Permanent Magnet Display System

By

Xenia Leon Rodriguez

Advisor Bridget Benson

Senior Project EE460

Completion Date June 2015

Electrical Engineering Department

California Polytechnic State University, San Luis Obispo

Table of Contents

List of Tables

Table of Figures

Abstract

Pacemakers can malfunction when exposed to high magnetic fields. This becomes especially problematic for patients with pacemakers who need MRI scans as MRI machines can subject the pacemaker to damaging magnetic fields. St.Jude Medical, one of the leading biotech companies, is working to make their pacemakers MRI-safe. In order to test their MRI-safe pacemaker designs, they need an automated test apparatus that will collect and report the magnetic field the pacemaker is being exposed to. This report describes the design of such a test system that reads the Tesla intensity the pacemaker is exposed to and wirelessly transmits the data outside the testing room where the data can be viewed and evaluated.

Acknowledgements

I would like to thank Dr.Benson as she was a great mentor throughout the project. She gave me the opportunity to complete the project within two quarters in order for me to graduate on time. I would also like to thank St.Jude Medical for providing me the opportunity to do a co-op with them and sponsorship my senior project. Also for all the technical support and advice they provided me.

Chapter 1: Introduction

An MRI (magnetic resonance imaging) machine scans patients to see the inside tissues without the need of surgery. The MRI machine uses a magnetic field, radio frequency, permanent magnet and computer to produce detailed pictures of the organs and other internal body structures. These images are further analyzed by doctors to determine any medical issues the patients might have such as certain types of heart problems, abnormalities of the brain and spinal cord and tumors [13].

A pacemaker's main function is to regulate the human heartbeat when its own regulating mechanism breaks down. A pacemaker's main components are a battery, lead wires, and electronics. The lead wires are one of the components which are negatively affected by MRI. Studies conducted by the department of Technology and Health have shown the temperature of the lead significantly increases during an MRI-scan. In-vitro temperature measurements of the leads which were exposed to a 1.5T MRI scanner showed a temperature increase from 2.1C to 15C due to the RF field [12]. The radiofrequency not only increases the temperature but also causes for voltage to be induced in a pacemaker's leads which may stimulate the heart and affect the pacing therapy.

St. Jude Medical is currently working on making their pacemakers MRI safe as many patients who have a pacemakers need an MRI scan. The standard MRI scans have a field intensity of 3T. St.Jude Medical has a permanent magnet, RF field and gradient field testing room. The pacemakers are exposed to each of the fields and the data is analyzed to motivate design decisions to make their pacemakers MRI-safe.

In order to conduct their testing in the permanent magnet room they need an automated test setup to characterize the magnetic field the pacemaker is being exposed. This report describes the design of such test system. The test apparatus reads the Tesla intensity the pacemaker is exposed to and wirelessly transmits the data outside the testing room where the data can be viewed and evaluated.

Chapter 2: Requirement and Specifications

Customer Needs Assessment

The primary customer of the permanent magnet display system is St. Jude Medical's hardware development team. They need a system that will measure the Tesla intensity the pacemaker is exposed to inside the testing room and report the measured values outside of the testing room (so that no human will be exposed to the magnetic field within the testing room). Also the sensor that measures the magnetic exposure needs a small diameter due to the area limitation of the testing apparatus. These are the main customer needs which form the base to develop the requirement and specifications shown in **Error! Reference source not found.**.

Requirement and Specifications

Transmitting permanent magnet data wirelessly is the project's primary requirement. The project must use Nordic nRF Bluetooth development kit as St. Jude Medical requested this platform. The prototype must also have a sensor small enough that fits in the permanent magnet testing machine of one cubic inch.

Due to the limited area of the permanent magnet machine, various specifications were set as shown on **Error! Reference source not found.**. St. Jude medical needs a magnetic output accuracy of ±0.5mT. These and other specifications and their respective justifications are shown in **Error! Reference source not found.**.

Table I Permanent magnet display system specifications

Marketing Requirements	Engineering Specifications	Justification
1,2	The system must include Nordic nRFgo Starter Kit Bluetooth low energy to transmit the data wirelessly.	Nordic nRFgo is the wireless communication system that St. Jude requested to be used for the system.
2,3,4,7	The system will have a on and off switch to increase the 9V battery life for more than 2 years.	The customer wants the system to last more than 2 years.
5	The magnetic field sensor must range from 0T-3T.	The subjects that are tested get exposed from 0T to 3T.
4	The dimensions of the prototype should not exceed 8"x6"x4"	The system must be able to fit next to the mechanical arm which has a limited space
5	Must be able to measure the permanent magnet output with an accuracy of ±0.5mT	Subjects that are tested are critical and need accurate exposure measurements to study the results.
6,1	The magnetic field at which the subjects are being exposed to will be displayed on a computer outside the permanent magnet testing room.	The permanent magnet could damage electronic devices if exposed closely to it. For safety reasons, it is important to limit the contact of operators and the permanent magnet.
7	The magnetic field sensor should not exceed the dimensions of 1"X 1" X 1".	Fixtures that hold the test subjects require these dimensions.
8	System transmits measurable data every 3 seconds	The customer needs a system that will update at least every 3 seconds

Marketing Requirements

1. The system must send the magnetic field data wirelessly
2. System must be powered by battery
3. System should not interfere with the mechanical arm
4. System needs to be compact
5. The measure of the permanent magnet needs to be accurate
6. The measurements need to be displayed outside the testing room
7. System must not interfere with the testing devices
8. System updates data regularly

Chapter 3: Functional Decomposition

Level 0 Block Diagram

Figure 1 shows the top level block diagram of the project. The permanent magnet sensor is battery powered and has a control switch. The voltage output of the permanent magnet is sent outside the room wirelessly to a tablet. Table II describes the module inputs, outputs and functionality. The Level one block diagram in shows a more detail design of the Figure 1 system.

Block Diagram

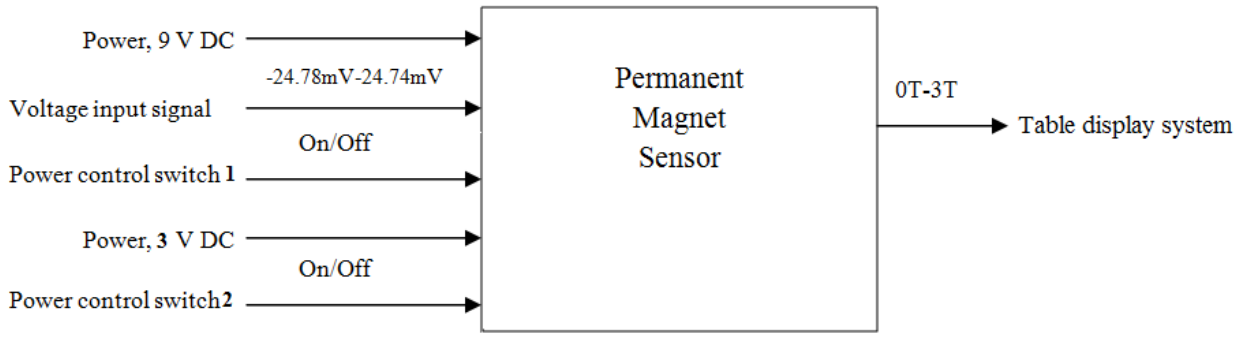


Figure 1: Level 0 Permanent Magnet Sensor

Table II: Level 0 Project Functionality

Module	Permanent Magnet Sensor
Inputs	-Permanent magnet voltage input: -24.78mV— 24.74mV -Power control switch 1: On and Off -Power control switch 2: On and Off -Power: 9 V DC -Power: 3 V DC
Outputs	-Permanent magnet output 0T-3T (Tesla)
Functionality	The permanent magnet voltage is sent to a tablet via Bluetooth and displayed on a tablet.

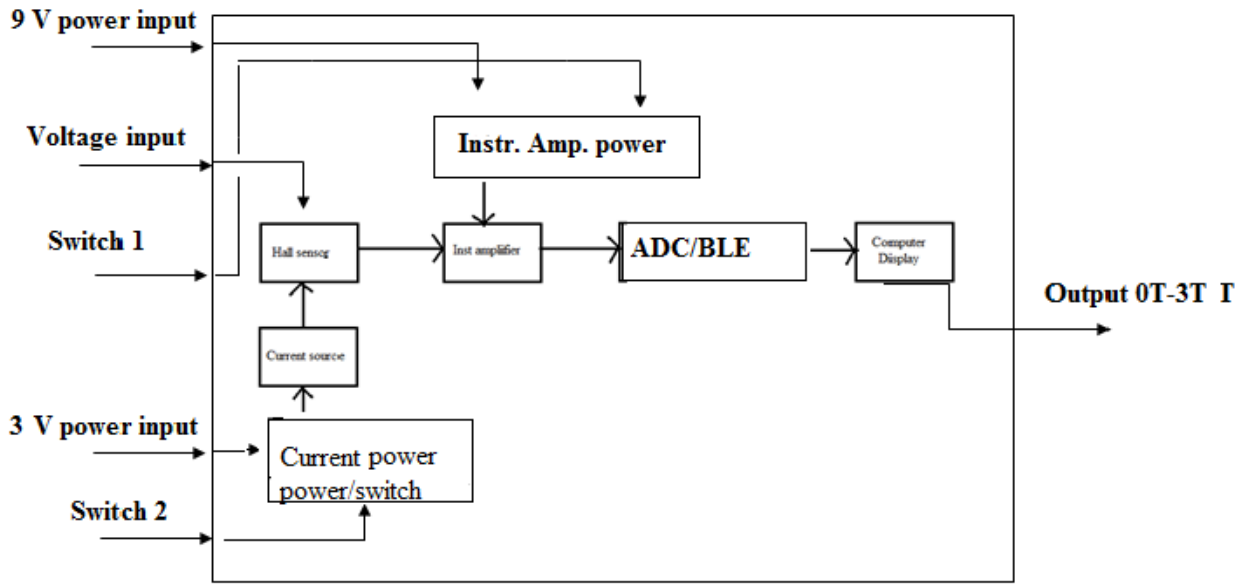


Figure 2 Level 1 permanent magnet sensor

Module	Current Power/Switch
Inputs	-On/off switch -3V battery
Outputs	- 0-2V
Functionality	Provides power to the current source

Module	Current Source
Inputs	-100 mA power supply
Outputs	-constant 100 mA
Functionality	Provides power to the current source

Project Planning

The project planning was spread out between winter 2015 and spring 2015. Figure 3 shows the winter gantt chart which consisted of the first planning phase and the first and second design of the project. Figure 4 shows the gantt chart for spring quarter which consisted of writing the report, alpha demo and completing the final project.

