Rose-Hulman Institute of Technology
Rose-Hulman Scholar

Graduate Theses - Electrical and Computer Engineering

Graduate Theses

Spring 5-2015

A Heart Rate Finger Ring and Its Smartphone APP Through Customized NFC

Yang Liu
Rose-Hulman Institute of Technology

Follow this and additional works at: http://scholar.rose-hulman.edu/electrical_grad_theses

Part of the Electrical and Electronics Commons

Recommended Citation
Liu, Yang, "A Heart Rate Finger Ring and Its Smartphone APP Through Customized NFC" (2015). *Graduate Theses - Electrical and Computer Engineering.* Paper 6.

This Thesis is brought to you for free and open access by the Graduate Theses at Rose-Hulman Scholar. It has been accepted for inclusion in Graduate Theses - Electrical and Computer Engineering by an authorized administrator of Rose-Hulman Scholar. For more information, please contact bernier@rose-hulman.edu.

A HEART RATE FINGER RING AND ITS SMARTPHONE APP THROUGH CUSTOMIZED NFC

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

Yang Liu

In Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Electrical Engineering

May 2015

ABSTRACT

Yang Liu

M.S. in E.E.

Rose-Hulman Institute of Technology

May 2015

A Heart Rate Finger Ring and Its Smartphone APP through Customized NFC

Major Professor: Dr. Jianjian Song

Population aging has become one of the most critical problems in contemporary society. Families and organizations are striving to provide better healthcare to the elderly and handicapped for their better living conditions. Due to these situations, the demand for remote health monitoring continues to grow rapidly. With the development of new technologies, such as smaller sensors and microcontrollers, the increasing widespread use of smartphones, and new wireless communication methods, a wireless body area network system can be constructed to provide more sophisticated solutions to satisfy this demand.

The objective of this thesis is to demonstrate that such a system is feasible. A ring-shaped hardware device is implemented to measure the user's heart rate and transfers the data to an Android phone through a customized Near Field Communication (NFC) tag. The tag is composed of a transponder to write data and a customized antenna to transfer data based on the resonance effect. An application is also developed to operate the NFC module to communicate

with the tag. Data is then received, stored, and utilized on the phone. The ring and Android phone serve as Body Sensor Unit (BSU) and Body Central Unit (BCU) respectively in the Wireless Body Area Network (WBAN) system. Then NFC technology links them together wirelessly.

In order to implement the NFC Ring, a sensor is placed within the ring to convert the heart rate into an electric signal. This signal is filtered and amplified and sent to a microcontroller. Next, the microcontroller generates a count for computing the time interval between two pulses. Then the count value is written to the NFC tag through an NFC transponder. The antenna is specially designed to meet two core constraints: the size should be as small as possible to fit the ring, while still maintaining the ability to produce a large enough magnetic field. When an Android phone approaches the ring, the application on the phone will execute and read data in the tag by controlling the NFC reader. After being received, the data is stored in a SQLite database on the phone for further processing, such as rendering a history chart to show the trend.

A prototype of this system has been developed to demonstrate the idea. This prototype can accurately read the heart rate per minute. Compared with a Radio-Frequency Identification ring, the NFC Ring has reduced system complexity and improved mobility. There are many possible improvements on both hardware and software. For instance, more research on NFC antenna design to enhance the stability of data transmission should be considered. The algorithm of heart rate measurement may be refined to generate more accurate data. More explanation of heart rate data and its trend await further exploration as well.

To

Hong Fan and Chang Zhang

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Song, for all the encouragement and instructions he provided during my Master of Science study and for guiding me to the conclusion of my study at Rose-Hulman. Dr. Song first suggested to me to investigate the details of NFC wireless technology and pursue my interest in the area further. I would also like to thank my other committee members, Dr. Hoover and Dr. Boutell. My thesis idea originated from Dr. Hoover's inspiring ECE554 Instrumentation class that teaches about the electrical theory behind various kinds of medical instruments. The energetic teaching style and effective course design of Dr. Boutell impressed me deeply, which attracted me to every one of his classes. I would especially like to thank Mrs. Dayhuff and Mr. Crosby for the assistance they provided me while I was purchasing materials and fabricating printed-circuit boards used in my research.

