is done successfully by catching the return value of this function.

In the following subsection, we will discuss about detailed configuration of ADC driver, sampling channels, and how ADC peripheral is communicating with other peripherals in the system.

3.3.1 Initializing Sampling Event

In order to trigger the compare event, we have set up a timer. In this implementation, sampling is happening every 250 milliseconds. The SAADC peripheral is configured along with buffers for storing samples directly in RAM. As we can see in Figure 3.1, Timer and SAADC modules are communicating through PPI channels. After allocating the first available PPI channel, a sample task and a compare event addresses have been assigned to the PPI channel. Whenever a compare event happens in the Timer module, the sampling task is triggered in SAADC without any intervention from the CPU. When the SAADC sample task has been triggered enough times to fill the buffer, an END event is generated by the peripheral. This END event then triggers the interrupt request handler inside the SAADC driver. Interrupt request handler calls the SAADC callback function which processes the samples and setup the buffers for reuse.

3.3.2 SAADC Configuration

Table 3.1 shows the ADC driver configuration. Oversampling is disabled as well as low power mode. The reason behind disabling the low power mode is that when the low-power mode is enabled, the CPU is required to trigger the sampling, and it will only work with a buffer size of one sample. In the case of a high sampling rate, there is little or no benefit from the

Option name	Configuration
Resolution	12-bit
Oversampling	Disabled
Low power mode	Disabled
Interrupt priority	6

Table 3.1: SAADC driver configuration

low-power mode. In this implementation, we have more than one sample in each buffer. Therefore, we decided to disable this mode.

After configuring the ADC driver, each sampling channel needs to be configured as well. In this implementation, we have used only one sampling channel. Table 3.2 shows the ADC channel configuration. Burst mode has been disabled to decrease power consumption. The ADC is using the interrupt mode. A timer starts the ADC conversions and an interrupt is generated when the conversion is over.

Option name	Configuration
Gain	$\frac{1}{4}$
Reference voltage	Internal 0.6 volt
Mode	Single ended
Burst	Disable

Table 3.2: SAADC channel configuration

Figure 3.3 shows the overall flow diagram of the ADC implementation. The conversion results are added to a buffer, then the buffer's data are passed to the NUS application. Then, the data will be sent through BLE channels to the mobile application or any other available receiver. BLE allows us to transfer a maximum of 20 Bytes in each transmission. Therefore, we have to make sure to avoid sending data larger than 20 Bytes [27]. At the end of each transmission, a call to the error handler is made to confirm a successful transmission.

3.4 Timer

This unit is responsible for managing time for the entire system as well as handling the advertising intervals. Every time a timer initializes inside this block it will stop any real-time controller, to prevent timers from expiring in case of re-initializing. Setting an interrupt priority and enabling interrupts are also happening in this unit.

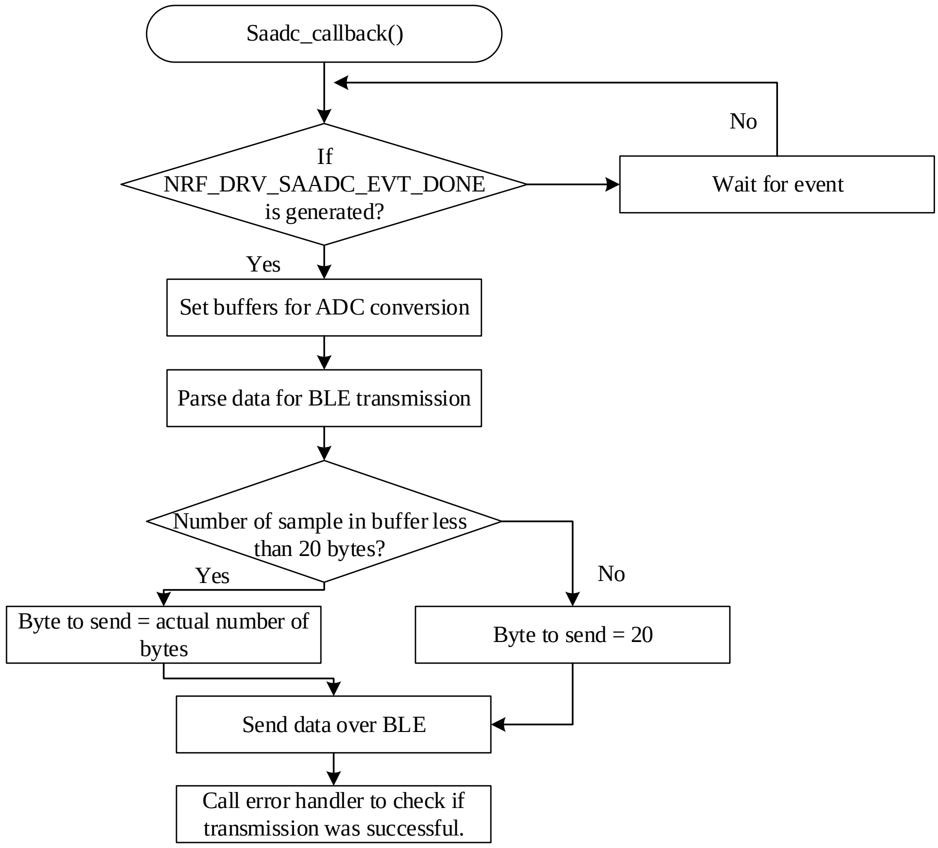


Figure 3.3: Simplified flow diagram of SAADC implementation

As we can see in Figure 3.1, Timer is communicating with other peripherals in the system through PPI channels. PPI system allows a timer event to trigger a task of any other system peripheral of the device. The PPI system also enables the timer task/event features to generate periodic output GPIO.

The timer runs on the high-frequency clock source (HFCLK) and includes a four-bit Prescaler that can divide the timer input clock from the HFCLK controller. The timer frequency is derived from *PCLK16M* as shown in the following equation, using the values specified in the Prescaler register:

$$f_{TIMER} = \frac{16MHz}{2^{Prescaler}} \tag{3.1}$$

Clock source selection between *PCLK16M* and *PCLK1M* is automatically selected according to the Timer base frequency which is set by the Prescaler. In this implementation, the timer is running on base frequency (Prescaler = 0).

3.4.1 Timer for Periodic Advertisement

To decrease power consumption, we implemented a periodic advertisement. So the device will send data in pre-specified time intervals. Selecting a correct timer mode is the first step.

There are two timer modes available:

- Single Shot mode: the timer would expire only once.
- Repeated mode: the timer would restart each time it expires.

To achieve a periodic advertisement, we have used the repeated timer mode. Using repeated mode and timeout handler function, the advertisement will restart each time it expires.

Figure 3.4, shows a flow diagram of the advertising timer module. There are two user defined variables in this diagram:

- *ADV_TIMER_INTERVAL*: Specifies the timeout ticks of the advertising timer.
- *APP_ADV_DURATION*: Specifies the advertising duration.

We can control the advertising and sleep duration of the system by setting these two variables to desired values.

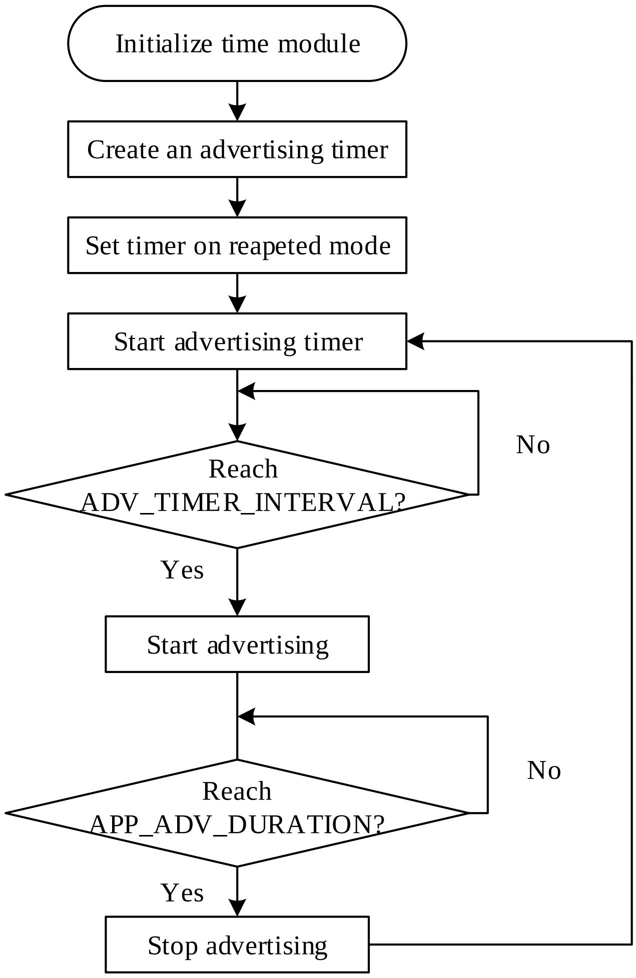


Figure 3.4: Simplified flow diagram of advertising timer module

3.5 Power Management

The last part of the implementation is handling the system's power during the IDLE state.

Based on nRF52840 specifications, there are two main power saving modes available in the IDLE state: System OFF and System ON modes. System OFF mode is the deepest power saving mode that the system can enter. In this mode, all core functionalities are powered down. All clock sources and peripherals on the chip are therefore non-functional or non-responsive. We can only wake up the system from System OFF through an external power, and not on a timer interrupt for instance, because as we mentioned earlier we cannot run timers in System OFF. In this implementation, we have an advertising timer that can run constantly. Therefore, our system cannot enter the System OFF mode at all.

In our design, we keep our system in System ON mode. System ON is the default state after powering on the system. In this mode, all functional blocks such as the CPU or peripherals can be in IDLE or RUN mode, depending on the configuration set by the software and the state of the application executing. System ON has a sub power mode named Low Power mode which is the power mode we are using in this implementation. In this mode, the system can switch the appropriate internal power sources on and off, depending on how much power is needed at a given time. The power requirement of a peripheral is directly related to its activity level, and the activity level of a peripheral is usually raised and lowered when specific tasks are triggered or events are generated.

To leverage this functionality in our system, when both the CPU and all the peripherals are in IDLE mode, the system will enter CPU sleep mode by using *sd_app_evt_wait* function. This will place the chip in Low Power mode. The chip will wake up from this mode on application interrupts. *sd_app_evt_wait* is called in an infinite loop in the main function. It will return when an application interrupt has occurred, thereby allowing the main thread to process it if needed. In this mode, we can restart the system using the timer interrupts.