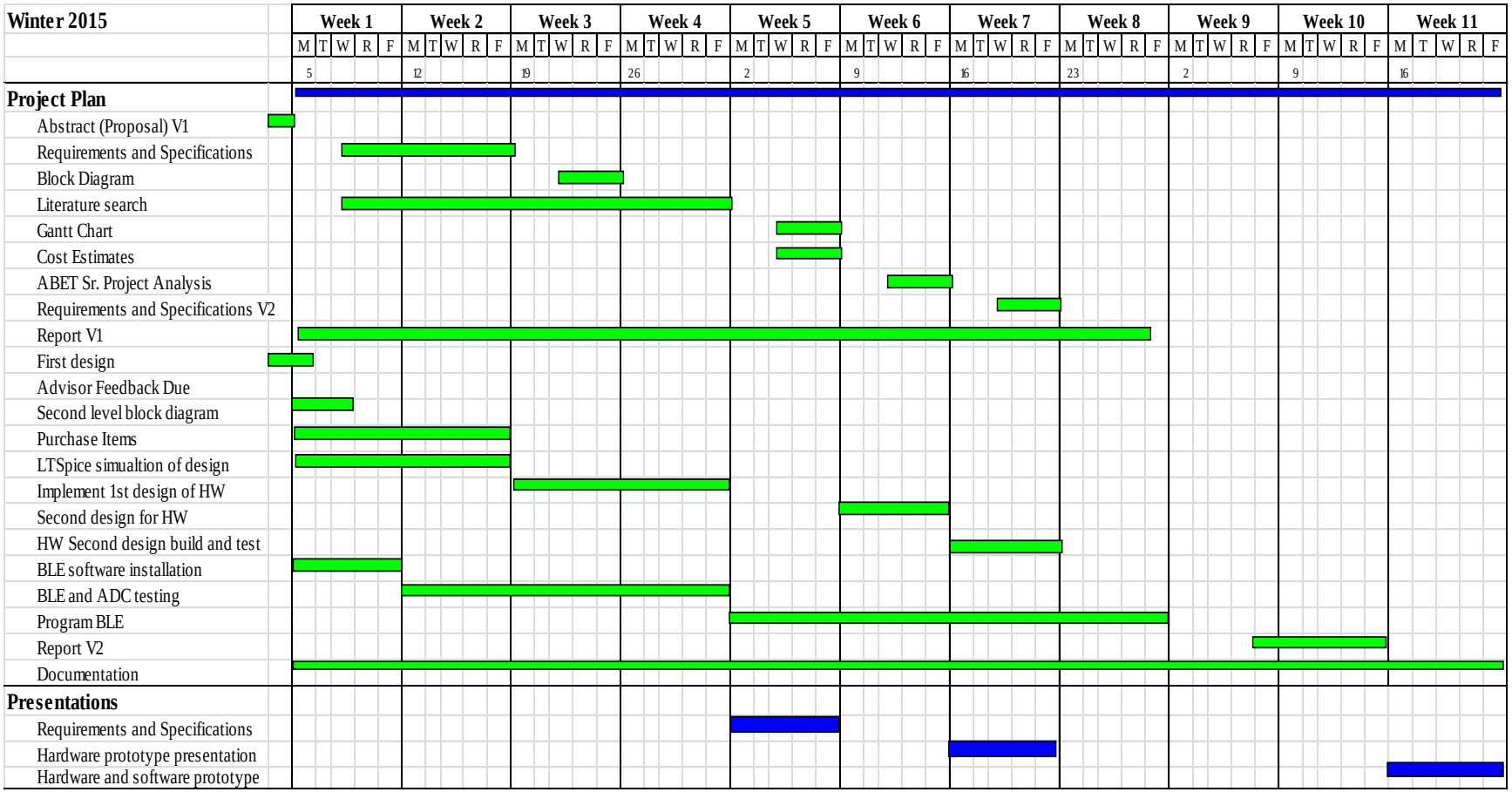


Figure 3 Winter Gantt Chart

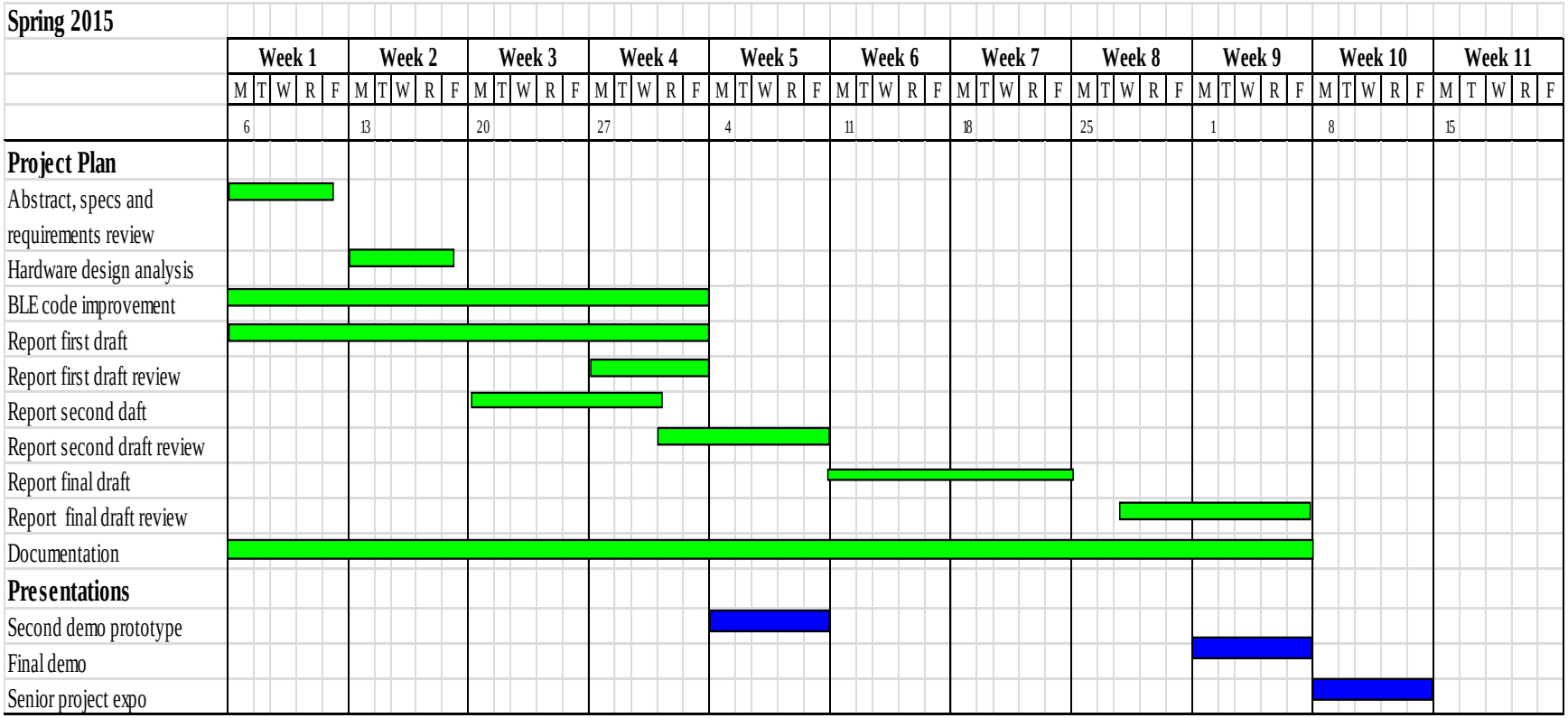


Figure 4 Spring Gantt Chart

Design

System Architecture

The system is compose of two major part which is hardware and software. The hardware part main component is the hall sensor which is power with a constant current source and the output is amplified with an instrumentation amplifier. The software part is compose of Nordic development kit (nRF51822) which includes the a mother board and an analog to digital converter. Also the Nordic starter kit (nRF6700) which includes the Bluetooth Low Energy (BLE). The nRF51822 motherboard analog to digital converter takes in the output of the instrumentation amplifier. The value then gets transmitted using the nRF6700 BLE.

Hardware

The hardware architecture of the system is shown in Figure 5 and the component list is shown in Table III. The main component of the system is the hall sensor which is power with a constant current source that is provided by the single resistor low dropout regulator. A 3V battery powers the low dropout regulator.

Table III System component list

Component	Part Number	Quantity	Value
Hall sensor	(HGT-3010)	1	N/A
Battery	N/A	9	3V , 9V
Single resistor low dropout regulator	LT3080	1	N/A
High speed FET-Input instrumentation amplifier	INA111	1	N/A
Dual output DC/DC converter	NMV0505SAC	1	N/A
Terminal adjustable regulator	LM317	1	N/A
Resistor	N/A	3	20kΩ, 20kΩ, 240kΩ,
Capacitor	N/A	2	1uF,4.7uF
Nordic nRFgo starter kit	nRF6700	1	N/A
Nordic Development kit	nRF51822	1	N/A

The output of the hall sensor is voltage that is amplified with an instrumentation amplifier. The output of the instrumentation amplifier is the input to the nRF51822 Bluetooth board. The instrumentation amplifier needs a positive and negative source. The DC/DC converter outputs the positive and negative 5V that the instrumentation amplifier needs. The DC/DC is power with a 5V that was steeped down from a 9V battery using an adjustable regulator.

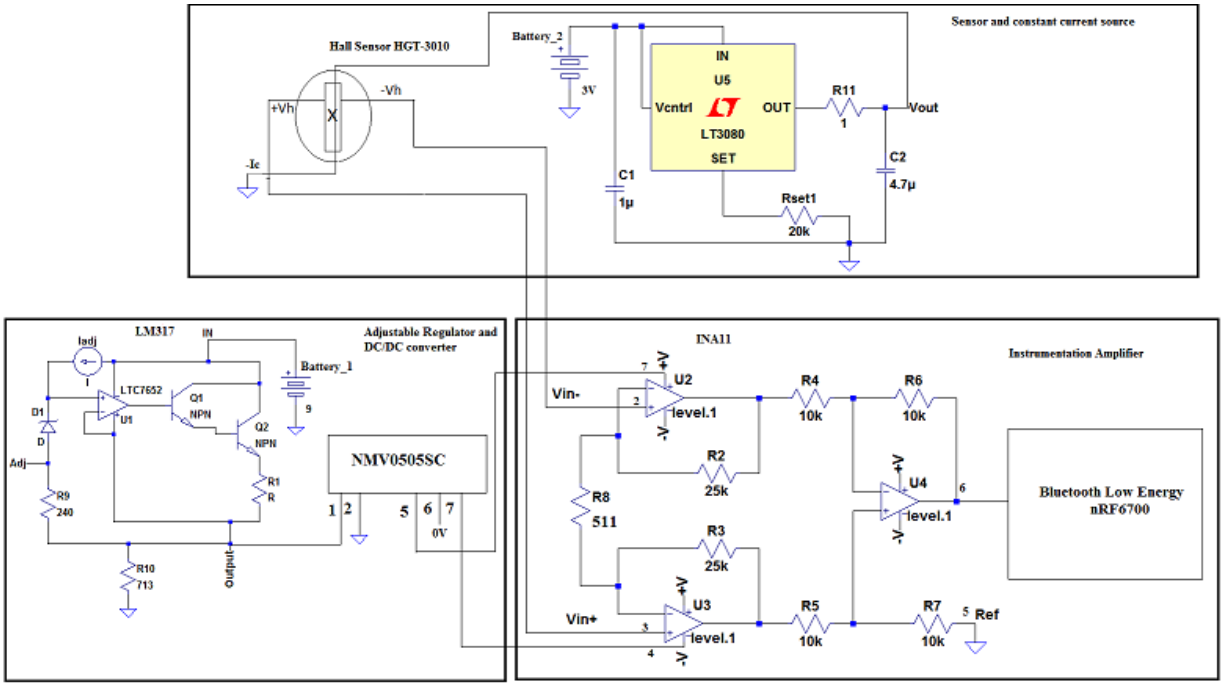


Figure 5: Hardware design

The sensor was tested using St.Jude Medical superconducting magnet as shown in Figure 6 . The four wires of the hall sensor was first solder to testing points and placed on a plastic stick. A constant current was provided using a Keithlye SMU. The superconducting magnet was first set to output .5T and the hall sensor was placed inside the middle of the superconducting magnet . The Tesla field tested and output expected and actual is shown in Table IV. The table shows that the hall sensor output match the one of the datasheet. The sensor resistance was also measured which was 2Ω.