TABLE OF CONTENTS

ABSTRACT III
LIST OF FIGURES X
LIST OF TABLES XII
LIST OF ABBREVIATIONS XIII

1. INTRODUCTION 1
  1.1. Overview of Wireless Body Area Network 2
  1.2. TCRT1000 Reflective Optical Sensor 4
  1.3. Near Field Communication and the Transponder 4
  1.4. System Architecture of the NFC Ring 6
2. THE HEARTBEAT SENSING SYSTEM 8
  2.1. Pulse Detection 9
  2.2. Signal Extraction and Pulse Amplification 10
  2.3. The Microcontroller Interface 12
  2.4. Frequency Calculation 14
  2.5. The Operations on The Transponder 18
3. NFC ANTENNA DESIGN 22
  3.1. The RF Interface Circuits 22
  3.2. The Inductance Coil 23
  3.3. The External Capacitor 24
  3.4. The Antenna Q Factor 26
4. THE SMARTRING ANDROID APPLICATION 29
  4.1. The SmartRing Application Architecture 29
  4.2. Application Functions 33
    4.2.1. Receiving Data from NFC 33
    4.2.2. Database Design 36
    4.2.3. Personalized Analysis 38
  4.3. The SmartRing APP Interfaces 39
5. NFC RING PROTOTYPE AND COMPARISON WITH A RFID RING 45
  5.1. A Prototype of the NFC Ring 45

5.2. Comparison with A RFID Ring 47
6. CONCLUSION AND FUTURE WORK 49
LIST OF REFERENCES.. 52
APPENDICES 55
Appendix A. Circuit Diagrams 55
Appendix B. PCB Layout Plots 58
Appendix C. Software on Microcontroller.. 61
Appendix D. Software on Android 66

LIST OF FIGURES

Figure 1.1 The Functional Block Diagram of RF430CL330H [26] 6
Figure 1.2 The System Architecture 6
Figure 2.1 The Reflectance Approach Using Finger Tip [17] 9
Figure 2.2 The TCRT1000/1010 Sensor and Its Inner Circuit [10]... 10
Figure 2.3 The First Stage of Signal conditioning 11
Figure 2.4 The Output Signal of the Signal Conditioning Stage 12
Figure 2.5 The Device Pinout of MSP430G2x53, 20-Pin Devices [31]....................................... 13
Figure 2.6 Importing Library Files 14
Figure 2.7 The Global Variables.. 15
Figure 2.8 The Setting Up Function 16
Figure 2.9 The RF430CL330H I2C Operation Diagram [26] 18
Figure 2.10 Example Data Definition File.. 20
Figure 2.11 Example of Writing Data to NFC Module 21
Figure 3.1 The RF Interface Module [26]... 22
Figure 3.2 The Antenna Inductance Coil 24
Figure 3.3 The Circuit of the NFC Transponder and Its Antenna 25
Figure 3.4 The Frequency and Bandwidth Measurement Experiment 26
Figure 3.5 The Frequency Response of the Antenna on a Spectrum Analyzer 28
Figure 4.1 The SmartRing APP Architecture 29
Figure 4.2 The Flowchart of the SmartRing APP... 31
Figure 4.3 Example Permissions in AndroidManifest.xml File 33
Figure 4.4 Example of Intent Filter... 34
Figure 4.5 Example of NFCAdapter Class 34
Figure 4.6 Example of onResume() Method... 35
Figure 4.7 Example of processIntent() Method 35
Figure 4.8 The Pulse Rate Table Design 36
Figure 4.9 The Example SQL Sentence to Generate History Graph 36
Figure 4.10 The Java Code for Getting the Values of Last Seven Days 37
Figure 4.11 The Icon Design 39