Chapter 4

Printed Circuit Board Design

In this chapter, we will discuss different parts of the Printed Circuit Board (PCB) design. The main components of this design are nRF52840 SoC, an amplifier, power sources, and a voltage converter unit that provides different power supplies in different parts of the design.

4.1 Custom PCB

The importance of making a costume PCB is in making the system implantable. So far, testing has been performed on nRF25840 Development Kit (DK). As we can see in Figure 4.1 the nRF52840 DK is too large to be implanted and it makes sense because it is designed for development purposes and as we can see there are too many components (buttons, switches, LED, ports, additional MCU) which are not in use for our final system. Moreover, during the testing process, input signals are simulated sensory data with a much higher range than the actual sensor output. Therefore, an amplifier must be added to the system to get an output in the ADC range. Fortunately, the nRF52840 SoC itself is very small (component inside the red rectangle) and even though it requires other ICs to function, the overall package is still small enough for our purpose. Implantable PCBs have specific packaging requirements. Flexible substrate and cover masks are often selected in medical industries. This is because medical devices often do not conform to typical standards of PCB shape and size, and medical device professionals want to make sure their PCBs can fit into as small of an area as possible while remaining resistant to damage [28].

Another important factor to consider when creating a custom PCB is the environmental factor. Environmental factors can have a direct impact on PCBs performance. In this design, temperature and humidity are two environmental factors to consider. Choosing an adhesive with a low water absorption level can prevent any damage related to humidity in a longer

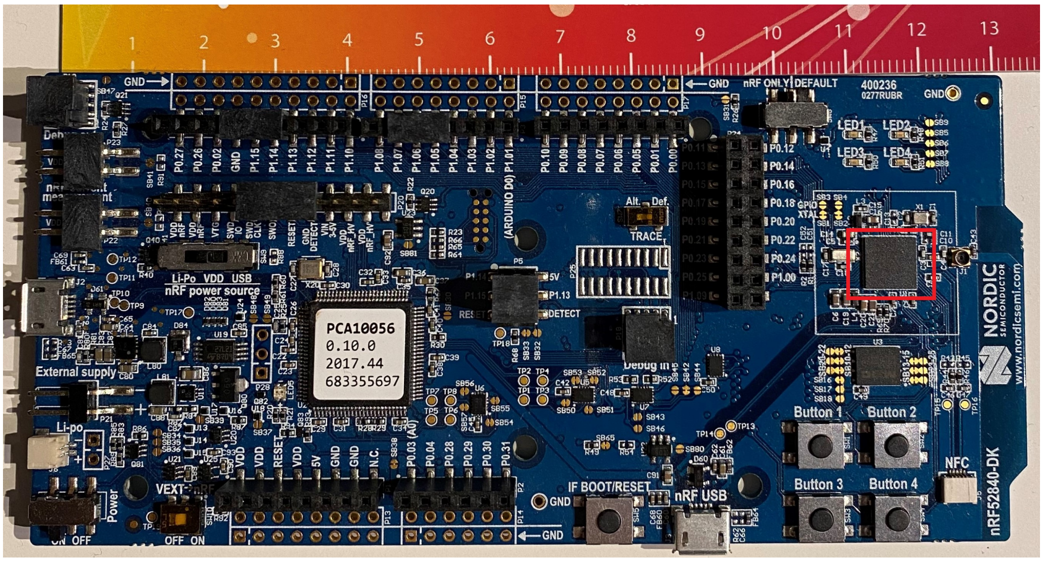


Figure 4.1: Overview of BLE event handler module

period. For long-term implants, it is important to use bio-compatible materials to reduce the chance of infection [29]. We can increase the long term reliability of the implantable device by utilizing encapsulation techniques. Encapsulation materials vary from a range of inorganic materials like aluminium oxide and silicon dioxide, organic polymers of polyimide, parylen and biocompatible materials [30].

At this point, after discussing the important factors to consider when making a custom implantable PCB, we will discuss about main components to include in our proposed PCB.

4.2 Power Source

Micro USB connector initially used for programming the SoC but it also can provide a 5-volt source voltage. However, since this design is meant to be an implantable device, a battery can be added to the design as a power source.

4.3 Amplifier

To increase the accuracy of analog to digital conversions, the ADC should receive input within its range. The voltage swing of the pressure sensor is in the $100mv$ range, therefore, an amplifier should be included in the design. The INA333 amplifier is chosen for this purpose. It is a small size, low power, low noise, and low distortion instrumentation amplifier. Instrumentation amplifiers are widely used in measurement and test equipment. The gain of this amplifier can be tuned with an external resistor and can reach up to 1000 [31].

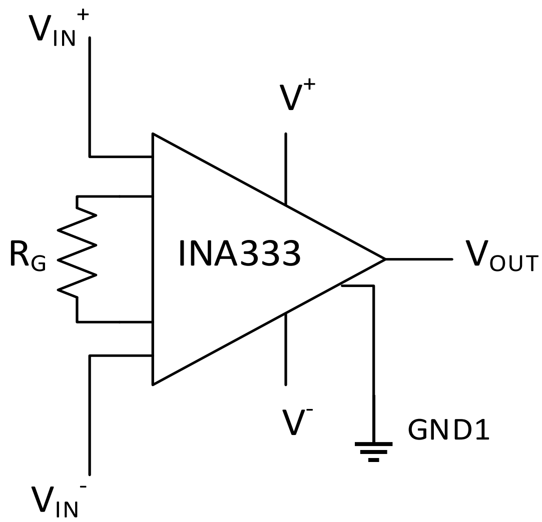


Figure 4.2: Simplified form of INA333 amplifier

Figure 4.2 shows a simplified form of INA333 amplifier. In this amplifier gain is set using an external resistor. Equation 4.1 shows gain equation for the INA333.

$$G = 1 + \frac{100K\Omega}{R_G} \tag{4.1}$$

The maximum accepted input voltage of ADC depends on multiple factors (Subsection 4.5.1). After calculating the maximum input range of ADC using equation 4.2 , it is straightforward to calculate R_G from equation 4.1.

4.4 Antenna

When it comes to choosing an antenna for our design, there are two common implementations: Ceramic chip antennas and PCB trace antennas. Even though PCB trace antennas are relatively low price as the trace is applied as part of the PCB assembly process, they are difficult to design, implement, and tune. Their performance is highly affected by the PCB design, even minor changes on the PCB can cause frequency detuning which will have a negative effect on antenna performance. Usually, these failures are detected after the full assembly of PCB. Since the trace antenna is designed during the PCB manufacturing process the chance of optimizing its performance after it has been applied is very little [32]. Therefore, there are often several design iterations that result in an increased time to market and an associated loss of revenue. Moreover, the PCB trace antenna is relatively large. This is because the area around the trace antenna needs to be kept clear of other objects to maintain performance, so the total area required is always much greater than just the size of the trace. Thus, they are not an optimal choice for the implementations which having a small fabrication is an important factor.

On the other hand, ceramic chip antennas are much smaller than the PCB trace antennas, and different configurations of these antennas are available in the market. A ceramic antenna is a separate component that is attached after the design phase has been completed. Therefore, it will give a more versatile tuning during the development. With the surface mount feature of ceramic chip antennas, they can easily be removed and replaced in case of hardware modifications.

The Fractus® Compact Reach Xtend™ chip antenna is used in this implementation. It is designed specifically for Bluetooth and other wireless devices that operate at the ISM 2.4GHz band. It has a small footprint that allows integration of the antenna into limited space easily and efficiently with minimum clearance area [**AN048**].

4.5 nRF52840 SoC

nRF52840 is the main processing unit in this design. It also includes the 12 bit ADC and 1 MB of flash and 256 Kb of RAM for data and code storage. Therefore, this will eliminate the use of external flash memory and ADC in the design which can have a positive impact on the overall design size as an important factor in implantable solutions.

4.5.1 Analog to Digital Converter

The ADC inside the SoC is a differential Successive Approximation Register (SAR). The ADC supports up to eight external analog input channels, depending on the package variant. The analog inputs can be configured as eight single-ended inputs, four differential inputs, or a combination of these [33]. ADC can operate in a one-shot mode with sampling under software control or a continuous conversion mode with a programmable sampling rate. In this implementation, one sampling channel in single-ended mode is used.

Each SAADC channel can have individual reference and gain settings. Available configuration options are:

- $\frac{V_{DD}}{4}$ or internal 0.6 V reference
- Gain ranging from $\frac{1}{6}$ to 4

The gain setting can be used to control the effective input range of the SAADC:

$$\text{Input range} = \frac{(\pm 0.6 \text{ or } \pm \frac{V_{DD}}{4})}{Gain} \tag{4.2}$$

In this implementation, an internal 0.6-volt reference voltage was chosen so that the input range of the ADC is independent of V_{DD} voltage. We selected $\frac{1}{4}$ as SAADC gain. Thus, based on equation 4.2 the input range for the ADC is between 0 to 2.4 volt.

Equation 4.3 is used to calculate the output result of the ADC. The conversion result depends on configurations on each channel. In this equation, V_P and V_N are the voltage at positive and negative input respectively. Since we are using the single-ended mode negative input is grounded.

$$Result = (V_P - V_N) * \frac{Gain}{ReferenceVoltage} * 2^{Resolution} \tag{4.3}$$

4.5.2 Clock Sources

The system clocks are sourced from a range of internal or external high or low-frequency oscillators. A clock control system distributes them to various modules depending on their requirements. Clock distribution is automated and grouped independently by the module to limit current consumption in unused branches of the clock tree. Figure 4.3 shows an overview of the clock control module inside the nRF52840.

To source the clocks used in this implementation, two 32.768kHz and 32MHz crystal oscillators are added to the design. As it is shown in Figure 4.3, these clock sources are used by the HFCLK and LFCLK controllers to provide clocks to the system. These clocks are used by different peripheral in this implementation such as Timer, UART, and ADC.