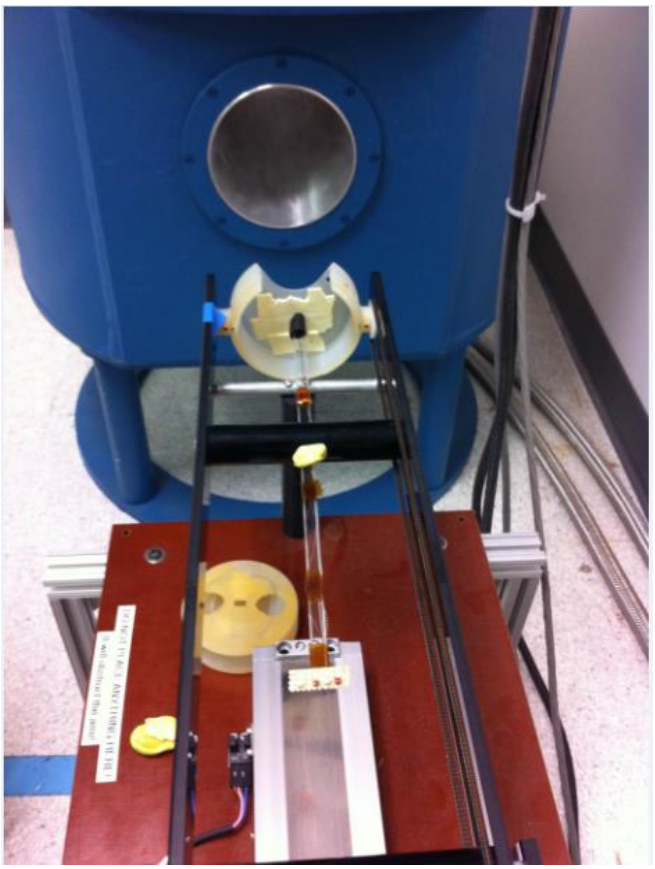


Figure 6: Hall sensor setup for testing

Table IV: Hall sensor data output

Superconducting field intensity (Testla)	Datasheet output (mV)	Measured output (mV)
.2	1.3358	1.346
1	6.7209	6.710
1.5	10.1113	10.10
2	13.5174	13.523
2.5	16.9391	16.947
3	20.3759	20.355

To provide a constant current of 100mA to the hall sensor a single resistor low dropout regulator (LT3080) was used. The regulator was tested using the setup as shown in Figure 8. The typology of the circuit (Figure 7) was modified to ensure that once it reached 3V it would maintain a constant current.

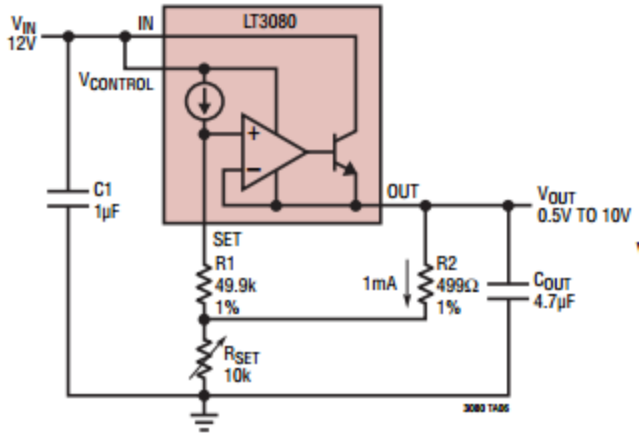


Figure 7 LT3080 Original Typology

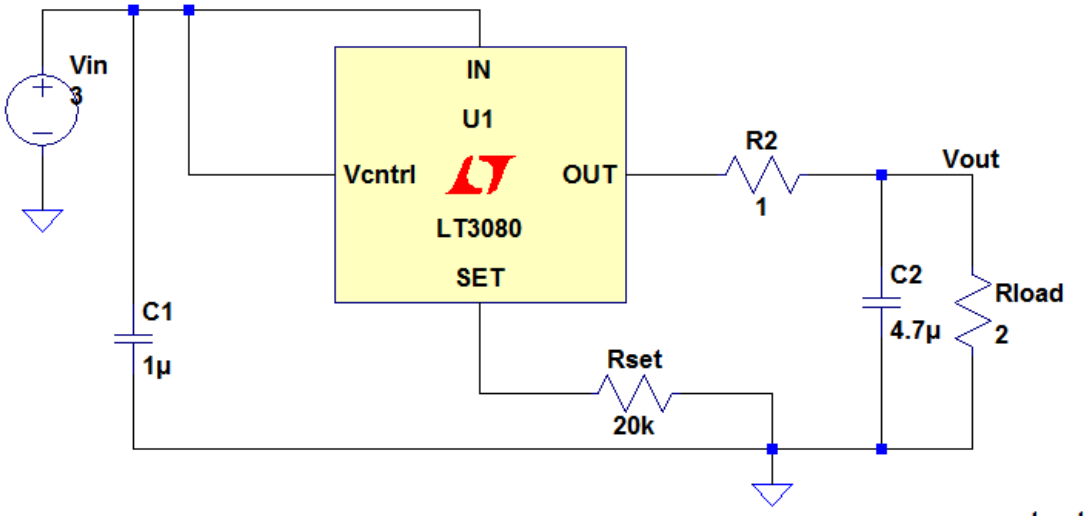


Figure 8 Sensor testing setup

The voltage input was vary from 0.6V to 3V with a load of 2Ω. The output was calculated and at 3V a steady 111mA was reached as shown in Table V and Figure 9. This was a desired value as the sensor needs a constant current of 100mA (maximum of 300mA).

Table V Regulator sensor data

Vin	Vout-measured (V)	Load (Ω)	I calculated (mA)
0.6	0	2	0
0.8	0	2	0

Vin	Vout-measured (V)	Load (Ω)	I calculated (mA)
1	0	2	0
1.2	0.03	2	15
1.5	0.208	2	104
1.7	0.222	2	111
2	0.222	2	111
2.5	0.222	2	111
2.7	0.222	2	111
3	0.222	2	111

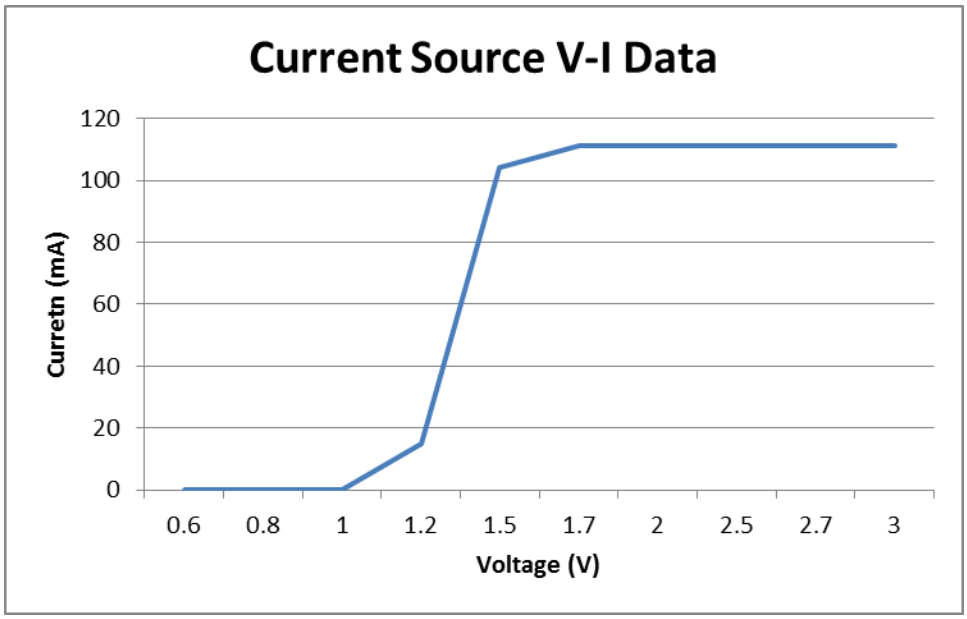


Figure 9 Current Source output

After the current source was establish the instrumentation amplifier with the DC/DC converter with the voltage regulator was tested as shown in Figure 10. The gain of the amplifier was set to 50 by selecting the resistor value of 511Ω between node 1 and 8 of the INA111.

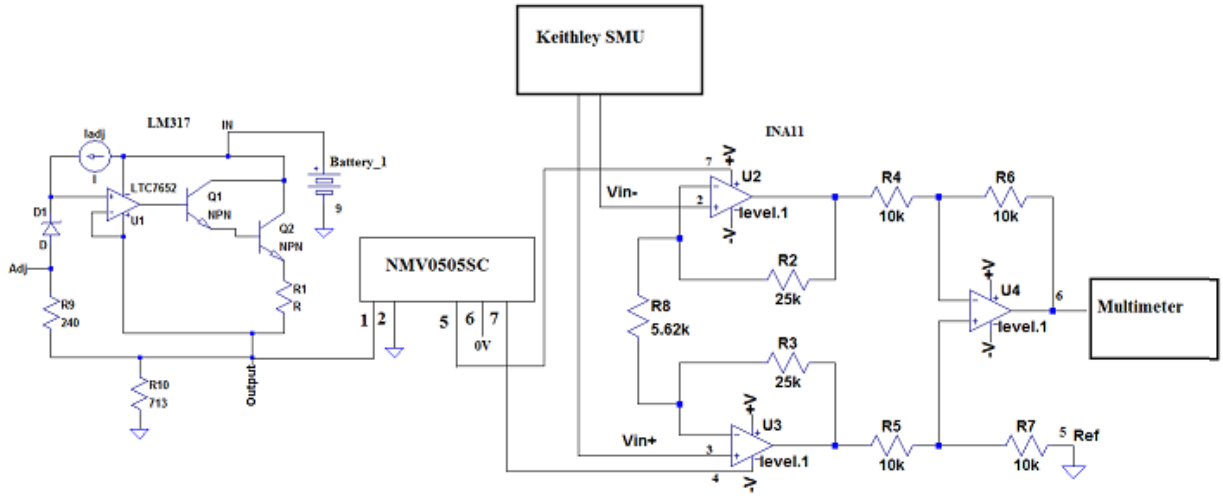


Figure 10 Instrumentation amplifier, DC/DC and voltage regulator test setup

The system was tested varying the voltage input and reading the voltage output and comparing it to the calculated value as shown in Table VI. The gain was set using equation 1 where R_G sets the gain and is connected between pin 1 and 8. The output was as expected which is shown in Table VI.

$$G = 1 + \frac{50k}{R_G} \qquad (1)$$

Table VI Instrumentation amplifier data

V+	V-	Theory Inst Amp output	Inst Amp output Measured V
0	0	0	32
0.001	-0.001	0.2	0.235
0.002	-0.002	0.4	0.432
0.003	-0.003	0.6	0.629
0.004	-0.004	0.8	0.826
0.005	-0.005	1	1.025
0.006	-0.006	1.2	1.222
0.007	-0.007	1.4	1.419
0.008	-0.008	1.6	1.616
0.009	-0.009	1.8	1.814
0.01	-0.01	2	2.011
0.011	-0.011	2.2	2.208

The next step was to interface the whole system which was done by soldering all the components together as shown in **Error! Reference source not found.**.