Figure 4.12 The Icon in Android System Launcher 40
Figure 4.13 The Measuring Activity.. 41
Figure 4.14 The Navigation Drawer 41
Figure 4.15 The Analysis Activity: History and Report.. 42
Figure 4.16 The Settings Activity.. 43
Figure 4.17 Screenshots of Example Notification 44
Figure 5.1 The Prototype 45
Figure A.1 Ring control board schematic... 54
Figure A.2 Ring sensing board schematic 55
Figure A.3 Ring testing board schematic... 56
Figure B.1 Control board layout 57
Figure B.2 Sensor board layout 58
Figure B.3 Testing board layout 59

LIST OF TABLES

Table 2.1 The NDEF Application Data 19
Table 5.1 The Comparison Between the RFID Ring and the NFC Ring...................................... 48

LIST OF ABBREVIATIONS

NFC	Near Field Communication
WBAN	Wireless Body Area Network
BSU	Body Sensor Unit
BCU	Body Central Unit
APP	Application
RFID	Radio-Frequency Identification
SQL	Structured Query Language
MCU	Microcontroller Unit
HPM/BPM	Heartbeat/Beat Per Minute
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
HRM	Heart Rate Measurement
AC	Alternating Current
DC	Direct Current
RC	Resistance Capacitance
HPF	High Pass Filter
LPF	Low Pass Filter
TTL	Transistor-Transistor Logic
DCO	Digitally Controlled Oscillator
NDEF	NFC Data Exchange Format
URL	Uniform Resource Locator

1. INTRODUCTION

The objective of this thesis is to design and implement a Wireless Body Area Network (WBAN) system: a heart rate finger ring and an Android phone application that are linked wirelessly through a customized Near Field Communication (NFC) antenna, which is called the NFC Ring. The NFC Ring is an instance of Body Sensor Unit (BSU) that can sense a user's real-time heart rate and write it to its NFC tag. The tag is designed and adapted to the size of a ring. The heart rate data is transmitted to an Android phone with our application installed by controlling the NFC reader module and is recorded on the phone in its SQLite database. The application can also render the stored data into a graph and provide valuable insight, such as the trend of data and workout intensity levels.

For the NFC Ring to be successful, it must be able to initiate the reading process to get data from its ring whenever a phone approaches the ring. The size of the NFC ring hardware should be as small as a normal finger ring, so that the electrical parts can fit inside a ring-shape case. At the same time, the loop antenna should be as big as possible to maintain stable wireless connection between the ring and a phone. An ideal user scenario would be when a user is wearing the NFC Ring while holding an Android phone in the same hand, the measure process in the application will be triggered automatically and start to receive data from the ring.

To meet this goal, an NFC Ring prototype is built with a TCRT1000/1010 reflective optical sensor, MSP430G2553 microcontroller (MCU) and RF430CL330H NFC Transponder from Texas Instruments. A customized loop antenna on a printed-circuit board is connected to the transponder to interact with the NFC reader module on an Android phone. An application named “SmartRing” is developed from scratch to run on Android devices to receive, store, and process data.

Leveraging the features of NFC, especially NFC target and initiator modes and low power consumption, the NFC Ring is able to forward measured data to an Android phone. A natural gesture for connecting the ring with the app can also be derived from it. An ideal scenario can be when you are wearing a NFC Ring and holding your smart phone in your hands to get the reading. NFC can cover the range between ring and phone.

1.1. Overview of Wireless Body Area Network

One of the major issues of aging societies with fewer children is how to improve the living conditions of the elderly and the handicapped. In order to alleviate this problem, numerous social welfare services and supporting machines or systems have been developed [1-3]. In particular, families and various organizations are now jointly undertaking the responsibility of offering healthcare to the elderly and handicapped. The elderly often struggle to deal with medical emergencies. Providing ubiquitous healthcare that allows remote monitoring on their health conditions can be the most promising solution.

Wearable Computers have been experiencing a rapid development in the past two decades. After the smartphone was widely adopted and became a must-have daily electronic device of people, companies started to make products electronic and “smarter”. Daily things like a pair of glasses, a watch, and even a necklace have all become smarter and much more powerful. But these are only a small proportion of wearable computers. In general, everything we wear can be computer-based when equipped with a microcontroller, some sensors, and wireless communication. Then a device would be able to collect data on its own and send the data to more powerful devices. With the advances of wearable computers and smartphones, monitoring human health conditions becomes more achievable than ever before.