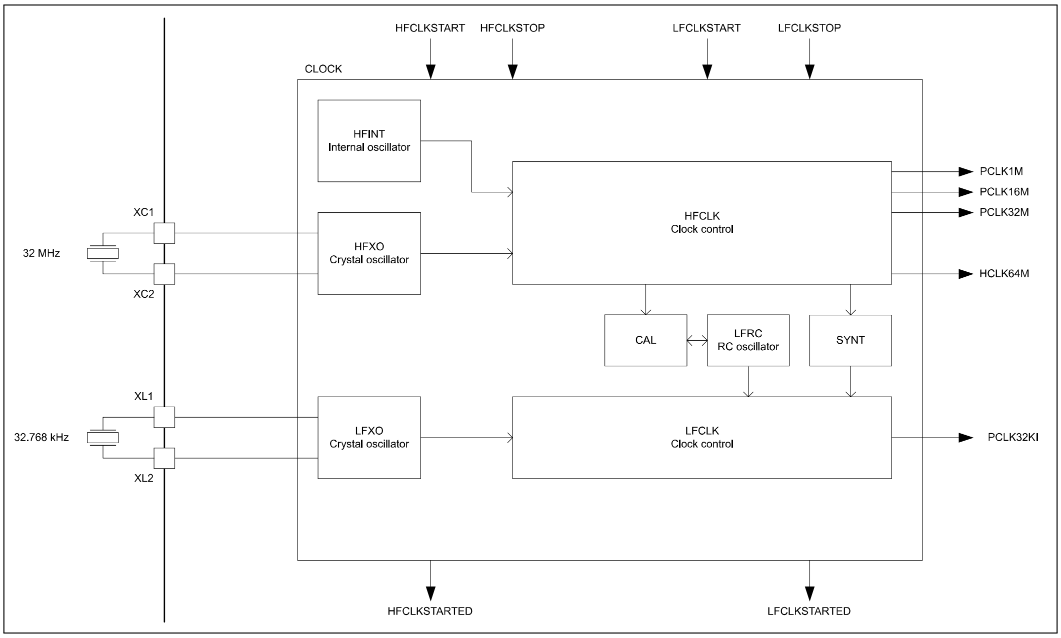


Figure 4.3: Clock Control System [33]

4.6 Schematics

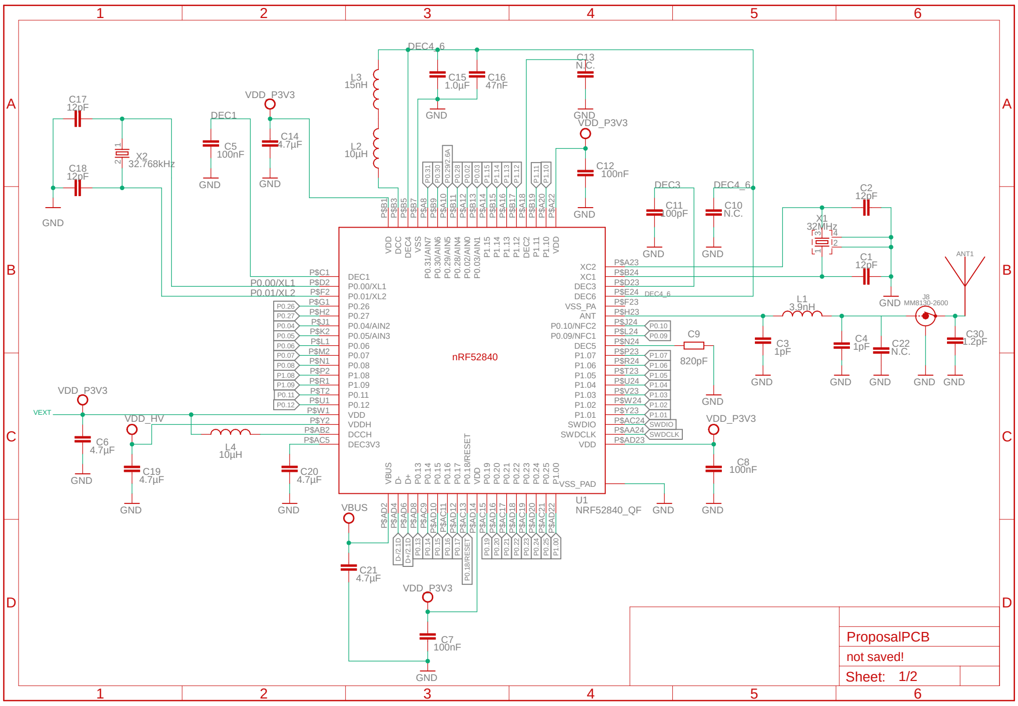


Figure 4.4: nRF52840 and RF setup

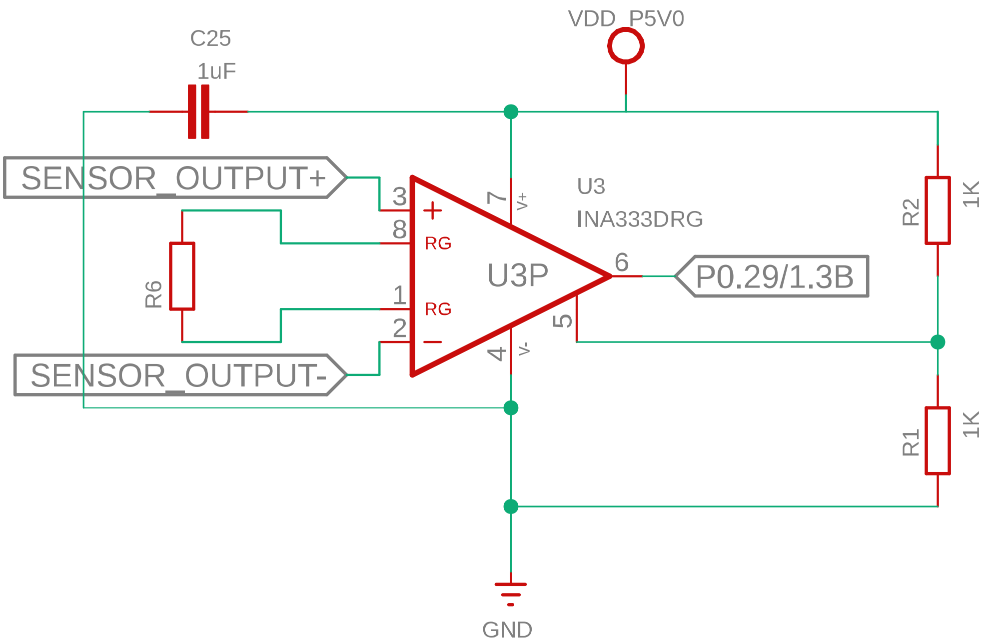
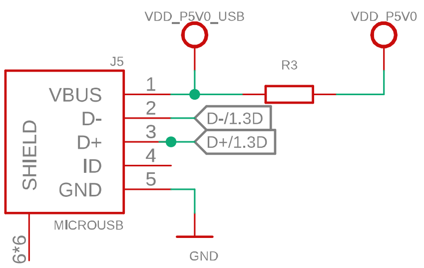
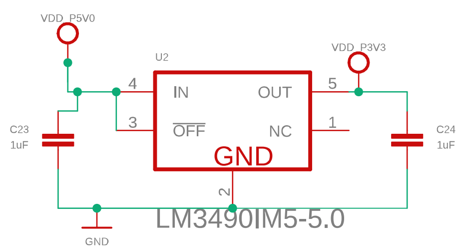


Figure 4.5: Amplifier



(a)



(b)

Figure 4.6: Schematics of power source (4.6a) and power regulator (4.6b) circuits

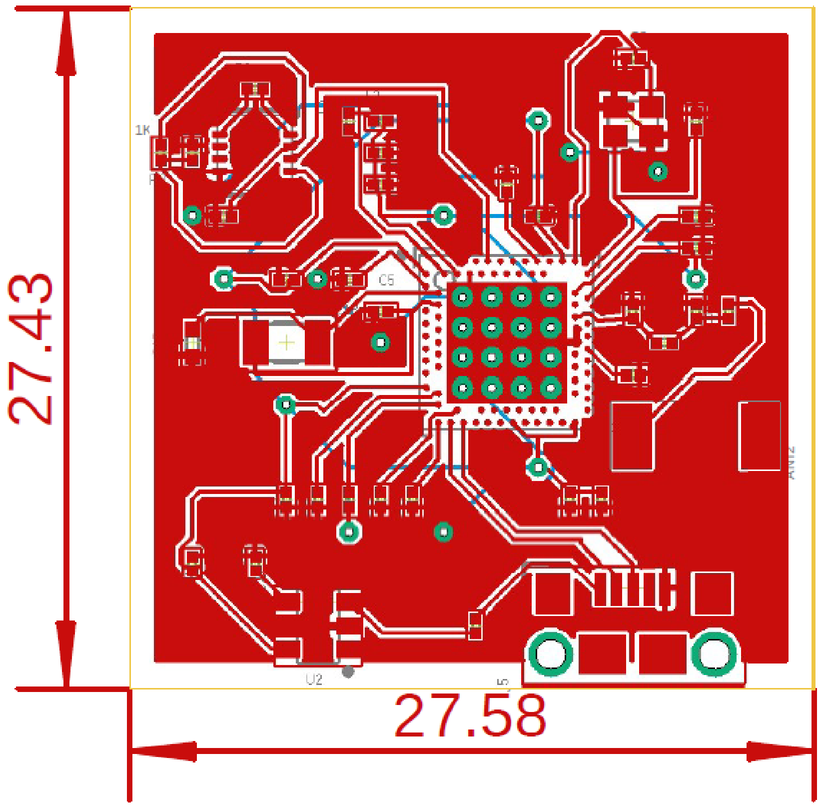


Figure 4.7: PCB Layout, dimensions in milimeter

Part III

Results and Discussions

Chapter 5

Measurements

We have performed two types of measurements using the development board. These measurements include the power consumption and ADC sensitivity. This section presents these results. We also compare these measurements with the available implementations.

5.1 Power Consumption

As it is mentioned earlier, one of the important reasons to choose BLE as a communication protocol is its low power characteristics.

nRF52840 is the main processing unit in this implementation, it is managing the Timer, ADC, advertising, and the transmission through BLE. Therefore, it is the main power consuming unit in this implementation. To perform the current measurement, we need to split the power domains for the nRF52840 SoC and the rest of the board. This was done by shorting a solder bridge on the development board. Then current consumption is measured by connecting a multimeter to the dedicated current measurement pins on the development kit. Table 5.1 shows the current consumption and the life span of a small 600mAh battery in two operating modes. Estimated life span is calculated using the following equation:

$$Battery\ Life = \frac{Battery\ Capacity\ in\ mAh}{Load\ Current\ in\ mA} \tag{5.1}$$

Operation Mode	**Not advertising**	**Advertising**
Current Consumption	0.915 mA	1.114 mA
Life span for 600 mAh battery	26 Days	23 Days

Table 5.1: Current consumption of nRF52840

Table 1.1 shows that the current consumption is higher during the advertisement. This is expected since the system continuously sends data.

However, the difference in the current consumption in these two operation modes is relatively small. As we mentioned earlier the system never enters the System OFF mode. When in the IDLE state the device stays in System ON mode low power state to be able to control the advertising with interrupts from the timer module. The main contributors to the current consumption in this implementation are modules that are running on HFCLK such as Timer, UART, and SAADC. During IDLE state the advertising timer is still running, therefore, this will keep the current consumption high.