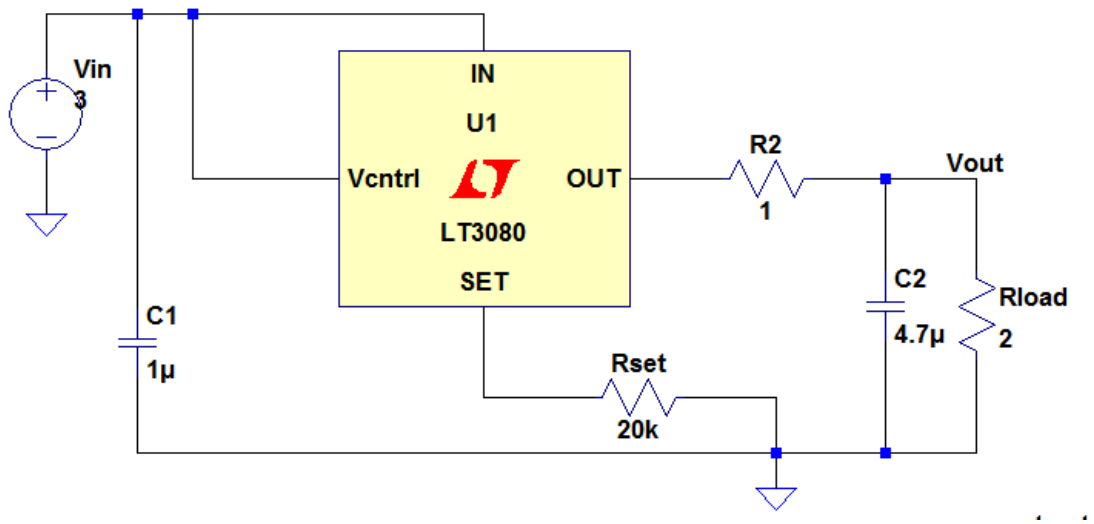


Figure 11 Interface of all components

The system was tested by using a small magnet which by using a Gauss meter the magnetic field was tested and the voltage output of it was 1.62mV. The system was tested at various points which is shown in Table VII. The final output of the instrumentation amplifier was 117mV which is off from 162mV the theoretical value. This may be due to current lost because of the solder joints.

Table VII Hardware system testing points

Setup	Theory Value (mV)	Output (mV)
Sensor output	1.62	1.62
Power supply, sensor, inst. Amplifier	162	172
All components sadder - switch, DC/DC, voltage regulator, current source, battery power	162	117

Software

The second part of the project was to program the Bluetooth low energy to transmit the data via air. Nordic nRF6700 starter kit and nRF51822 development kit was used to transmit the data. The starter kit provides the nRFgo Motherboard(nRF6310). The development kits includes the antenna module which also has a 10 bit analog to digital converter and also a USB dongle for debussing purpose. Figure 13 shows the setup for the Nordic Bluetooth low energy. The example

of the Heart Rate service specifically the battery information service update was modified to transmit the voltage via air. The example was used as the main frame for the project because it provided all the necessary setups which are required by the host and controller as shown in Figure 12. The only portion that needed to be change was the main to ensure the right data was send out.

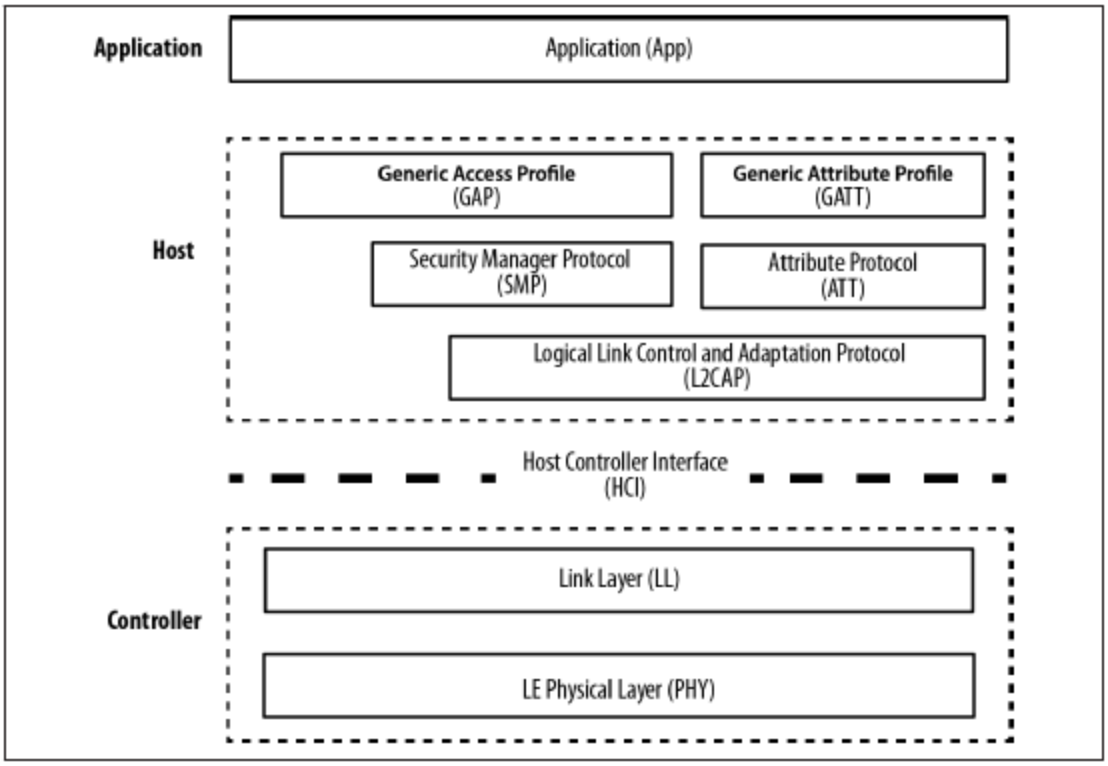


Figure 12 BLE overview

The first step was to setup nRFgo Mother board which was program using the nRFgo studio (Nordic provided all the software installations necessary). The code was written and compile using Keil uVision5. The dongle was used for debugging purpose. The overall setup is shown in Figure 13.

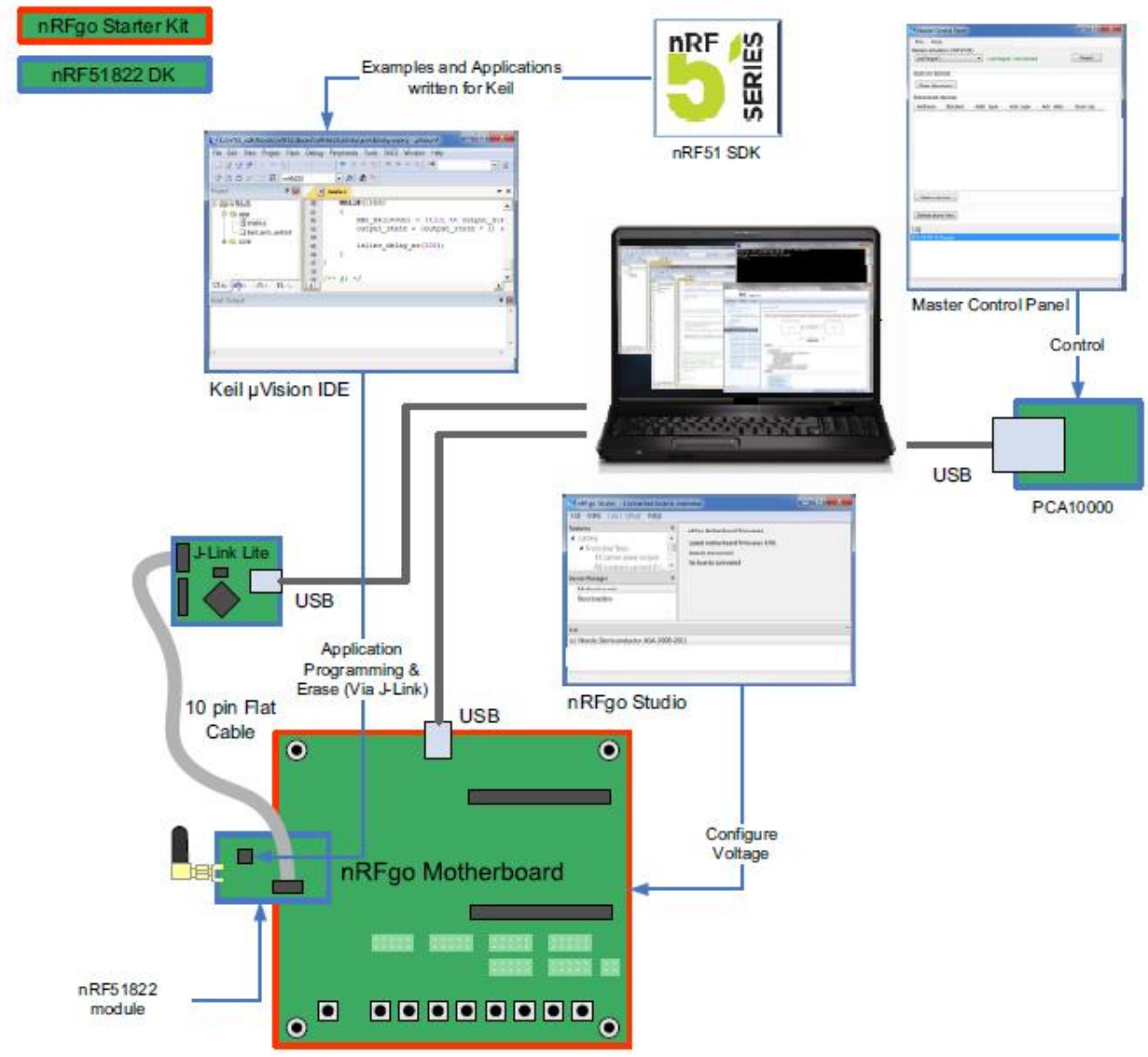


Figure 13 Nordic Board setup

The first step was to modified the nRF51822 to ensure the ADC could be used. The analog input (AIN2) was used as the analog input. The resistor attached to the was removed in order to reduce noise from the motherboard.

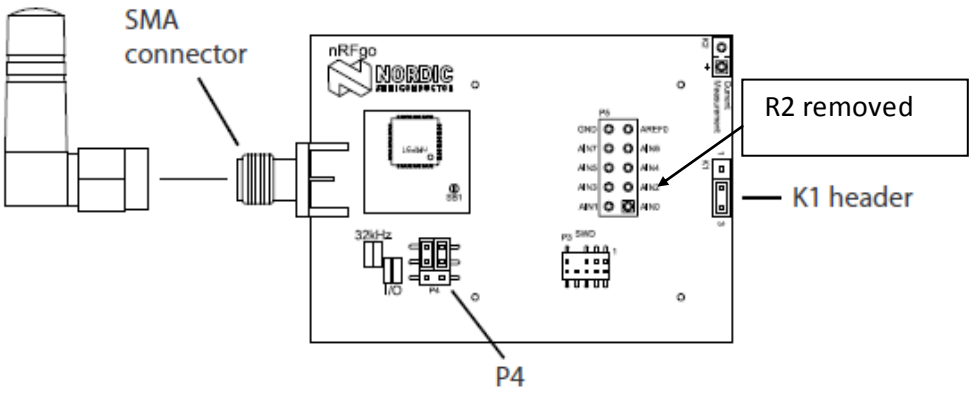
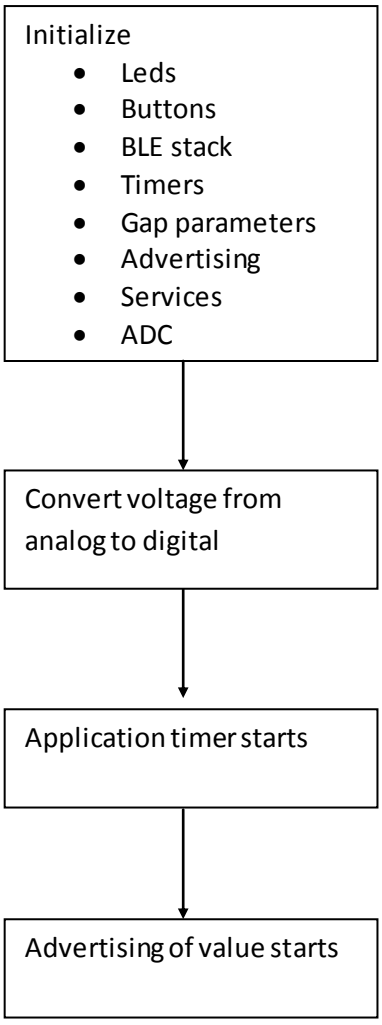


Figure 14 nRFgo51822 modified

The first step was to implement the ADC sampling. An additional application was setup to trigger the ADC sampling which is performed 10 times per second. The ADC is sampled with a 10 bit resolution, samples from analog input pin 2 and uses the internal 1.2VBG fixed references voltage as reference. One third prescaling is setup which enables the input voltage range of 0-3.6V. The code below are the two main code sections that enables the ADC. The figure below shows the software flow diagram.