In fact, wearable computers are not an entirely new concept. The real technology behind wearable computers is the Wireless Body Area Network (WBAN), also referred to as Body Area Network (BAN) or Body Sensor Network (BSN), which has been developing for years since 1995 [4-6]. The earliest WBAN applications are expected to appear primarily in the healthcare domain, especially for helping the patients who suffers from chronic diseases by continuous monitoring and logging critical parameters. Extending the technology to new areas, for example, sports, can also help athletes by improving their training efficiency.

In dealing with WBAN, there is a continuing trend toward splitting the system into two parts: minor body sensor units (BSUs) and a single body central unit (BCU) [7][8]. The rapid growth in physiological sensors, low-power integrated circuits and wireless communication has provided enormous choices for BSUs. Together with smart phones and pads, which can act as perfect BCUs, these BSUs can focus on data acquisition and forward the collected data to BCU for storage and further processing. In addition to a data hub, BCU also provides a mature user interface to view and manage WBAN applications and a way for medical professionals to access data online regardless of the location of the patient.

The idea of our research is to leverage the trend to design and implement a WBAN system connected with NFC wirelessly, with a heart rate finger ring as a BSU and an Android phone as a BCU.

1.2. TCRT1000 Reflective Optical Sensor

The core of adding “feeling” to devices is using a sensor. A sensor can detect certain events or changes in quantities and provide a relevant output. Generally, the output is an electrical or optical signal [9]. Sensors play a significant role in making electronic devices smart. When a device is regarded as smart, it means that it is somehow conscious of its surrounding environment. In addition, the device can further automatically adjust itself according to what it feels, or connect with other devices, or even directly connect to the Internet. This automated process greatly reduces the need of constant manual operations and increases the flexibility in hardware design.

The sensor chosen in this paper to build the NFC Ring is TCRT1000 from Vishay Semiconductors [10]. With a built-in phototransistor and infrared emitter, the TCRT1000 sensor can project infrared light into tissues under skin and detect the change of luminosity when the light reflects back. This luminosity change is caused by the flow of blood, thus indicating the heart rate. Then the change is converted into an electric signal and the signal updates whenever there is a new pulse. As a consequence, the heart rate per minute value can be directly computed based on the frequency of the signal. The infrared light emitting diode (LED) is placed beside the detector in a leaded package, hence blocking the surrounding ambient light, which could otherwise impact the sensing performance [10].

1.3. Near Field Communication and the Transponder

NFC is a collection of technologies and ideas that enable smartphones or other devices to establish wireless communication within a close range, typically a distance of 10 cm or less. NFC is derived from Radio-Frequency Identification (RFID) technology, which allows an initiator to actively send out radio waves to power a passive electronic tag for identification, authentication, and tracking purposes. If both NFC devices are powered, it is possible to set up a peer-to-peer communication in between [11]. NFC operates at 13.56 MHz on ISO/IEC 18000-3 air interface. The relatively low frequency is the main reason why the communication range is limited. A complete NFC device has three work modes: NFC target, NFC initiator and NFC peer to peer. With loop antenna built inside, it can form an air-core transformer effectively with another loop antenna located in its near field through magnetic induction.

The advances in NFC bring about more applications than RFID. Former systems with RFID, such as contactless smart cards, support one-way communication only. Building upon RFID, NFC allows two-way communication between two devices. Therefore, it is able to build new systems or to replace earlier one-way applications, thus can be applied to additional fields [12]. For example, in contactless payment systems, Apple announced support for NFC-powered transactions in their Apple Pay program on Sept. 9, 2014. This kind of mobile payment could replace credit cards and electronic ticket smart cards in the near future. Other NFC applications are bootstrapping other more capable wireless connections, task automation, electronic identity documents, social networking, and gaming.