A comparison between our implementation and some other devices detailed in the references ([9],[10],[7]) is presented in Table 5.2. Table 5.2 illustrates that the current consumption of our method is lower than that of [9] and [10] but higher than [7]. The reason for this extra consumption can be contributed to usage of HFCLK which requires additional power. However, it should be noted that communication with the implantable device in [7] requires an additional based station outside the human body to be able to receive the transmitted data. In our design using BLE standard is a big advantage. It is a standard which is available in smartphones, tablets, laptops, and other electronic devices. In this design, we are able to read the values directly from the sensor using a smartphone. Moreover, BLE will provide a device with a higher range than other standards. This makes it possible for health specialists to monitor for instance the bladder pressure from other rooms.

Parameter	**[10]**	**[9]**	**[7]**	**This work**
Frequency (MHz)	2.7	434	402-403	2400-2483
Standard	N/A	N/A	MICS	BLE
Range (meter)	0.3	0.1	2.5	>10
Current Consumption(mA)	3.01	1.78	0.641	1.11
Life span (day)	8.3	14	39	23

Table 5.2: Performances Summery

5.2 ADC Results

To validate the ADC results, we performed a basic measurement. A multimeter was used to measure the input voltage of analog pins of the development kit. The input voltage was changed from zero to 2.4 volt with a step of 100mV.

The digital output was read out from the smartphone application. Conversion results are stored inside the buffers. We have sent these buffer data as string values to the smartphone application so these values can be read without performing any conversion.

Table 5.3 shows these measurement results. Theoretical values are calculated using equation 4.3. The third column shows the ADC conversion results read out from the smartphone. The output of each measurement changes for every sample. Therefore, for each analog input, 10 observations have been made and the average value is documented.

The comparison between the theoretical results and the observed ADC outputs show average of 1 bit deviation. Equation 5.2 shows that for this input range a 12-bit ADC is supposed to have 0.585 mV accuracy. Thus, our results are in accordance with the expected accuracy of 12 bit ADC.

$$\frac{2400\ mV}{2^{12}} = 0.585mV \tag{5.2}$$

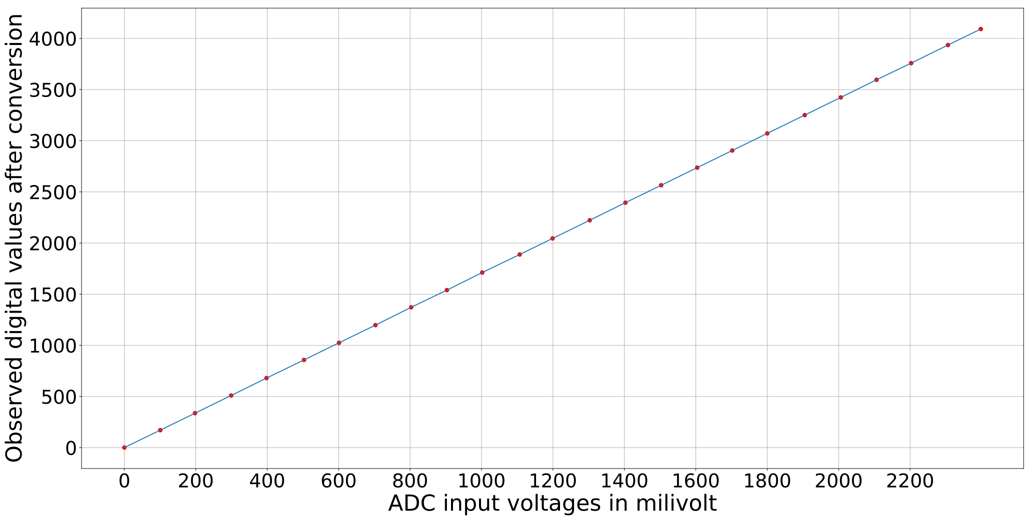


Figure 5.1: Plot of data on Table 5.3

Voltage (mV)	**Theoretical**	**Average**
GND	0	0.7
100.5	171.5	170.9
197.8	337.5	336.8
299	510.3	509.8
398	679.25	679.9
503	858.45	857.3
601	1025.7	1024.8
703	1198.08	1197.6
803	1370.45	1371.7
903	1541.12	1539.5
1002	1710.08	1711.58
1107	1889.28	1887.83
1199	2046.29	2045
1303	2223.78	2222.5
1403	2394.45	2394.55
1503	2565.12	2564.8
1604	2737.49	2737
1702	2904.74	2904.1
1800	3072	3071.3
1905	3251.2	3250.45
2006	3423.57	3423.3
2107	3595.94	3595.8
2203	3759.78	3758.3
2306	3935.57	3935.14
2398	4092.58	4091.6

Table 5.3: Theoretical and averaged measurement results of ADC

Chapter 6

Conclusion

In this thesis, we presented an embedded system based solution for implantable bladder pressure sensors using BLE communication protocols. With the help of this device, we can reconstruct or restore the bladder function in patients with neurogenic bladder dysfunction.

Tests and development are done using nRF52840 Development Kit. We have been able to configure the SoC to perform an analog to digital conversion on signals coming from analog pins and send the results to a smartphone application through Bluetooth channels.

A current consumption of $0.915mA$ was achieved in the IDLE mode and $1.114mA$ in the operating mode. We also demonstrated that with using a small 600 mAh battery a life span of 23 days on advertising mode is feasible. This makes deployment of the implantable sensor devices more practical inside and outside of the hospital. With this system, patients can monitor their bladder pressure everywhere and at any time. It makes it also possible for them to take proper action and avoid risks introduced by excessive bladder pressure.

We also made a primary implantable PCB proposal. The estimated size of the PCB design was $27.43mm \times 27.58mm$. In this implementation, we used 0201 and 0402 package size which helped us to achieve a relatively small size. However, we skipped building the actual PCB and verifying its functionality.

6.1 Limitations and Future Work

6.1.1 Software Implementation

One of the major concerns in health monitoring applications is security issues. In this implementation, the Just Works pairing method was used. This pairing method is quite vulnerable to active eavesdropping and MITM attacks. To achieve more secure communication Out Of Band (OOB) pairing method can be used. In this method, authentication is done outside the BLE communication channel for example using NFC. As NFC requires the devices to be in close proximity, it avoids the MITM issues and prevents unwanted devices from connecting without the user's knowledge or permission. Security keys for pairing information can be sent through NFC and BLE pairing can happen only after successful authentication [34]. One possible implementation would be to use a combination of NFC and BLE protocols. A passive NFC tag can be used as a means of authorized pairing and after pairing is performed BLE would take control of data transmission. In this way, the design would utilize both NFC security and BLE availability and convenience.

This implementation consumes a relatively high current. One of the major contributors to high current consumption is an external 32 MHz crystal oscillator which was used as a clock source for SAADC, timer, and UART peripheral. However, using this crystal oscillator is necessary for reaching high accuracy.

In this implementation, the system is always in ON mode even when it is not sending data. One way to reduce power is to add an external MCU and program it to send a wake-up signal to SoC on a specified time interval. Therefore, it is possible to reduce the time in which the high-frequency clock is running. This subsequently reduces current consumption.

6.1.2 PCB

PCB design still demands additional efforts before it can be printed and tested. Currently, the design is powered using a micro USB which is also used to reprogram the SoC. Given the fact that the design is meant to be an implantable solution using a USB interface as a power source is an unrealistic approach. Onboard batteries are a common choice for implantable devices. However, there are some limitations in the use of onboard batteries such as static energy-density, shorter lifespan, and larger size [35]. In this implementation, if we rely only on the battery, we have to change the battery quite frequently.

Another possibility is to supply implantable devices through wireless power transfer approaches. Suzuki *et al.* [36] presented a new way of supplying electric power to implanted biomedical devices. Their system was non-invasive and used two kinds of energy, magnetic and ultrasonic. It could provide high power levels harmlessly. The energies were obtained by two types of vibrators, i.e., piezo and magnetostriction devices. The internal and external magnetostriction devices were set up and biased by a permanent magnet to operate optimally. Majerus *et al.* [10] presented RF-based rechargeable battery design allows for the sensor to operate and transmit signals using a battery during the day eliminating the need for an externally worn transmitter, and then recharge the battery at night using an external coil within the patient's bed. When considering this method it is important to note the reliance on battery operation during normal measurements during the day. Therefore, specific methods should be considered to increase the lifespan of the lithium battery, such as the recharging of the battery to under the maximum power, such that the number of recharge cycles was extended. A hybrid system such as this would be able to eliminate the need for a continuous power signal and mandatory wearable external transmitter and provide a more efficient strategy for long-term continuous pressure monitoring.

Another limitation of the proposed method is that the free version of Eagle is used for PCB design. This version only allows designing a two-layer PCB. An alternative would be to follow a four-layer PCB approach which gives us two signal layers and dedicated ground and V_{CC} layer. Four-layer board will allow us to route signal, power, and ground, directly over each other in a larger variety of ways. Keeping ground directly under the power plane will reduce cross talk for close proximity lines and reduce noise by improved overall routing choices [37].

However, it is important to note that on PCBs with more than two layers, a keep-out area should have been put on the inner layers directly below the antenna matching circuitry (components between device pin ANT, and the antenna) to reduce the stray capacitance that influences RF performance [33].

The proposed PCB has a relatively small ($27.43mm \times 27.58mm$) size. However, it is possible to reach a more compact design by utilizing a four-layer PCB and using both sides of the PCB for mounting components.

6.1.3 Smartphone Application

In this project, we have used the nRFTools smartphone application which is available for Nordic products. This application is not an ideal solution for this implementation since it shows the sensor values as a binary output. Implementing a customized smartphone application that shows the pressure data as a percentage for instance is a better solution. Such an application would send a notification to the patient or health specialist before the pressure reaches a critical point.

6.2 Possible Applications

The proposed PCB with some modifications especially in power requirements can be an implantable solution in the future.