To test the system the LEDs (of the motherboard) were programmed to when the sample is taken pin 10 (led2) is toggled and when the sample is ready pin 11 (led 3) is toggled and the sampling is output on port 2 (pins 16-23, LEDs turn on according to voltage input).

Once the ADC was confirm to be working the next step was to package the data and send it out. The code below shows the modified code to package and send out the ADC data.

The last step of the code was to send the data using the battery level update function. In the main code for the static void battery_level_update was modified. This functions updates the current value where the battery_level takes in the ADC value.

To test the system the BLE scanner application was used. The application scans for any BLE and connects to it. The only thing that was not change was the output format of the data. The original code outputs the data in percentage. This was kept the same, when a .1 voltage is the input to the system the output on the application of the phone reads 1 %.

Testing

Each component of the system was tested individually according to the specifications as shown in Table VIII. Once the certain requirement/specification is met, it is added to the final system and again, tested. The following table shows what was tested and their results.

Table VIII Testing parameters and results

Requirement/Specification	Test Used	Passed?
Test the hall sensor	• Connect constant current source and digital multimeter • Use a permanent magnet ranging from 0T-3T	Yes
Test current source	• Vary voltage until 100mA have been reach	Yes
Test voltage regulator, DC/DC and instrumentation amplifier	• Input a 9V and decrease the voltage to 5V • Apply various voltage to the amplifier	Yes
Test the instrumentation amplifier, current source and hall sensor , DC/DC, battery and	• Apply a magnet to the sensor and read the output voltage	

voltage regulator solder together		Yes
Test the ADC of the nRFgo 51822	• Apply a voltage to the ADC and vary the voltage and observe the LEDs change on the Motherboard	Yes
Test entire system (hardware interface with the nRFgo51822 and the motherboard together)	• Test the DAC code from the previous lab • Connect the DAC with the system	Yes

Conclusion

This project was perform to provide St.Jude Medical with a system that would allow them to determine the Testla value the pacemakers are exposed to. Through this project a better understanding of BLE has been obtain. Many hours were spend understanding the software system as is compose of many different architectures. The main thing learned from the project was to be able to understand a big system such as the BLE but being able to manipulate the main in order to obtain the desired values.

References

[1] K. Townsend, C. Cufi, Akiba, R. Davidson. *Getting Started with Bluetooth Low Energy*. Sebastopol , California: O'Reilly Media, 2014, pp. 15-95.

[2] Heydon, Robin. *Bluetooth Low Energy*. Upper Saddle River, NJ: Prentice Hall, 2012.

[3] Linear Technology, "Adjustable 1.1A Single Resistor Low Dropout Regulator," LT3080, 2007. http://www.linear.com/product/LT3080 January 5,2015.

[4] J. Decuir "Introducing Bluetooth Smart: Part 1: A look at both classic and new technologies", Consumer Electronics Maganize, Vol.3, no.1, pp.12-18, January 2014 [Online].Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6685914&queryText%3DBluetooth+Low+Energy+Technology [Accessed January 23, 2015].

[5] P. Pfister and Y.Perriard. "Slotless Permanent-Magnet Machines: General Analytical Magnetic Field Calculation", Magnetics, Vol.47, no.6, pp.1739-1752, January 2011 [Online].Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5713841&queryText%3Dpermanent+magnet [Accessed January 23, 2015].

[6] J. Decuir. "Introducing Bluetooth Smart: Part II: Applications and updates", Consumer Electronics Magazine, Vol.3, no.2, pp.25-29, April 2014 [Online].Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6776529 [Accessed January 23, 2015].

[7] Texas Instrument, "High Speed FET-Input INSTRUMENTATION AMPLIFIER" INA111, 2013. http://www.ti.com/lit/ds/symlink/ina111.pdf January 5,2015.

[8] Lake Shore Cryotronics, "Hall Sensors Magnetic Field Sensors" HGA-3010, 2014. http://www.lakeshore.com/products/hall-magnetic-sensors/pages/Specifications.aspx , January 5,2015.

[9] L. Kenneth and Kantor "Magnetic suspension transducer," USPTO Patent Full-Text and Image Database, January 27,2015. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=7&f=G&l=50&co1=AND&d=PTXT&s1=%22permanent+magnet%22&OS=%22permanent+magnet%22&RS=%22permanent+magnet%22 . [Accessed: Feb. 2, 2015].

[10] Murata Power Solutions, "NMV 5V,12V & 15V Series," NMV0509SC, 2012. http://www.mouser.com/ds/2/281/kdc_nmv-10836.pdf January 5,2015.

Appendix A. Senior Project Analysis

Project Title: Permanent Magnet Display System
Student's Name: Xenia Leon Rodriguez
Advisor's Name: Benson Bridget **Initial:** **Date:**

Summary of Functional Requirements

Pacemakers can malfunction when exposed to high magnetic fields. This becomes especially problematic for patients with pacemakers who need MRI scans as MRI machines can subject the pacemaker to damaging magnetic fields. St.Jude Medical, one of the leading biotech companies, is working to make their pacemakers MRI-safe. In order to test their MRI-safe pacemaker designs, they need an automated test apparatus that will collect and report the magnetic field the pacemaker is being exposed to. This report describes the design of such a test system that reads the Tesla intensity the pacemaker is exposed to and wirelessly transmits the data outside the testing room where the data can be viewed and evaluated.

Primary Constraints

The primary constraint involves sending the data from the Bluetooth low energy (BLE) of the permanent magnet , the tablet display and ensuring accuracy.

Economics

Human Capital

This project requires approximately 180 man hours of one person for the project to be completed. The project will be divided in two portions, the hardware and software. The hardware part requires time to develop a design and to study the datasheets. The hardware phase includes individual testing of the parts as well as the integration design testing. The second part of the project consist of the development of the Bluetooth low energy programming. This requires time to do some readings on Bluetooth and fully understand how it works. Over this time several code examples are studied. The project cost estimate for each component is shown on Table IX.

Table IX: Project component cost estimate

Description	Manufacture	Manufacture Part Number	Quantity	Cost
Instrumentational Amplifier	Texas Instruments	INA111BP-ND	2	33.34
DC-DC converter	Murata Power Solution	NMV0505SC	2	20.30
DC-DC converter	Murata Power Solution	NMV0509SC	2	20.30
BLE Development Kit	Nordic Semiconductor	nRF51822	1	114.14
BLE Starter kit	Nordic Semiconductor	nRF6700	1	443.99
Axial Hall Effect Sensor	Lakeshore Cryotronics	HGA-3010	3	1149.00
9V battery	Panasonic	6LR61XWA/1SB	1	2.40
AAA Battery	Panasonic	LR03XWA/B	3	1.05
Labor Cost	N/A	N/A	N/A	1440.00
			Total Cost	3224.52

Manufactured Capital

The manufacture capital is provided by St.Jude Medical sponsorship.

Natural Capital

Relatively none.

Timing

The project requires time for the hardware development portion which includes design, testing and colleting data. The second major portion of the project is software which includes researching existing code, building code and debugging. Figure 15 and Figure 16 shown the Gantt chart for the project.

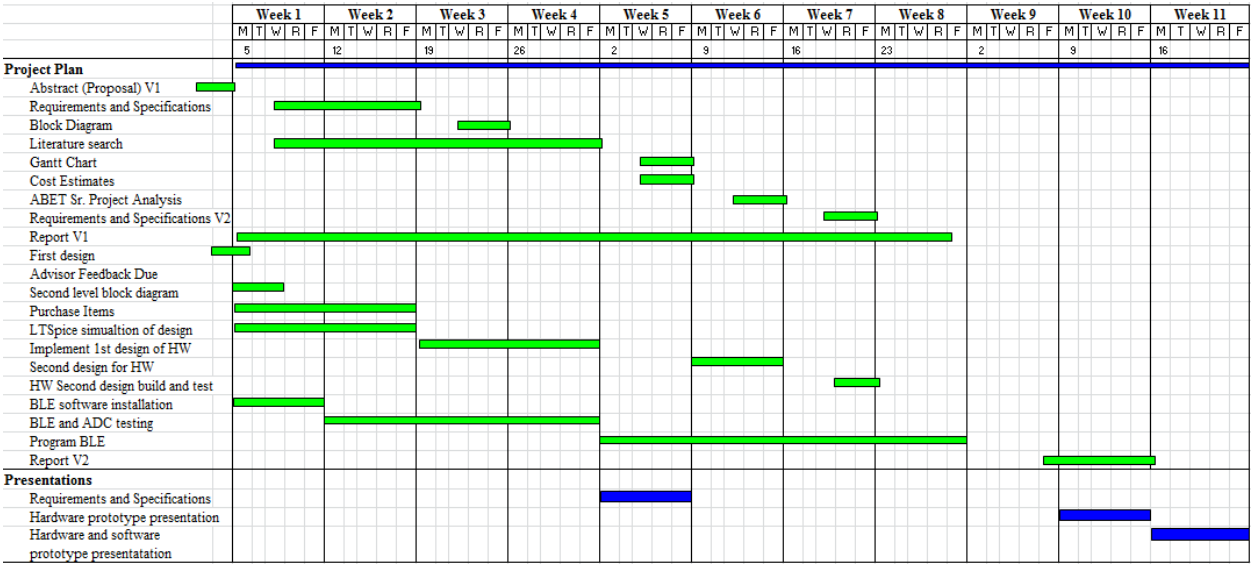


Figure 15: Winter Quarter Gantt Chart

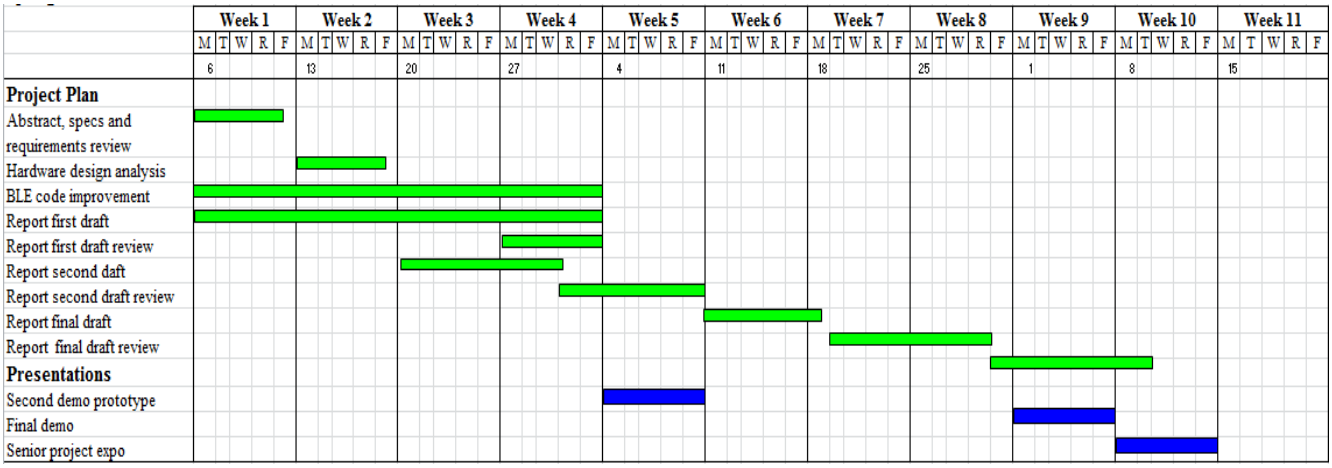


Figure 16: Fall Quarter Gantt Chart

If manufactured on a commercial basis

a) Estimated number of devices sold per year: 100
b) Estimated manufacturing cost for each devices: $100
c) Estimated purchase price for each device: 150
d) Estimated profit per year:5,000
e) Estimated cost for user to operate device, per unit time: approximately 2 years