A customized NFC module is developed based on the Texas Instruments Dynamic NFC Interface Transponder RF430CL330H. Figure 1.1 shows its functional block diagram. The transponder is an NFC Tag Type 4 device that combines a wireless NFC interface and a wired serial peripheral interface (SPI) or inter-integrated circuit (I2C) interface to connect the chip to a host. With a microcontroller, the NFC Data Exchange Format (NDEF) message can be written and read from the integrated SPI or I2C serial communication interface to the SRAM. The message can also be accessed and updated wirelessly via the integrated ISO14443B-compliant

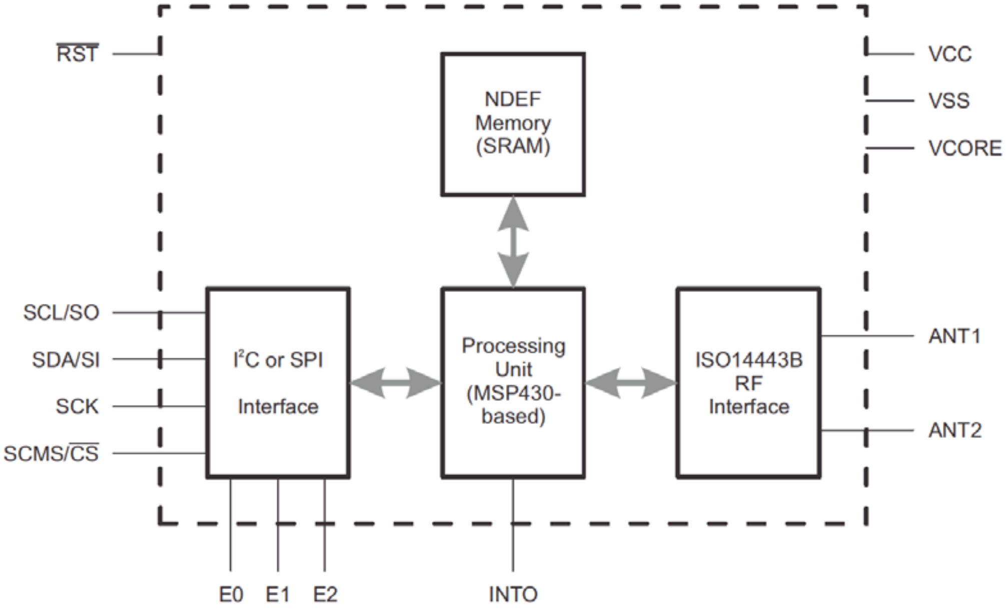


Figure 1.1 The Functional Block Diagram of RF430CL330H [26]

RF interface that supports data rates to 848 kbps at most. As a general-purpose NFC interface, the RF430CL330H empowers the wireless communication between an end-equipment and the rapidly spreading NFC-enabled smartphones, tablets, and notebooks [26].

1.4. System Architecture of the NFC Ring

As Figure 1.2 presents, the basic system architecture of the NFC Ring is composed of two main parts: hardware and software. The hardware part is mainly for heart rate data

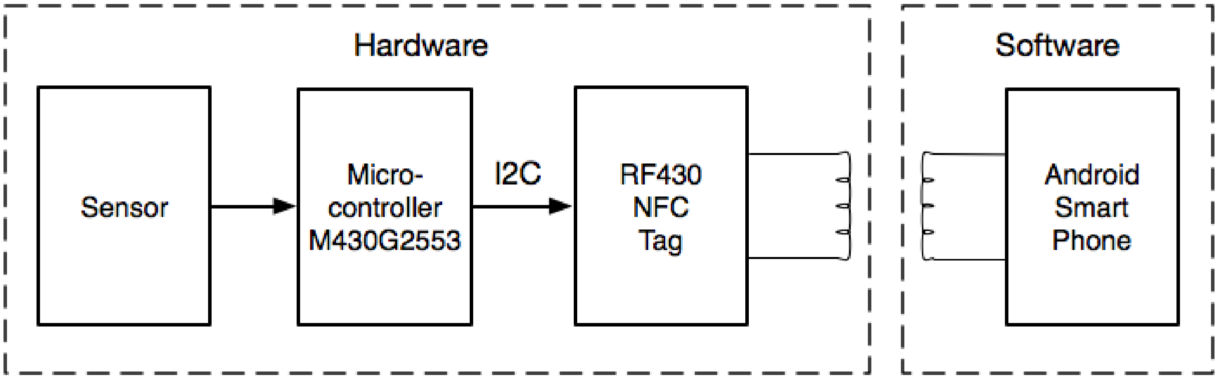


Figure 1.2 The System Architecture

acquisition. It also plays a part in forwarding data to an Android phone by having an NFC tag. The software part, on the other hand, is responsible for data receiving and data processing.