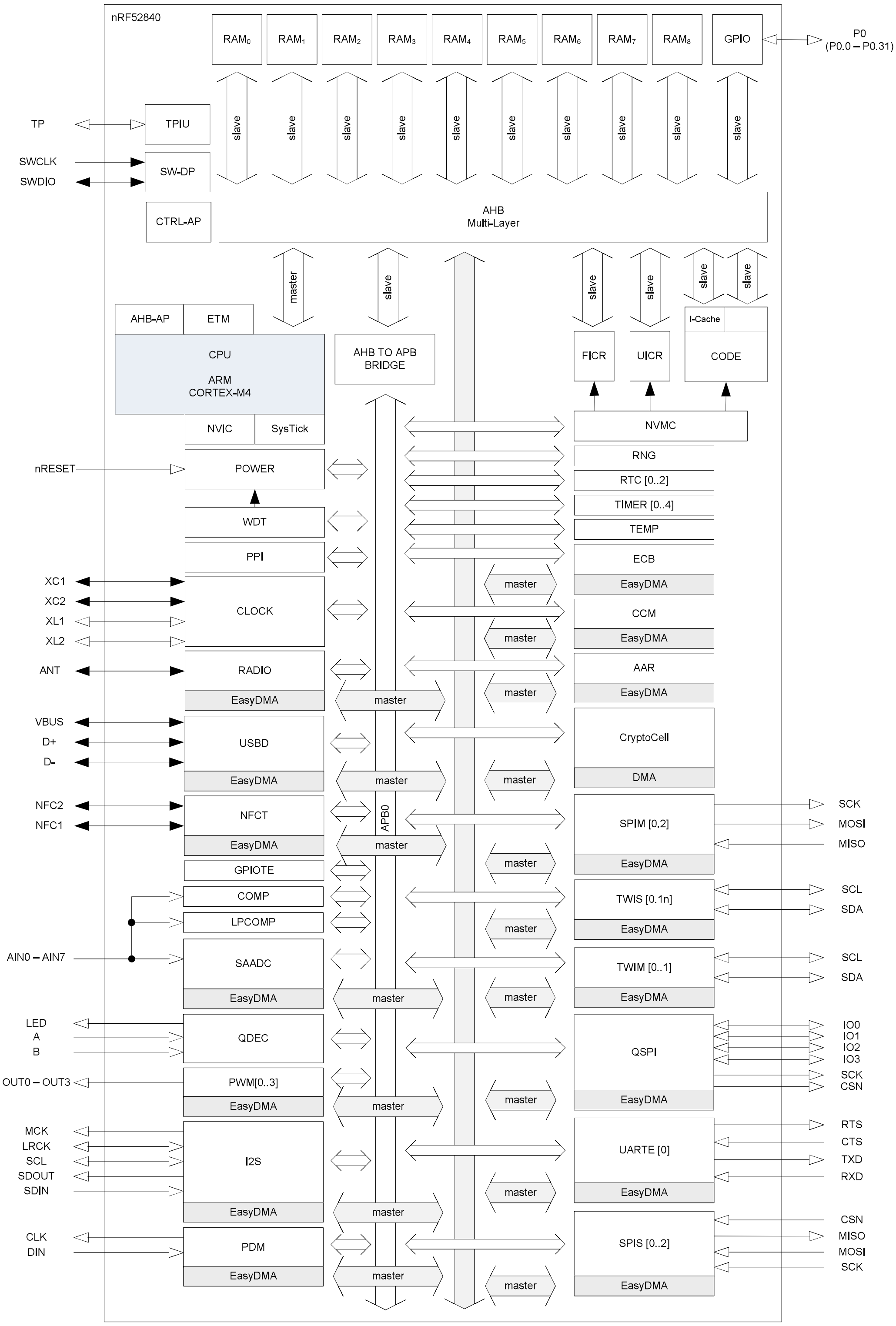
The software implementation can be used in any device that requires sample analog inputs, digitalize it, and send it through BLE channels. In this implementation, only one analog channel has been used. With small modifications in software implementation, it is possible to sample from several analog channels. This makes it possible to interact with different sensors at the same time.

The possible application for this design could be for example blood glucose sensors for diabetic patients, and bladder pressure sensors.

Appendices

Appendix A

nRF52840 Block Diagram


nRF52840
RAM0
RAM1
RAM2
RAM3
RAM4
RAM5
RAM6
RAM7
RAM8
GPIO
P0
(P0.0 – P0.31)
TP
TPIU
SWCLK
SWDIO
SW-DP
CTRL-AP
AHB
Multi-Layer
slave
master
AHB-AP
ETM
CPU
ARM
CORTEX-M4
NVIC
SysTick
AHB TO APB
BRIDGE
FICR
UICR
I-Cache
CODE
NVMC
RNG
RTC [0..2]
TIMER [0..4]
TEMP
ECB
EasyDMA
CCM
AAR
CryptoCell
DMA
SPIM [0.2]
TWIS [0,1n]
TWIM [0..1]
QSPI
UARTE [0]
SPIS [0..2]
nRESET
POWER
WDT
PPI
XC1
XC2
XL1
XL2
CLOCK
ANT
RADIO
VBUS
D+
D-
USBD
NFC2
NFC1
NFCT
GPIOTE
COMP
LPCOMP
AIN0 – AIN7
SAADC
LED
A
B
QDEC
OUT0 – OUT3
PWM[0..3]
MCK
LRCK
SCL
SDOUT
SDIN
I2S
CLK
DIN
PDM
APB0
SCK
MOSI
MISO
SCL
SDA
IO0
IO1
IO2
IO3
CSN
RTS
CTS
TXD
RXD

Appendix B

Source Code

```
#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "ble_hci.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "nrf_sdh.h"
#include "nrf_sdh_soc.h"
#include "nrf_sdh_ble.h"
#include "nrf_ble_gatt.h"
#include "nrf_ble_qwr.h"
#include "peer_manager.h"
#include "peer_manager_handler.h"
#include "app_timer.h"
#include "ble_nus.h"
#include "app_uart.h"
#include "app_util_platform.h"
#include "bsp_btn_ble.h"
#include "nrf_pwr_mgmt.h"
#include "nrf_drv_saadc.h"
#include "nrf_drv_ppi.h"
#include "nrf_drv_timer.h"
#include "fds.h"
#include "nrf_fstorage.h"

#if defined (UART_PRESENT)
#include "nrf_uart.h"
#endif
#if defined (UARTE_PRESENT)
#include "nrf_uarte.h"
```

```
#endif

#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"

#define APP_BLE_CONN_CFG_TAG        1                                           /**< A tag identifying the SoftDevice BLE configuration. */

#define DEVICE_NAME                     "Nordic_UART"                               /**< Name of device. Will be included in the advertising data. */
#define NUS_SERVICE_UUID_TYPE           BLE_UUID_TYPE_VENDOR_BEGIN                  /**< UUID type for the Nordic UART Service (vendor specific). */

#define APP_BLE_OBSERVER_PRIO           3                                           /**< Application's BLE observer priority. You shouldn't need to modify this value. */


#define APP_ADV_INTERVAL                100                                          /**< The advertising interval (in units of 0.625 ms. This value corresponds to 40 ms). */

#define APP_ADV_DURATION                6000                                       /**< The advertising duration (180 seconds) in units of 10 milliseconds. */

#define MIN_CONN_INTERVAL               MSEC_TO_UNITS(20, UNIT_1_25_MS)             /**< Minimum acceptable connection interval (20 ms), Connection interval uses 1.25 ms units. */
#define MAX_CONN_INTERVAL               MSEC_TO_UNITS(75, UNIT_1_25_MS)             /**< Maximum acceptable connection interval (75 ms), Connection interval uses 1.25 ms units. */
#define SLAVE_LATENCY                   0                                           /**< Slave latency. */
#define CONN_SUP_TIMEOUT                MSEC_TO_UNITS(4000, UNIT_10_MS)             /**< Connection supervisory timeout (4 seconds), Supervision Timeout uses 10 ms units. */
#define FIRST_CONN_PARAMS_UPDATE_DELAY APP_TIMER_TICKS(5000)                       /**< Time from initiating event (connect or start of notification) to first time sd_ble_gap_conn_param_update is called (5 seconds). */
```

```
#define NEXT_CONN_PARAMS_UPDATE_DELAY APP_TIMER_TICKS(30000) /**< Time between each call to sd_ble_gap_conn_param_update after the first call (30 seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT 3 /**< Number of attempts before giving up the connection parameter negotiation. */
#define APP_TIMER_OP_QUEUE_SIZE     4 /**< Size of timer operation queues. */

#define LESC_DEBUG_MODE                 0 /**< Set to 1 to use LESC debug keys, allows you to use a sniffer to inspect traffic. */

#define SEC_PARAM_BOND                  1 /**< Perform bonding. */
#define SEC_PARAM_MITM                  0 /**< Man In The Middle protection not required. */
#define SEC_PARAM_LESC                  1 /**< LE Secure Connections enabled. */
#define SEC_PARAM_KEYPRESS              0 /**< Keypress notifications not enabled. */
#define SEC_PARAM_IO_CAPABILITIES       BLE_GAP_IO_CAPS_NONE /**< No I/O capabilities. */
#define SEC_PARAM_OOB                   0 /**< Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE          7 /**< Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE          16 /**< Maximum encryption key size. */

#define DEAD_BEEF                   0xDEADBEEF /**< Value used as error code on stack dump, can be used to identify stack location on stack unwind. */

#define UART_TX_BUF_SIZE            256 /**< UART TX buffer size. */
#define UART_RX_BUF_SIZE            256 /**< UART RX buffer size. */
```

```
#define SAADC_SAMPLES_IN_BUFFER     10
#define SAADC_SAMPLE_RATE  500                              /**< SAADC sample rate in ms. */

#define SAADC_BURST_MODE            0                        /**Set to 1 to enable BURST mode, otherwise set to 0.*/


#define ADV_TIMER_INTERVAL              APP_TIMER_TICKS(40000)

static uint8_t           m_adv_handle = BLE_GAP_ADV_SET_HANDLE_NOT_SET;

APP_TIMER_DEF(m_advertising_timer_id);




BLE_NUS_DEF(m_nus, NRF_SDH_BLE_TOTAL_LINK_COUNT);                           /**< BLE NUS service instance. */
NRF_BLE_GATT_DEF(m_gatt);                                                   /**< GATT module instance. */
NRF_BLE_QWR_DEF(m_qwr);                                                     /**< Context for the Queued Write module.*/
BLE_ADVERTISING_DEF(m_advertising);                                         /**< Advertising module instance. */

static uint16_t m_conn_handle        = BLE_CONN_HANDLE_INVALID;                 /**< Handle of the current connection. */
static uint16_t m_ble_nus_max_data_len = BLE_GATT_ATT_MTU_DEFAULT - 3;      /**< Maximum length of data (in bytes) that can be transmitted to the peer by the Nordic UART service module. */
static ble_uuid_t m_adv_uuids[]      =                                      /**< Universally unique service identifier. */
{
    {BLE_UUID_NUS_SERVICE, NUS_SERVICE_UUID_TYPE}
};
static volatile uint8_t write_flag=0;
volatile uint8_t state = 1;

#ifdef NRF52810_XXAA
```

```
static const nrf_drv_timer_t m_timer = NRF_DRV_TIMER_INSTANCE(2);
#else
static const nrf_drv_timer_t m_timer = NRF_DRV_TIMER_INSTANCE(3);
#endif
static nrf_saadc_value_t m_buffer_pool[2][SAADC_SAMPLES_IN_BUFFER];
static nrf_ppi_channel_t    m_ppi_channel;
static uint32_t             m_adc_evt_counter;
static uint8_t              adc_event_counter = 0;
static nrf_saadc_value_t adc_buffer[SAADC_SAMPLES_IN_BUFFER];           /**< ADC buffer. */


/**@brief Function for assert macro callback.
 *
 * @details This function will be called in case of an assert in the SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final product. You need to analyse
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on reset.
 *
 * @param[in] line_num  Line number of the failing ASSERT call.
 * @param[in] p_file_name File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}

/**@brief Clear bond information from persistent storage.
 */
static void delete_bonds(void)
{
    ret_code_t err_code;

    NRF_LOG_INFO("Erase bonds!");

    err_code = pm_peers_delete();
    APP_ERROR_CHECK(err_code);
}

static void advertising_timeout_handler(void *p_context)
```

```
{
 UNUSED_PARAMETER(p_context);


 uint32_t err_code = ble_advertising_start(&m_advertising,
     BLE_ADV_MODE_FAST);
    APP_ERROR_CHECK(err_code);

}


/**@brief Function for handling Peer Manager events.
 *
 * @param[in] p_evt Peer Manager event.
 */
static void pm_evt_handler(pm_evt_t const * p_evt)
{
    pm_handler_on_pm_evt(p_evt);
    pm_handler_flash_clean(p_evt);

    switch (p_evt->evt_id)
    {
        case PM_EVT_PEERS_DELETE_SUCCEEDED:
            advertising_start(false);
            break;

        default:
            break;
    }
}

/**@brief Function for initializing the timer module.
 */
static void timers_init(void)
{
    ret_code_t err_code;

    // Initialize timer module.
    err_code = app_timer_init();
    APP_ERROR_CHECK(err_code);

    // Create advertising start timer
    err_code = app_timer_create(&m_advertising_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                advertising_timeout_handler);
     APP_ERROR_CHECK(err_code);
}
```

```c

/**@brief Function for starting application timers.
 */
//added for periodic advertising
static void application_timers_start(void)
{
    ret_code_t err_code;

    // Start application timers.
    err_code = app_timer_start(m_advertising_timer_id,
        ADV_TIMER_INTERVAL, NULL);
    APP_ERROR_CHECK(err_code);


}

/**@brief Function for the GAP initialization.
 *
 * @details This function will set up all the necessary GAP
     (Generic Access Profile) parameters of
 *         the device. It also sets the peramissions and appearance.
 */
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);
}