Environmental

a) Environmental Impact: One must decompose of the solder and wiring waste to its proper containers. Also there is plastic waste from the manufacturing of all the components bought need to be dispose correctly.
b) Which natural resources and ecosystem services does the project use directly or indirectly?
   The project requires electrical energy provided by the batteries. There is an indirect effect on the environment by all the trucks that deliver the batteries and all components for the project. During the transportation process trucks will produce carbon dioxin which has a negative effect on the environment.
c) What natural resources and ecosystem services does the project improve or harm?
   The project could potentially hurt the ecosystem if the product is not dispose correctly.
d) How does the project impact other species?
   The project does not pose an impact on other species.

Manufacturability

One of the manufacturing issues that could be presented is that the sensor is very sensitive and doing a mass production will need a better design of a sensor.
Sustainability: A calibration system is needed to ensure the sensor gives proper readings. Also the design could improve by reducing the number of components used which decrease the environmental waste.

Ethical

Utilitarian ethics is one of the used through the project. Utilitarian ethics states that decisions are based upon the decision that will produce the highest good for all. The permanent magnet display system tries to improve the technology of similar products. Thought the project decisions are made ensure it produces the best outcome for the project. The project will also use IEEE code of ethics (item 3,7 and 6) . The design is tested numerous times to ensure its quality and proper readings are given. The design is tested at each stage to ensure proper results. Previous designs and codes are studied to ensure the best method is used. Throughout the project reviewers should be able to offer honest criticism in a constructive way for the benefit of the project. One must accept suggestions in a positive way and apply them to the project.

Health and Safety

One of the positive health outcomes consists of indirectly providing data which in the near future is going to allow patients with pacemakers to receive an MRI scan. A negative effect of the

project can arise if the necessary precautions are not taken while soldering all the components to the board.

Social and Political

Political and social issues could arise with users of the product and companies who develop similar technology on the market. St. Jude Medical is a direct beneficiary because they are using during pacemaker characterization. Other companies that develop electromagnetic sensor could be harmed by the addition of a new design as it can be seen as a competition.

Development

The Gantt chart is used as the main development planning tool. The Gantt chart provides clear milestones thoughtout the project and all the key progress presentations. The literature research provides valuable information which provides valuable information to further understanding permanent magnet and Bluetooth technology. Sending data via Bluetooth and learning the protocols and standards is one of the personal development the literature search provides.