```

```
/**@brief Function for handling Queued Write Module errors.
 *
 * @details A pointer to this function will be passed to each
     service which may need to inform the
 *          application about an error.
 *
 * @param[in]  nrf_error  Error code containing information about
     what went wrong.
 */
static void nrf_qwr_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/**@brief Function for handling the data from the Nordic UART
    Service.
 *
 * @details This function will process the data received from the
     Nordic UART BLE Service and send
 *          it to the UART module.
 *
 * @param[in] p_evt    Nordic UART Service event.
 */
/**@snippet [Handling the data received over BLE] */
static void nus_data_handler(ble_nus_evt_t * p_evt)
{

    if (p_evt->type == BLE_NUS_EVT_RX_DATA)
    {
        uint32_t err_code;

        NRF_LOG_DEBUG("Received data from BLE NUS. Writing data on
            UART.");
        NRF_LOG_HEXDUMP_DEBUG(p_evt->params.rx_data.p_data,
            p_evt->params.rx_data.length);

        for (uint32_t i = 0; i < p_evt->params.rx_data.length; i++)
        {
            do
            {
                err_code =
                    app_uart_put(p_evt->params.rx_data.p_data[i]);
                if ((err_code != NRF_SUCCESS) && (err_code !=
                    NRF_ERROR_BUSY))
                {
```

```
                    NRF_LOG_ERROR("Failed receiving NUS message. Error 0x%x. ", err_code);
                    APP_ERROR_CHECK(err_code);
                }
            } while (err_code == NRF_ERROR_BUSY);
        }
        if (p_evt->params.rx_data.p_data[p_evt->params.rx_data.length - 1] == '\r')
        {
            while (app_uart_put('\n') == NRF_ERROR_BUSY);
        }
    }

}
/**@snippet [Handling the data received over BLE] */


/**@brief Function for initializing services that will be used by the application.
 */
static void services_init(void)
{
    uint32_t         err_code;
    ble_nus_init_t   nus_init;
    nrf_ble_qwr_init_t qwr_init = {0};

    // Initialize Queued Write Module.
    qwr_init.error_handler = nrf_qwr_error_handler;

    err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

    // Initialize NUS.
    memset(&nus_init, 0, sizeof(nus_init));

    nus_init.data_handler = nus_data_handler;

    err_code = ble_nus_init(&m_nus, &nus_init);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling an event from the Connection Parameters Module.
 *
```

```
 * @details This function will be called for all events in the Connection Parameters Module
 *          which are passed to the application.
 *
 * @note All this function does is to disconnect. This could have been done by simply setting
 *       the disconnect_on_fail config parameter, but instead we use the event handler
 *       mechanism to demonstrate its use.
 *
 * @param[in] p_evt Event received from the Connection Parameters Module.
 */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
    {
        err_code = sd_ble_gap_disconnect(m_conn_handle, BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
        APP_ERROR_CHECK(err_code);
    }
}


/**@brief Function for handling errors from the Connection Parameters module.
 *
 * @param[in] nrf_error Error code containing information about what went wrong.
 */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/**@brief Function for initializing the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;

    memset(&cp_init, 0, sizeof(cp_init));
```

```
    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay =
        FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay =
        NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count =
        MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail         = false;
    cp_init.evt_handler                = on_conn_params_evt;
    cp_init.error_handler              = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for putting the chip into sleep mode.
 *
 * @note This function will not return.
 */
static void sleep_mode_enter(void)
{
    uint32_t err_code = bsp_indication_set(BSP_INDICATE_IDLE);
    APP_ERROR_CHECK(err_code);

    // Prepare wakeup buttons.
    err_code = bsp_btn_ble_sleep_mode_prepare();
    APP_ERROR_CHECK(err_code);

    // Go to system-off mode (this function will not return; wakeup
        will cause a reset).
    //err_code = sd_power_system_off();
   // APP_ERROR_CHECK(err_code);
    err_code = sd_app_evt_wait();
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling advertising events.
 *
 * @details This function will be called for advertising events
     which are passed to the application.
 *
 * @param[in] ble_adv_evt Advertising event.
 */
```

```c
static void on_adv_evt(ble_adv_evt_t ble_adv_evt)
{
    uint32_t err_code;

    switch (ble_adv_evt)
    {
        case BLE_ADV_EVT_FAST:
            err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
            APP_ERROR_CHECK(err_code);
            break;
        case BLE_ADV_EVT_IDLE:
            sleep_mode_enter();
            break;
        default:
            break;
    }
}

/**@brief Function for handling BLE events.
 *
 * @param[in]  p_ble_evt  Bluetooth stack event.
 * @param[in]  p_context  Unused.
 */
static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
{
    ret_code_t err_code;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            NRF_LOG_INFO("Connected.");
            err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
            APP_ERROR_CHECK(err_code);
            m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, m_conn_handle);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GAP_EVT_DISCONNECTED:
            NRF_LOG_INFO("Disconnected, reason %d.",
                         p_ble_evt->evt.gap_evt.params.disconnected.reason);
            m_conn_handle = BLE_CONN_HANDLE_INVALID;
            break;

        case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
```

```
        {
            NRF_LOG_DEBUG("PHY update request.");
            ble_gap_phys_t const phys =
            {
                .rx_phys = BLE_GAP_PHY_AUTO,
                .tx_phys = BLE_GAP_PHY_AUTO,
            };
            err_code =
                sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.conn_handle,
                &phys);
            APP_ERROR_CHECK(err_code);
        } break;

        case BLE_GATTC_EVT_TIMEOUT:
            // Disconnect on GATT Client timeout event.
            NRF_LOG_DEBUG("GATT Client Timeout.");
            err_code =
                sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,
                                             BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GATTS_EVT_TIMEOUT:
            // Disconnect on GATT Server timeout event.
            NRF_LOG_DEBUG("GATT Server Timeout.");
            err_code =
                sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,
                                             BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
            NRF_LOG_DEBUG("BLE_GAP_EVT_SEC_PARAMS_REQUEST");
            break;

        case BLE_GAP_EVT_AUTH_KEY_REQUEST:
            NRF_LOG_INFO("BLE_GAP_EVT_AUTH_KEY_REQUEST");
            break;

        case BLE_GAP_EVT_LESC_DHKEY_REQUEST:
            NRF_LOG_INFO("BLE_GAP_EVT_LESC_DHKEY_REQUEST");
            break;

         case BLE_GAP_EVT_AUTH_STATUS:
             NRF_LOG_INFO("BLE_GAP_EVT_AUTH_STATUS: status=0x%x
                 bond=0x%x lv4: %d kdist_own:0x%x kdist_peer:0x%x",
                          p_ble_evt->evt.gap_evt.params.auth_status.auth_status,
```

```
                              p_ble_evt->evt.gap_evt.params.auth_status.bonded,
                              p_ble_evt->evt.gap_evt.params.auth_status.sm1_levels.lv4,
                              *((uint8_t
                                  *)&p_ble_evt->evt.gap_evt.params.auth_status.kdist_own),
                              *((uint8_t
                                  *)&p_ble_evt->evt.gap_evt.params.auth_status.kdist_peer));
            break;

        default:
            // No implementation needed.
            break;
    }
}


/**@brief Function for the SoftDevice initialization.
 *
 * @details This function initializes the SoftDevice and the BLE
     event interrupt.
 */
static void ble_stack_init(void)
{
    ret_code_t err_code;

    err_code = nrf_sdh_enable_request();
    APP_ERROR_CHECK(err_code);

    // Configure the BLE stack using the default settings.
    // Fetch the start address of the application RAM.
    uint32_t ram_start = 0;
    err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG,
        &ram_start);
    APP_ERROR_CHECK(err_code);

    // Enable BLE stack.
    err_code = nrf_sdh_ble_enable(&ram_start);
    APP_ERROR_CHECK(err_code);

    // Register a handler for BLE events.
    NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO,
        ble_evt_handler, NULL);
    // Register with the SoftDevice handler module for BLE events.


}


```

```
/**@brief Function for handling events from the GATT library. */
void gatt_evt_handler(nrf_ble_gatt_t * p_gatt, nrf_ble_gatt_evt_t
    const * p_evt)
{
    if ((m_conn_handle == p_evt->conn_handle) && (p_evt->evt_id ==
        NRF_BLE_GATT_EVT_ATT_MTU_UPDATED))
    {
        m_ble_nus_max_data_len = p_evt->params.att_mtu_effective -
            OPCODE_LENGTH - HANDLE_LENGTH;
        NRF_LOG_INFO("Data len is set to 0x%X(%d)",
            m_ble_nus_max_data_len, m_ble_nus_max_data_len);
    }
    NRF_LOG_DEBUG("ATT MTU exchange completed. central 0x%x
        peripheral 0x%x",
                  p_gatt->att_mtu_desired_central,
                  p_gatt->att_mtu_desired_periph);
}


/**@brief Function for initializing the GATT library. */
void gatt_init(void)
{
    ret_code_t err_code;

    err_code = nrf_ble_gatt_init(&m_gatt, gatt_evt_handler);
    APP_ERROR_CHECK(err_code);

    err_code = nrf_ble_gatt_att_mtu_periph_set(&m_gatt,
        NRF_SDH_BLE_GATT_MAX_MTU_SIZE);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling events from the BSP module.
 *
 * @param[in]  event  Event generated by button press.
 */
void bsp_event_handler(bsp_event_t event)
{
    uint32_t err_code;
    switch (event)
    {
        case BSP_EVENT_SLEEP:
            sleep_mode_enter();
            break;

        case BSP_EVENT_DISCONNECT:
```

```
            err_code = sd_ble_gap_disconnect(m_conn_handle, BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;

        case BSP_EVENT_WHITELIST_OFF:
            if (m_conn_handle == BLE_CONN_HANDLE_INVALID)
            {
                err_code = ble_advertising_restart_without_whitelist(&m_advertising);
                if (err_code != NRF_ERROR_INVALID_STATE)
                {
                    APP_ERROR_CHECK(err_code);
                }
            }
            break;

        default:
            break;
    }
}


/**@brief  Function for handling app_uart events.
 *
 * @details This function will receive a single character from the app_uart module and append it to
 *          a string. The string will be be sent over BLE when the last character received was a
 *          'new line' '\n' (hex 0x0A) or if the string has reached the maximum data length.
 */
/**@snippet [Handling the data received over UART] */
void uart_event_handle(app_uart_evt_t * p_event)
{
    static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
    static uint8_t index = 0;
    uint32_t      err_code;

    switch (p_event->evt_type)
    {
        case APP_UART_DATA_READY:
            UNUSED_VARIABLE(app_uart_get(&data_array[index]));
            index++;
```

```

            if ((data_array[index - 1] == '\n') ||
                (data_array[index - 1] == '\r') ||
                (index >= m_ble_nus_max_data_len))
            {
                if (index > 1)
                {
                    NRF_LOG_DEBUG("Ready to send data over BLE NUS");
                    NRF_LOG_HEXDUMP_DEBUG(data_array, index);

                    do
                    {
                        uint16_t length = (uint16_t)index;
                        err_code = ble_nus_data_send(&m_nus,
                            data_array, &length, m_conn_handle);
                        if ((err_code != NRF_ERROR_INVALID_STATE) &&
                            (err_code != NRF_ERROR_RESOURCES) &&
                            (err_code != NRF_ERROR_NOT_FOUND))
                        {
                            APP_ERROR_CHECK(err_code);
                        }
                    } while (err_code == NRF_ERROR_RESOURCES);
                }

                index = 0;
            }
            break;

        case APP_UART_COMMUNICATION_ERROR:
            APP_ERROR_HANDLER(p_event->data.error_communication);
            break;

        case APP_UART_FIFO_ERROR:
            APP_ERROR_HANDLER(p_event->data.error_code);
            break;

        default:
            break;
    }
}
/**@snippet [Handling the data received over UART] */


/**@brief Function for initializing the UART module.
 */
/**@snippet [UART Initialization] */
static void uart_init(void)
```

```c
{
    uint32_t                     err_code;
    app_uart_comm_params_t const comm_params =
    {
        .rx_pin_no    = RX_PIN_NUMBER,
        .tx_pin_no    = TX_PIN_NUMBER,
        .rts_pin_no   = RTS_PIN_NUMBER,
        .cts_pin_no   = CTS_PIN_NUMBER,
        .flow_control = APP_UART_FLOW_CONTROL_DISABLED,
        .use_parity   = false,
#if defined (UART_PRESENT)
        .baud_rate    = NRF_UART_BAUDRATE_115200
#else
        .baud_rate    = NRF_UARTE_BAUDRATE_115200
#endif
    };

    APP_UART_FIFO_INIT(&comm_params,
                       UART_RX_BUF_SIZE,
                       UART_TX_BUF_SIZE,
                       uart_event_handle,
                       APP_IRQ_PRIORITY_LOWEST,
                       err_code);
    APP_ERROR_CHECK(err_code);
}
/**@snippet [UART Initialization] */


static void peer_manager_init(void)
{
    ble_gap_sec_params_t sec_param;
    ret_code_t           err_code;

    err_code = pm_init();
    APP_ERROR_CHECK(err_code);

    memset(&sec_param, 0, sizeof(ble_gap_sec_params_t));

    // Security parameters to be used for all security procedures.
    sec_param.bond           = SEC_PARAM_BOND;
    sec_param.mitm           = SEC_PARAM_MITM;
    sec_param.lesc           = SEC_PARAM_LESC;
    sec_param.keypress       = SEC_PARAM_KEYPRESS;
    sec_param.io_caps        = SEC_PARAM_IO_CAPABILITIES;
    sec_param.oob            = SEC_PARAM_OOB;
    sec_param.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    sec_param.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
```

```
    sec_param.kdist_own.enc = 1;
    sec_param.kdist_own.id = 1;
    sec_param.kdist_peer.enc = 1;
    sec_param.kdist_peer.id = 1;

    err_code = pm_sec_params_set(&sec_param);
    APP_ERROR_CHECK(err_code);

    err_code = pm_register(pm_evt_handler);
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for initializing the Advertising functionality.
 */
static void advertising_init(void)
{
    uint32_t             err_code;
    ble_advertising_init_t init;

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

    init.advdata.name_type       = BLE_ADVDATA_FULL_NAME;
    init.advdata.include_appearance = false;
    init.advdata.flags           =
        BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE;

    init.srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) /
        sizeof(m_adv_uuids[0]);
    init.srdata.uuids_complete.p_uuids = m_adv_uuids;

    init.config.ble_adv_fast_enabled = true;
    init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;
    init.config.ble_adv_fast_timeout = APP_ADV_DURATION;
    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising,
        APP_BLE_CONN_CFG_TAG);
}


/**@brief Function for initializing buttons and leds.
 *
 * @param[out] p_erase_bonds Will be true if the clear bonding
     button was pressed to wake the application up.
```

```
 */
static void buttons_leds_init(bool * p_erase_bonds)
{
    bsp_event_t startup_event;

    uint32_t err_code = bsp_init(BSP_INIT_LEDS | BSP_INIT_BUTTONS,
        bsp_event_handler);
    APP_ERROR_CHECK(err_code);

    err_code = bsp_btn_ble_init(NULL, &startup_event);
    APP_ERROR_CHECK(err_code);

    *p_erase_bonds = (startup_event ==
        BSP_EVENT_CLEAR_BONDING_DATA);
}


/**@brief Function for initializing the nrf log module.
 */
static void log_init(void)
{
    ret_code_t err_code = NRF_LOG_INIT(NULL);
    APP_ERROR_CHECK(err_code);

    NRF_LOG_DEFAULT_BACKENDS_INIT();
}


/**@brief Function for initializing power management.
 */
static void power_management_init(void)
{
    ret_code_t err_code;
    err_code = nrf_pwr_mgmt_init();
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling the idle state (main loop).
 *
 * @details If there is no pending log operation, then sleep until
     next the next event occurs.
 */
static void idle_state_handle(void)
{

    UNUSED_RETURN_VALUE(NRF_LOG_PROCESS());
```

```
    nrf_pwr_mgmt_run();
}



void advertising_start(bool erase_bonds)
{
    if (erase_bonds == true)
    {
        delete_bonds();
        // Advertising is started by PM_EVT_PEERS_DELETE_SUCCEEDED
            event.
    }
    else
    {
        ret_code_t err_code;

        err_code = ble_advertising_start(&m_advertising,
            BLE_ADV_MODE_FAST);
        APP_ERROR_CHECK(err_code);
    }
}


void timer_handler(nrf_timer_event_t event_type, void* p_context)
{

}




void saadc_sampling_event_init(void)
{
    ret_code_t err_code;
    err_code = nrf_drv_ppi_init();
    APP_ERROR_CHECK(err_code);

    nrf_drv_timer_config_t timer_config =
        NRF_DRV_TIMER_DEFAULT_CONFIG;
    timer_config.frequency = NRF_TIMER_FREQ_31250Hz;
    err_code = nrf_drv_timer_init(&m_timer, &timer_config,
        timer_handler);
    APP_ERROR_CHECK(err_code);

    /* setup m_timer for compare event */
```

```
    uint32_t ticks =
        nrf_drv_timer_ms_to_ticks(&m_timer,SAADC_SAMPLE_RATE);
    nrf_drv_timer_extended_compare(&m_timer, NRF_TIMER_CC_CHANNEL0,
        ticks, NRF_TIMER_SHORT_COMPARE0_CLEAR_MASK, false);
    nrf_drv_timer_enable(&m_timer);

    uint32_t timer_compare_event_addr =
        nrf_drv_timer_compare_event_address_get(&m_timer,
        NRF_TIMER_CC_CHANNEL0);
    uint32_t saadc_sample_event_addr =
        nrf_drv_saadc_sample_task_get();

    /* setup ppi channel so that timer compare event is triggering
        sample task in SAADC */
    err_code = nrf_drv_ppi_channel_alloc(&m_ppi_channel);
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_assign(m_ppi_channel,
        timer_compare_event_addr, saadc_sample_event_addr);
    APP_ERROR_CHECK(err_code);
}


void saadc_sampling_event_enable(void)
{
    ret_code_t err_code = nrf_drv_ppi_channel_enable(m_ppi_channel);
    APP_ERROR_CHECK(err_code);
}


void saadc_callback(nrf_drv_saadc_evt_t const * p_event)
{
    if (p_event->type == NRF_DRV_SAADC_EVT_DONE)
    {
        ret_code_t err_code;
        uint16_t adc_value;
        uint8_t value[SAADC_SAMPLES_IN_BUFFER*2];
        uint16_t bytes_to_send;



        // set buffers
        err_code =
            nrf_drv_saadc_buffer_convert(p_event->data.done.p_buffer,
            SAADC_SAMPLES_IN_BUFFER);
        APP_ERROR_CHECK(err_code);