Appendix B. Code

```
/* Copyright (c) 2013 Nordic Semiconductor. All Rights Reserved.
 *
 * Use of this source code is governed by a BSD-style license that can be
 * found in the license.txt file.
 */


#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "app_error.h"
#include "nrf_gpio.h"
#include "nrf51_bitfields.h"
#include "ble.h"
#include "ble_hci.h"
#include "ble_srv_common.h"
#include "ble_advdata.h"
#include "ble_bas.h"
#include "ble_hrs.h"
#include "ble_dis.h"
#include "ble_conn_params.h"
#include "boards.h"
#include "ble_sensorsim.h"
#include "softdevice_handler.h"
#include "app_timer.h"
#include "ble_error_log.h"
#include "device_manager.h"
#include "ble_debug_assert_handler.h"
#include "pstorage.h"
#include "app_trace.h"
#include "simple_uart.h"



#define IS_SRVC_CHANGED_CHARACT_PRESENT  0                                          /**< Include or not the
service_changed characteristic. if not enabled, the server's database cannot be changed for the lifetime
of the device*/

#define WAKEUP_BUTTON_PIN                BUTTON_0                                   /**< Button used to wake up
the application. */
#define BOND_DELETE_ALL_BUTTON_ID        BUTTON_1                                   /**< Button used for
deleting all bonded centrals during startup. */

#define ADVERTISING_LED_PIN_NO           LED_0                                      /**< Is on when device is
advertising. */
```

```
#define CONNECTED_LED_PIN_NO            LED_1                       /**< Is on when device has connected. */
#define ASSERT_LED_PIN_NO               LED_0                       /**< Is on when application has asserted. */

#define DEVICE_NAME                     "NorHRM"                    /**< Name of device. Will be included in the advertising data. */
#define MANUFACTURER_NAME               "NordicSemiconductor"       /**< Manufacturer. Will be passed to Device Information Service. */
#define APP_ADV_INTERVAL                40                          /**< The advertising interval (in units of 0.625 ms. This value corresponds to 25 ms). */
#define APP_ADV_TIMEOUT_IN_SECONDS      180                         /**< The advertising timeout in units of seconds. */

#define APP_TIMER_PRESCALER             0                           /**< Value of the RTC1 PRESCALER register. */
#define APP_TIMER_MAX_TIMERS            6                           /**< Maximum number of simultaneously created timers. */
#define APP_TIMER_OP_QUEUE_SIZE         4                           /**< Size of timer operation queues. */

#define BATTERY_LEVEL_MEAS_INTERVAL     APP_TIMER_TICKS(500, APP_TIMER_PRESCALER) /**< Battery level measurement interval (ticks). */
#define MIN_BATTERY_LEVEL               81                          /**< Minimum simulated battery level. */
#define MAX_BATTERY_LEVEL               100                         /**< Maximum simulated battery level. */
#define BATTERY_LEVEL_INCREMENT         1                           /**< Increment between each simulated battery level measurement. */

#define HEART_RATE_MEAS_INTERVAL        APP_TIMER_TICKS(1000, APP_TIMER_PRESCALER) /**< Heart rate measurement interval (ticks). */
#define MIN_HEART_RATE                  140                         /**< Minimum heart rate as returned by the simulated measurement function. */
#define MAX_HEART_RATE                  300                         /**< Maximum heart rate as returned by the simulated measurement function. */
#define HEART_RATE_INCREMENT            10                          /**< Value by which the heart rate is incremented/decremented for each call to the simulated measurement function. */

#define RR_INTERVAL_INTERVAL            APP_TIMER_TICKS(300, APP_TIMER_PRESCALER)  /**< RR interval interval (ticks). */
#define MIN_RR_INTERVAL                 100                         /**< Minimum RR interval as returned by the simulated measurement function. */
#define MAX_RR_INTERVAL                 500                         /**< Maximum RR interval as returned by the simulated measurement function. */
#define RR_INTERVAL_INCREMENT           1                           /**< Value by which the RR interval is incremented/decremented for each call to the simulated measurement function. */
```

```
#define SENSOR_CONTACT_DETECTED_INTERVAL    APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER)
/**< Sensor Contact Detected toggle interval (ticks). */

#define ADC_SAMPLING_INTERVAL            APP_TIMER_TICKS(100, APP_TIMER_PRESCALER) /**<
Sampling rate for the ADC */

#define MIN_CONN_INTERVAL                MSEC_TO_UNITS(500, UNIT_1_25_MS)        /**< Minimum
acceptable connection interval (0.5 seconds). */
#define MAX_CONN_INTERVAL                MSEC_TO_UNITS(1000, UNIT_1_25_MS)       /**< Maximum
acceptable connection interval (1 second). */
#define SLAVE_LATENCY                    0                                       /**< Slave latency. */
#define CONN_SUP_TIMEOUT                 MSEC_TO_UNITS(4000, UNIT_10_MS)         /**< Connection
supervisory timeout (4 seconds). */

#define FIRST_CONN_PARAMS_UPDATE_DELAY      APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER)
/**< 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,
APP_TIMER_PRESCALER)/**<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 SEC_PARAM_TIMEOUT                30                                      /**< Timeout for Pairing Request or
Security Request (in seconds). */
#define SEC_PARAM_BOND                   1                                       /**< Perform bonding. */
#define SEC_PARAM_MITM                   0                                       /**< Man In The Middle protection not
required. */
#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. */


static uint16_t                  m_conn_handle = BLE_CONN_HANDLE_INVALID;  /**< Handle of the
current connection. */
static ble_gap_adv_params_t      m_adv_params;                             /**< Parameters to be passed
to the stack when starting advertising. */
static ble_bas_t                 m_bas;                                    /**< Structure used to identify the battery
service. */
static ble_hrs_t                 m_hrs;                                    /**< Structure used to identify the heart rate
service. */
```

```
static bool                     m_rr_interval_enabled = true;           /**< Flag for enabling and disabling
the registration of new RR interval measurements (the purpose of disabling this is just to test sending
HRM without RR interval data. */

static ble_sensorsim_cfg_t      m_battery_sim_cfg;                      /**< Battery Level sensor
simulator configuration. */
static ble_sensorsim_state_t    m_battery_sim_state;                    /**< Battery Level sensor
simulator state. */
static ble_sensorsim_cfg_t      m_heart_rate_sim_cfg;                   /**< Heart Rate sensor
simulator configuration. */
static ble_sensorsim_state_t    m_heart_rate_sim_state;                 /**< Heart Rate sensor
simulator state. */
static ble_sensorsim_cfg_t      m_rr_interval_sim_cfg;                  /**< RR Interval sensor
simulator configuration. */
static ble_sensorsim_state_t    m_rr_interval_sim_state;                /**< RR Interval sensor
simulator state. */

static app_timer_id_t           m_battery_timer_id;                     /**< Battery timer. */
static app_timer_id_t           m_heart_rate_timer_id;                  /**< Heart rate measurement
timer. */
static app_timer_id_t           m_rr_interval_timer_id;                 /**< RR interval timer. */
static app_timer_id_t           m_sensor_contact_timer_id;              /**< Sensor contact detected
timer. */
static app_timer_id_t           m_adc_sampling_timer_id;
                                /**< ADC timer */

static dm_application_instance_t m_app_handle;                          /**< Application identifier
allocated by device manager */

static bool                     m_memory_access_in_progress = false;    /**< Flag to keep track of
ongoing operations on persistent memory. */

//NORDIC these variables needs to be available in several functions
static ble_advdata_t            advdata;
static ble_advdata_manuf_data_t adc_data;
static uint8_t                  adc = 0x00; // Adc initail value

/**@brief Function for error handling, which is called when an error has occurred.
 *
 * @warning This handler is an example only and does not fit a final product. You need to analyze
 *          how your product is supposed to react in case of error.
 *
 * @param[in] error_code  Error code supplied to the handler.
 * @param[in] line_num    Line number where the handler is called.
 * @param[in] p_file_name Pointer to the file name.
 */
void app_error_handler(uint32_t error_code, uint32_t line_num, const uint8_t * p_file_name)
{
```

```
    nrf_gpio_pin_set(ASSERT_LED_PIN_NO);

    // This call can be used for debug purposes during application development.
    // @note CAUTION: Activating this code will write the stack to flash on an error.
    //              This function should NOT be used in a final product.
    //              It is intended STRICTLY for development/debugging purposes.
    //              The flash write will happen EVEN if the radio is active, thus interrupting
    //              any communication.
    //              Use with care. Uncomment the line below to use.
    // ble_debug_assert_handler(error_code, line_num, p_file_name);

    // On assert, the system can only recover with a reset.
    NVIC_SystemReset();
    while(1);
}


/**@brief Callback function for asserts in the SoftDevice.
 *
 * @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 analyze
 *          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]   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 Function for performing battery measurement and updating the Battery Level characteristic
 *        in Battery Service.
 */
static void battery_level_update(void)
{
    uint32_t err_code;
    uint8_t  battery_level;

    //battery_level = (uint8_t)ble_sensorsim_measure(&m_battery_sim_state, &m_battery_sim_cfg);

            battery_level= adc;

                    //battery_level = 0x05;
```

```
    err_code = ble_bas_battery_level_update(&m_bas, battery_level);
    if ((err_code != NRF_SUCCESS) &&
        (err_code != NRF_ERROR_INVALID_STATE) &&
        (err_code != BLE_ERROR_NO_TX_BUFFERS) &&
        (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
    )
    {
        APP_ERROR_HANDLER(err_code);
    }
}


/**@brief Function for handling the Battery measurement timer timeout.
 *
 * @details This function will be called each time the battery level measurement timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void battery_level_meas_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
    battery_level_update();

    // NORDIC reinitate the complete advertising. Just copy paste the code form advertise_init() and
update the adc value
    uint32_t err_code;
    uint8_t       flags = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
    ble_uuid_t adv_uuids[] =
    {
        {BLE_UUID_HEART_RATE_SERVICE,        BLE_UUID_TYPE_BLE},
        {BLE_UUID_BATTERY_SERVICE,           BLE_UUID_TYPE_BLE},
        {BLE_UUID_DEVICE_INFORMATION_SERVICE, BLE_UUID_TYPE_BLE}
    };
    adc_data.company_identifier         = 0x0059; // Nordics company ID
    adc_data.data.p_data                = &adc;
    adc_data.data.size                  = sizeof(adc);

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

    advdata.name_type             = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance    = true;
    advdata.flags.size            = sizeof(flags);
    advdata.flags.p_data          = &flags;
    advdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) / sizeof(adv_uuids[0]);
    advdata.uuids_complete.p_uuids  = adv_uuids;
    // NORDIC add adc value to data packet
```

```
    advdata.p_manuf_specific_data  = &adc_data;

    err_code = ble_advdata_set(&advdata, NULL);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling the Heart rate measurement timer timeout.
 *
 * @details This function will be called each time the heart rate measurement timer expires.
 *          It will exclude RR Interval data from every third measurement.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void heart_rate_meas_timeout_handler(void * p_context)
{
    static uint32_t cnt = 0;
    uint32_t        err_code;
    uint16_t        heart_rate;

    UNUSED_PARAMETER(p_context);

    heart_rate = (uint16_t)ble_sensorsim_measure(&m_heart_rate_sim_state, &m_heart_rate_sim_cfg);

    cnt++;
    err_code = ble_hrs_heart_rate_measurement_send(&m_hrs, heart_rate);
    if ((err_code != NRF_SUCCESS) &&
        (err_code != NRF_ERROR_INVALID_STATE) &&
        (err_code != BLE_ERROR_NO_TX_BUFFERS) &&
        (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
    )
    {
        APP_ERROR_HANDLER(err_code);
    }

    // Disable RR Interval recording every third heart rate measurement.
    // NOTE: An application will normally not do this. It is done here just for testing generation
    //       of messages without RR Interval measurements.
    m_rr_interval_enabled = ((cnt % 3) != 0);
}

// ADC timer handler to start ADC sampling
static void adc_sampling_timeout_handler(void * p_context)
{
            uint32_t p_is_running = 0;

            sd_clock_hfclk_request();
```

```
        while(!p_is_running) {                                          //wait for the hfclk to
be available
                sd_clock_hfclk_is_running((&p_is_running));
        }
        nrf_gpio_pin_toggle(LED_1);             //Toggle LED2 to indicate start of sampling
        NRF_ADC->TASKS_START = 1;                                       //Start ADC
sampling
}

/**@brief Function for handling the RR interval timer timeout.
 *
 * @details This function will be called each time the RR interval timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void rr_interval_timeout_handler(void * p_context)
{
  UNUSED_PARAMETER(p_context);

  if (m_rr_interval_enabled)
  {
    uint16_t rr_interval;

    rr_interval = (uint16_t)ble_sensorsim_measure(&m_rr_interval_sim_state,
                                  &m_rr_interval_sim_cfg);
    ble_hrs_rr_interval_add(&m_hrs, rr_interval);
  }
}


/**@brief Function for handling the Sensor Contact Detected timer timeout.
 *
 * @details This function will be called each time the Sensor Contact Detected timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void sensor_contact_detected_timeout_handler(void * p_context)
{
  static bool sensor_contact_detected = false;

  UNUSED_PARAMETER(p_context);

  sensor_contact_detected = !sensor_contact_detected;
  ble_hrs_sensor_contact_detected_update(&m_hrs, sensor_contact_detected);
}
```

```
/**@brief Function for the LEDs initialization.
 *
 * @details Initializes all LEDs used by this application.
 */
static void leds_init(void)
{
    nrf_gpio_cfg_output(ADVERTISING_LED_PIN_NO);
    nrf_gpio_cfg_output(CONNECTED_LED_PIN_NO);
                        nrf_gpio_cfg_output(LED_1);                 //Added for ADC operation
                nrf_gpio_cfg_output(LED_1);                 //Added for ADC operation
    nrf_gpio_cfg_output(ASSERT_LED_PIN_NO);

                        nrf_gpio_port_dir_set(NRF_GPIO_PORT_SELECT_PORT2,
NRF_GPIO_PORT_DIR_OUTPUT);  //Added output to show ADC result
}


/**@brief Function for the Timer initialization.
 *
 * @details Initializes the timer module. This creates and starts application timers.
 */
static void timers_init(void)
{
    uint32_t err_code;

    // Initialize timer module.
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_MAX_TIMERS, APP_TIMER_OP_QUEUE_SIZE,
false);

    // Create timers.
    err_code = app_timer_create(&m_battery_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                battery_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_create(&m_heart_rate_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                heart_rate_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_create(&m_rr_interval_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                rr_interval_timeout_handler);
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_create(&m_sensor_contact_timer_id,
                                APP_TIMER_MODE_REPEATED,
```

```
                    sensor_contact_detected_timeout_handler);
    APP_ERROR_CHECK(err_code);

                    err_code = app_timer_create(&m_adc_sampling_timer_id,
                      APP_TIMER_MODE_REPEATED,
                      adc_sampling_timeout_handler);
    APP_ERROR_CHECK(err_code);
}


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

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

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

    err_code =
sd_ble_gap_appearance_set(BLE_APPEARANCE_HEART_RATE_SENSOR_HEART_RATE_BELT);
    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 initializing the Advertising functionality.
 *
 * @details Encodes the required advertising data and passes it to the stack.
 *          Also builds a structure to be passed to the stack when starting advertising.
```

```
 */
static void advertising_init(void)
{
    uint32_t      err_code;
    // NORDIC. Moved some variables to make them accessible in all functions
    uint8_t       flags = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;

    ble_uuid_t adv_uuids[] =
    {
        {BLE_UUID_HEART_RATE_SERVICE,        BLE_UUID_TYPE_BLE},
        {BLE_UUID_BATTERY_SERVICE,           BLE_UUID_TYPE_BLE},
        {BLE_UUID_DEVICE_INFORMATION_SERVICE, BLE_UUID_TYPE_BLE}
    };

    // NORDIC Prepare the Manufacturer specific data packet
    adc_data.company_identifier         = 0x0059; // Nordics company ID
    adc_data.data.p_data                = &adc;
    adc_data.data.size                  = sizeof(adc);

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

    advdata.name_type             = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance    = true;
    advdata.flags.size            = sizeof(flags);
    advdata.flags.p_data          = &flags;
    advdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) / sizeof(adv_uuids[0]);
    advdata.uuids_complete.p_uuids  = adv_uuids;
    // NORDIC add Manufacturer specific data packet
    advdata.p_manuf_specific_data  = &adc_data;

    err_code = ble_advdata_set(&advdata, NULL);
    APP_ERROR_CHECK(err_code);

    // Initialize advertising parameters (used when starting advertising).
    memset(&m_adv_params, 0, sizeof(m_adv_params));

    m_adv_params.type       = BLE_GAP_ADV_TYPE_ADV_IND;
    m_adv_params.p_peer_addr = NULL;                    // Undirected advertisement.
    m_adv_params.fp         = BLE_GAP_ADV_FP_ANY;
    m_adv_params.interval   = APP_ADV_INTERVAL;
    m_adv_params.timeout    = APP_ADV_TIMEOUT_IN_SECONDS;
}


/**@brief Function for initializing services that will be used by the application.
 *
 * @details Initialize the Heart Rate, Battery and Device Information services.
```

```
 */
static void services_init(void)
{
    uint32_t       err_code;
    ble_hrs_init_t hrs_init;
    ble_bas_init_t bas_init;
    ble_dis_init_t dis_init;

            ble_advdata_t adc_init; // Xenia
    uint8_t       body_sensor_location;
   ////////////////////////////////////////////////////////////////////////////
    // Initialize Heart Rate Service.
    body_sensor_location = BLE_HRS_BODY_SENSOR_LOCATION_FINGER;

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

    hrs_init.evt_handler                 = NULL;
    hrs_init.is_sensor_contact_supported = true;
    hrs_init.p_body_sensor_location      = &body_sensor_location;

    // Here the sec level for the Heart Rate Service can be changed/increased.
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&hrs_init.hrs_hrm_attr_md.cccd_write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&hrs_init.hrs_hrm_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&hrs_init.hrs_hrm_attr_md.write_perm);

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&hrs_init.hrs_bsl_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&hrs_init.hrs_bsl_attr_md.write_perm);

    err_code = ble_hrs_init(&m_hrs, &hrs_init);
    APP_ERROR_CHECK(err_code);

    // Initialize Battery Service.
    memset(&bas_init, 0, sizeof(bas_init));


    // Here the sec level &adcfor the Battery Service can be changed/increased.
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_char_attr_md.cccd_write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_char_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&bas_init.battery_level_char_attr_md.write_perm);

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_report_read_perm);

    bas_init.evt_handler         = NULL;
    bas_init.support_notification = true;
    bas_init.p_report_ref        = NULL;
    bas_init.initial_batt_level  = 100;

    err_code = ble_bas_init(&m_bas, &bas_init);
```

```
    APP_ERROR_CHECK(err_code);

                    // Initialize adc service Xenia
                    memset(&adc_init,0,sizeof(adc_init)); //Xenia

    // Initialize Device Information Service.
    memset(&dis_init, 0, sizeof(dis_init));

    ble_srv_ascii_to_utf8(&dis_init.manufact_name_str, (char *)MANUFACTURER_NAME);

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&dis_init.dis_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&dis_init.dis_attr_md.write_perm);

    err_code = ble_dis_init(&dis_init);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for initializing the sensor simulators.
 */
static void sensor_sim_init(void)
{
    m_battery_sim_cfg.min          = MIN_BATTERY_LEVEL;
    m_battery_sim_cfg.max          = MAX_BATTERY_LEVEL;
    m_battery_sim_cfg.incr         = BATTERY_LEVEL_INCREMENT;
    m_battery_sim_cfg.start_at_max = true;

    ble_sensorsim_init(&m_battery_sim_state, &m_battery_sim_cfg);

    m_heart_rate_sim_cfg.min          = MIN_HEART_RATE;
    m_heart_rate_sim_cfg.max          = MAX_HEART_RATE;
    m_heart_rate_sim_cfg.incr         = HEART_RATE_INCREMENT;
    m_heart_rate_sim_cfg.start_at_max = false;

    ble_sensorsim_init(&m_heart_rate_sim_state, &m_heart_rate_sim_cfg);

    m_rr_interval_sim_cfg.min          = MIN_RR_INTERVAL;
    m_rr_interval_sim_cfg.max          = MAX_RR_INTERVAL;
    m_rr_interval_sim_cfg.incr         = RR_INTERVAL_INCREMENT;
    m_rr_interval_sim_cfg.start_at_max = false;

    ble_sensorsim_init(&m_rr_interval_sim_state, &m_rr_interval_sim_cfg);
}


/**@brief Function for starting application timers.
 */
static void application_timers_start(void)
```

```
{
  uint32_t err_code;

  // Start application timers.
  err_code = app_timer_start(m_battery_timer_id, BATTERY_LEVEL_MEAS_INTERVAL, NULL);
  APP_ERROR_CHECK(err_code);

  err_code = app_timer_start(m_heart_rate_timer_id, HEART_RATE_MEAS_INTERVAL, NULL);
  APP_ERROR_CHECK(err_code);

  err_code = app_timer_start(m_rr_interval_timer_id, RR_INTERVAL_INTERVAL, NULL);
  APP_ERROR_CHECK(err_code);

  err_code = app_timer_start(m_sensor_contact_timer_id, SENSOR_CONTACT_DETECTED_INTERVAL,
NULL);
  APP_ERROR_CHECK(err_code);

                  //ADC timer start
                  err_code = app_timer_start(m_adc_sampling_timer_id, ADC_SAMPLING_INTERVAL,
NULL);
  APP_ERROR_CHECK(err_code);
}


/**@brief Function for starting advertising.
 */
static void advertising_start(void)
{
  uint32_t err_code;
  uint32_t count;

  // Verify if there is any flash access pending, if yes delay starting advertising until
  // it's complete.
  err_code = pstorage_access_status_get(&count);
  APP_ERROR_CHECK(err_code);

  if (count != 0)
  {
    m_memory_access_in_progress = true;
    return;
  }

  err_code = sd_ble_gap_adv_start(&m_adv_params);
  APP_ERROR_CHECK(err_code);

  nrf_gpio_pin_set(ADVERTISING_LED_PIN_NO);
}
```

```
/**@brief Function for handling 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 a Connection Parameters error.
 *
 * @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    = m_hrs.hrm_handles.cccd_handle;
    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 handling the Application's BLE Stack events.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t err_code;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            nrf_gpio_pin_set(CONNECTED_LED_PIN_NO);
            nrf_gpio_pin_clear(ADVERTISING_LED_PIN_NO);

            m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            break;

        case BLE_GAP_EVT_DISCONNECTED:
            nrf_gpio_pin_clear(CONNECTED_LED_PIN_NO);


            advertising_start();
            break;

        case BLE_GAP_EVT_TIMEOUT:
            if (p_ble_evt->evt.gap_evt.params.timeout.src == BLE_GAP_TIMEOUT_SRC_ADVERTISEMENT)
            {
                nrf_gpio_pin_clear(ADVERTISING_LED_PIN_NO);

                // 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);
            }
            break;

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