```

```
        // print samples on hardware UART and parse data for BLE transmission
        printf("ADC event number: %d\r\n",(int)m_adc_evt_counter);
        for (int i = 0; i < SAADC_SAMPLES_IN_BUFFER; i++)
        {
            printf("%d\r\n", p_event->data.done.p_buffer[i]);

            adc_value = p_event->data.done.p_buffer[i];
            value[i*2] = adc_value;
            value[(i*2)+1] = adc_value >> 8;
        }

        // Send data over BLE via NUS service. Makes sure not to send more than 20 bytes.
        if((SAADC_SAMPLES_IN_BUFFER*2) <= 4)
        {
            bytes_to_send = (SAADC_SAMPLES_IN_BUFFER*2);
        }
        else
        {
            bytes_to_send = 4;
        }
        err_code = ble_nus_data_send(&m_nus, value, &bytes_to_send, m_conn_handle);
        if ((err_code != NRF_ERROR_INVALID_STATE) && (err_code != NRF_ERROR_NOT_FOUND))
        {
            APP_ERROR_CHECK(err_code);
        }

        m_adc_evt_counter++;
    }
}


void saadc_init(void)
{
    ret_code_t err_code;

    nrf_drv_saadc_config_t saadc_config = NRF_DRV_SAADC_DEFAULT_CONFIG;
    saadc_config.resolution = NRF_SAADC_RESOLUTION_12BIT;
    saadc_config.oversample = NRF_SAADC_OVERSAMPLE_32X;

    nrf_saadc_channel_config_t channel_0_config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN4);
    channel_0_config.gain = NRF_SAADC_GAIN1_4;
```

```
    channel_0_config.burst = NRF_SAADC_BURST_DISABLED;
    channel_0_config.reference = NRF_SAADC_REFERENCE_INTERNAL;
 /*
    nrf_saadc_channel_config_t channel_1_config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN5);
    channel_1_config.gain = NRF_SAADC_GAIN1_4;
    channel_1_config.reference = NRF_SAADC_REFERENCE_VDD4;

    nrf_saadc_channel_config_t channel_2_config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN6);
    channel_2_config.gain = NRF_SAADC_GAIN1_4;
    channel_2_config.reference = NRF_SAADC_REFERENCE_VDD4;

    nrf_saadc_channel_config_t channel_3_config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN7);
    channel_3_config.gain = NRF_SAADC_GAIN1_4;
    channel_3_config.reference = NRF_SAADC_REFERENCE_VDD4;
 */
    err_code = nrf_drv_saadc_init(&saadc_config, saadc_callback);
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_saadc_channel_init(0, &channel_0_config);
    APP_ERROR_CHECK(err_code);
    /*
    err_code = nrf_drv_saadc_channel_init(1, &channel_1_config);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_saadc_channel_init(2, &channel_2_config);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_saadc_channel_init(3, &channel_3_config);
    APP_ERROR_CHECK(err_code);
*/
    err_code =
        nrf_drv_saadc_buffer_convert(m_buffer_pool[0],SAADC_SAMPLES_IN_BUFFER);
    APP_ERROR_CHECK(err_code);
    err_code =
        nrf_drv_saadc_buffer_convert(m_buffer_pool[1],SAADC_SAMPLES_IN_BUFFER);
    APP_ERROR_CHECK(err_code);
}

/**@brief Application main function.
 */



/**@brief Application main function.
 */
int main(void)
```

```
{
    bool erase_bonds;

    // Initialize.
    uart_init();
    log_init();
    timers_init();
    application_timers_start();
    buttons_leds_init(&erase_bonds);
    power_management_init();
    ble_stack_init();
    gap_params_init();
    gatt_init();
    services_init();
    advertising_init();
    conn_params_init();
    nrf_drv_saadc_calibrate_offset();
    saadc_sampling_event_init();
    saadc_sampling_event_enable();
    saadc_init();
    peer_manager_init();



    // Start execution.
    printf("\r\nUART started.\r\n");
    NRF_LOG_INFO("Debug logging for UART over RTT started.");
    //advertising_start();

    // Enter main loop.
    for (;;)
    {
        idle_state_handle();
    }
}


/**
 * @}
 */
```

Bibliography

[1] World Health Organization. "Spinal cord injury". In: (2013).

[2] Waleed Al Taweel and Raouf Seyam. "Neurogenic bladder in spinal cord injury patients". In: *Research and reports in urology* 7 (2015), p. 85.

[3] David Ginsberg. "Optimizing therapy and management of neurogenic bladder". In: *Am J Manag Care* 19.Suppl 10 (2013), pp. 197–204.

[4] Roger CL Feneley, Ian B Hopley, and Peter NT Wells. "Urinary catheters: history, current status, adverse events and research agenda". In: *Journal of medical engineering & technology* 39.8 (2015), pp. 459–470.

[5] Dennis G Maki and Paul A Tambyah. "Engineering out the risk for infection with urinary catheters." In: *Emerging infectious diseases* 7.2 (2001), p. 342.

[6] World Health Organization et al. *Antimicrobial resistance: global report on surveillance*. World Health Organization, 2014.

[7] A Tantin et al. "Implantable MICS-based wireless solution for bladder pressure monitoring". In: *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE. 2017, pp. 1–4.

[8] Ali Zaher et al. "Integrated electronic system for implantable sensory NFC tag". In: *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE. 2015, pp. 7119–7122.

[9] Chua-Chin Wang et al. "A mini-invasive long-term bladder urine pressure measurement ASIC and system". In: *IEEE Transactions on Biomedical Circuits and Systems* 2.1 (2008), pp. 44–49.

[10] Steve JA Majerus et al. "Wireless, ultra-low-power implantable sensor for chronic bladder pressure monitoring". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 8.2 (2012), pp. 1–13.

[11] Y. Zhong et al. "Development of an Implantable Wireless and Batteryless Bladder Pressure Monitor System for Lower Urinary Tract Dysfunction". In: *IEEE Journal of Translational Engineering in Health and Medicine* 8 (2020), pp. 1–7. DOI: 10.1109/JTEHM.2019.2943170.

[12] Fei Zhang et al. "Wireless energy transfer platform for medical sensors and implantable devices". In: *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2009, pp. 1045–1048.

[13] *ASIC or FPGA? Each solution has Advantages and Disadvantages*. https://www.swindonsilicon.com/asic-fpga-advantages-and-disadvantages.

[14] *Asic development*. https://zipcpu.com/blog/2017/10/13/fpga-v-asic.html. 2017.

[15] *The Basics of Bluetooth Low Energy (BLE)*. https://www.novelbits.io/tag/introduction/.

[16] Artem Dementyev et al. "Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario". In: *2013 IEEE International Wireless Symposium (IWS)*. IEEE. 2013, pp. 1–4.

[17] Mohammad Afaneh. *Intro to Bluetooth Low Energy: The Easiest Way to Learn BLE*. Novel Bits, 2018.

[18] Martin Woolley and S Schmidt. "Bluetooth 5 Go Faster. Go Further". In: *Bluetooth SIG* 1.1 (2019), pp. 1–25.

[19] Food, Drug Administration, et al. *FDA Informs Patients, Providers and Manufacturers About Potential Cybersecurity Vulnerabilities in Certain Medical Devices with Bluetooth Low Energy*. 2020.

[20] *nRF52840-DK*. https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-DK.

[21] *SEGGER Embedded Studio*. https://www.nordicsemi.com/Software-and-tools/Development-Tools/Segger-Embedded-Studio.

[22] *Is a single-chip SOC processor right for your embedded project)*. https://www.embedded.com/is-a-single-chip-soc-processor-right-for-your-embedded-project/.

[23] *What is a System on Chip*. https://anysilicon.com/what-is-a-system-on-chip-soc/.

[24] https://www.segger.com/products/development-tools/embedded-studio/.

[25] *Softdevices*. https://infocenter.nordicsemi.com/topic/struct_nrf52/struct/nrf52_softdevices.html.

[26] *Diffie-Hellman*. https://www.sciencedirect.com/topics/computer-science/diffie-hellman.

[27] SIG Bluetooth. "Bluetooth core specification version 4.0". In: *Specification of the Bluetooth System* 1 (2010), p. 7.

[28] *Applications and Types of PCBs for Medical Industry*. https://www.pcbcart.com/article/content/medical-pcbs.html.

[29] Inc Micro Systems Technologies. *Electronics Packaging Methods and Materials for Implantable Medical Devices*. 2016.

[30] Seung-Hee Ahn, Joonsoo Jeong, and Sung June Kim. "Emerging encapsulation technologies for long-term reliability of microfabricated implantable devices". In: *Micromachines* 10.8 (2019), p. 508.

[31] *INA333 Micro-Power (50A), Zerø-Drift, Rail-to-Rail Out Instrumentation Amplifier (Rev. C)*. https://www.ti.com/document-viewer/INA333/datasheet#.

[32] *Ceramic Chip Antennas vs. PCB Trace Antennas: A Comparison*. https://www.mouser.co.id/pdfDocs/ceramicchipantennasvspcbtraceantennasacomparison pdf.

[33] *nRF52840 product specification*. https://infocenter.nordicsemi.com/topic/struct_nrf52/struct/nrf52840.html. June 2020.

[34] Digi-Key's European Editors. *Leveraging Near Field Communication (NFC) to Connect with BLE Smart Sensors*. 2017.

[35] R. V. Taalla et al. "A Review on Miniaturized Ultrasonic Wireless Power Transfer to Implantable Medical Devices". In: *IEEE Access* 7 (2019), pp. 2092–2106. DOI: 10.1109/ACCESS.2018.2886780.

[36] S. Suzuki, T. Katane, and O. Saito. "Fundamental study of an electric power transmission system for implanted medical devices using magnetic and ultrasonic energy". In: *Journal of Artificial Organs* (2003).

[37] *PCB Design Guidelines For Reduced EMI*. https://www.ti.com/lit/an/szza009. 1999.