```
/**@brief Function for handling the Application's system events.
 *
 * @param[in]   sys_evt   system event.
 */
static void on_sys_evt(uint32_t sys_evt)
{
    switch(sys_evt)
    {
        case NRF_EVT_FLASH_OPERATION_SUCCESS:
        case NRF_EVT_FLASH_OPERATION_ERROR:
            if (m_memory_access_in_progress)
            {
                m_memory_access_in_progress = false;
                advertising_start();
            }
            break;
        default:
            // No implementation needed.
            break;
    }
}

/**@brief Function for dispatching a BLE stack event to all modules with a BLE stack event handler.
 *
 * @details This function is called from the BLE Stack event interrupt handler after a BLE stack
 *          event has been received.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
    dm_ble_evt_handler(p_ble_evt);
    ble_hrs_on_ble_evt(&m_hrs, p_ble_evt);
    ble_bas_on_ble_evt(&m_bas, p_ble_evt);
    ble_conn_params_on_ble_evt(p_ble_evt);
    on_ble_evt(p_ble_evt);
}


/**@brief Function for dispatching a system event to interested modules.
 *
 * @details This function is called from the System event interrupt handler after a system
 *          event has been received.
 *
 * @param[in]   sys_evt   System stack event.
 */
static void sys_evt_dispatch(uint32_t sys_evt)
```

```
{
  pstorage_sys_event_handler(sys_evt);
  on_sys_evt(sys_evt);
}


/**@brief Function for initializing the BLE stack.
 *
 * @details Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
  uint32_t err_code;

  // Initialize the SoftDevice handler module.
  SOFTDEVICE_HANDLER_INIT(NRF_CLOCK_LFCLKSRC_XTAL_20_PPM, false);

#ifdef S110
  // Enable BLE stack
  ble_enable_params_t ble_enable_params;
  memset(&ble_enable_params, 0, sizeof(ble_enable_params));
  ble_enable_params.gatts_enable_params.service_changed =
IS_SRVC_CHANGED_CHARACT_PRESENT;
  err_code = sd_ble_enable(&ble_enable_params);
  APP_ERROR_CHECK(err_code);
#endif

  // Register with the SoftDevice handler module for BLE events.
  err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
  APP_ERROR_CHECK(err_code);

  // Register with the SoftDevice handler module for BLE events.
  err_code = softdevice_sys_evt_handler_set(sys_evt_dispatch);
  APP_ERROR_CHECK(err_code);
}


/**@brief Function for initializing buttons.
 */
static void buttons_init(void)
{
  // Set Wakeup and Bonds Delete buttons as wakeup sources.
  nrf_gpio_cfg_sense_input(WAKEUP_BUTTON_PIN,
                           BUTTON_PULL,
                           NRF_GPIO_PIN_SENSE_LOW);

  nrf_gpio_cfg_sense_input(BOND_DELETE_ALL_BUTTON_ID,
                           BUTTON_PULL,
```

```
                    NRF_GPIO_PIN_SENSE_LOW);
}


/**@brief Function for handling the Device Manager events.
 *
 * @param[in]   p_evt   Data associated to the device manager event.
 */
static uint32_t device_manager_evt_handler(dm_handle_t const   * p_handle,
                                           dm_event_t const    * p_event,
                                           api_result_t        event_result)
{
    APP_ERROR_CHECK(event_result);
    return NRF_SUCCESS;
}


/**@brief Function for the Device Manager initialization.
 */
static void device_manager_init(void)
{
    uint32_t            err_code;
    dm_init_param_t     init_data;
    dm_application_param_t register_param;

    // Initialize persistent storage module.
    err_code = pstorage_init();
    APP_ERROR_CHECK(err_code);

    // Clear all bonded centrals if the Bonds Delete button is pushed.
    init_data.clear_persistent_data = (nrf_gpio_pin_read(BOND_DELETE_ALL_BUTTON_ID) == 0);

    err_code = dm_init(&init_data);
    APP_ERROR_CHECK(err_code);

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

    register_param.sec_param.timeout      = SEC_PARAM_TIMEOUT;
    register_param.sec_param.bond         = SEC_PARAM_BOND;
    register_param.sec_param.mitm         = SEC_PARAM_MITM;
    register_param.sec_param.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    register_param.sec_param.oob          = SEC_PARAM_OOB;
    register_param.sec_param.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    register_param.sec_param.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
    register_param.evt_handler            = device_manager_evt_handler;
    register_param.service_type           = DM_PROTOCOL_CNTXT_GATT_SRVR_ID;

    err_code = dm_register(&m_app_handle, &register_param);
```

```
   APP_ERROR_CHECK(err_code);
}


/**@brief Function for the Power manager.
 */
static void power_manage(void)
{
   uint32_t err_code = sd_app_evt_wait();
   APP_ERROR_CHECK(err_code);
}

//ADC initialization
static void adc_init(void)
{
        /* Enable interrupt on ADC sample ready event*/
        NRF_ADC->INTENSET = ADC_INTENSET_END_Msk;
        sd_nvic_SetPriority(ADC_IRQn, NRF_APP_PRIORITY_LOW);
        sd_nvic_EnableIRQ(ADC_IRQn);

        NRF_ADC->CONFIG      = (ADC_CONFIG_EXTREFSEL_None << ADC_CONFIG_EXTREFSEL_Pos) /*
Bits 17..16 : ADC external reference pin selection. */
                                                                |
(ADC_CONFIG_PSEL_AnalogInput2 << ADC_CONFIG_PSEL_Pos)                                   /*!<
Use analog input 2 as analog input. */
                                                                |
(ADC_CONFIG_REFSEL_SupplyOneThirdPrescaling << ADC_CONFIG_REFSEL_Pos)
                        /*!< Use internal 1.2V bandgap voltage as reference for conversion. */
                                                                |
(ADC_CONFIG_INPSEL_AnalogInputOneThirdPrescaling << ADC_CONFIG_INPSEL_Pos) /*!< Analog input
specified by PSEL with no prescaling used as input for the conversion. */
                                                                | (ADC_CONFIG_RES_8bit <<
ADC_CONFIG_RES_Pos);                                                                    /*!<
8bit ADC resolution. */

        /* Enable ADC*/
        NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;
}

/* Interrupt handler for ADC data ready event */
void ADC_IRQHandler(void)
{
   /* Clear dataready event */
   NRF_ADC->EVENTS_END = 0;
   // NORDIC adc value is updated here
   adc = NRF_ADC->RESULT;

   nrf_gpio_port_write(NRF_GPIO_PORT_SELECT_PORT2, adc); /* Write ADC result to port 2 */
```

```
   nrf_gpio_pin_toggle(LED_1);

   NRF_ADC->TASKS_STOP = 1;//Use the STOP task to save current. Workaround for PAN_028 rev1.5
anomaly 1.
      //Release the external crystal
   sd_clock_hfclk_release();
}

/**@brief Function for application main entry.
 */
int main(void)
{
   // Initialize.
   //simple_uart_config(RTS_PIN_NUMBER, TX_PIN_NUMBER, CTS_PIN_NUMBER, RX_PIN_NUMBER,
false);
   //simple_uart_putstring("Start\n\r");
   leds_init();
   buttons_init();
   ble_stack_init();

   timers_init();
   device_manager_init();
   gap_params_init();
   advertising_init();
   services_init();
   sensor_sim_init();
   conn_params_init();

                     adc_init();       //Initialize ADC

   // Start execution.
   application_timers_start();
   advertising_start();

   // Enter main loop.
   for (;;)
   {
      power_manage();
   }
}
```