There are three main hardware operations: sensing, processing and forwarding. The NFC Ring must be able to sense a heart rate directly or some value change closely related to a heart rate and to generate an electric signal, thus data can be gained or calculated in later stages. Before processing, the signal is purified and amplified for accuracy. Then it is sent to an MCU that is responsible for converting the analog signal into a digital value. In order to transmit the value, it is encoded according to NFC message format convention and written to an NFC module through an I2C channel of MCU. If the message is correctly encoded, the NFC Tag can fully parse it and send it when an Android smartphone is within transmission range.

The software application running on an Android device aims at receiving, storing and utilizing the data. After data is written to the NFC tag, an Android phone will read the tag when it is close in range. Then the measurement process in the application will start and continuously obtain data from the NFC Ring. A final result for this measurement will be generated and stored in the SQLite database. Not only is the heart rate saved, but other important data, such as measured time, is inserted into the database as well. Hence, a graph based on the information can be rendered to show a recent trend. Some personalized heart rate range suggestions are also provided to user in this way.

2. THE HEARTBEAT SENSING SYSTEM

The main responsibility of our heart is to pump oxygen-rich blood to the muscles all around our body. Meanwhile, the circulation of our blood carries carbon dioxide and other cell waste away. The more intensive the exercise is, the more our muscles are used, and our heart also works harder to provide oxygen supply to these tasks. In other words, it beats faster to deliver more blood to every corner of our body.

To measure the condition of our heart, some kind of sensor needs to be implemented to take a sample of heartbeat signal in a certain period of time and computes the Beats per Minute (BPM) value so that the data is standardized. That is a heart rate monitor. There are two main methods to develop heart rate monitors - electrical and optical. The optical method in this thesis has low power consumption but is less accurate than electrical ones [14].

In order to measure heart rate of a person, his pulse can be measured instead because a pulse will follow each time the heart beats. Pulses can be easily measured using various techniques. Traditional methods, for example, using a stethoscope to listen to pulse, or merely pressing one’s fingers against an artery on the wrist or neck, are considered accurate to measure the pulse [15]. There are also many alternative methods for pulse measurements, such as Phonocardiogram (PCG), Electrocardiogram (ECG), blood pressure waveform, and pulse meters. But these methods are costly and clinical [14].

The economical method adopted in this thesis is a low power consumption heart rate measurement (HRM) device. It provides an accurate heart rate reading using optical technology. By combining a standard LED and photo-sensor, the optical technology is introduced to measure the heart rate on a fingertip in seconds. The heart rate is then forwarded by the microcontroller that counts the pulse and sends to a mobile device.

2.1. Pulse Detection

The theory of pulse sensor makes use of an observation on the change of ingredients, their proportions, and their biology properties. Hemoglobin is the oxygen-carrying pigment of red blood cells that gives them red color and serves to carry oxygen to tissues. There are two main forms: oxidized hemoglobin (HbO_2) and reduced-oxygen hemoglobin (Hb). One of the most significant features of this substance is how it reflects and absorbs light. Hb absorbs more (and reflects less) visible red light, while HbO_2 absorbs more (and reflects less) infrared light. Making use of this observation, there are two common measuring approaches: (1) transmissive approach that measures the light transmitted through tissue and (2) reflectance approach that measures the light reflected by tissue. The penetration depth of the light through organ tissue is restricted; hence the transmissive approach is only applicable to few body parts, such as a finger or the ear lope. While with reflectance approach, the light does not have to penetrate anything; it

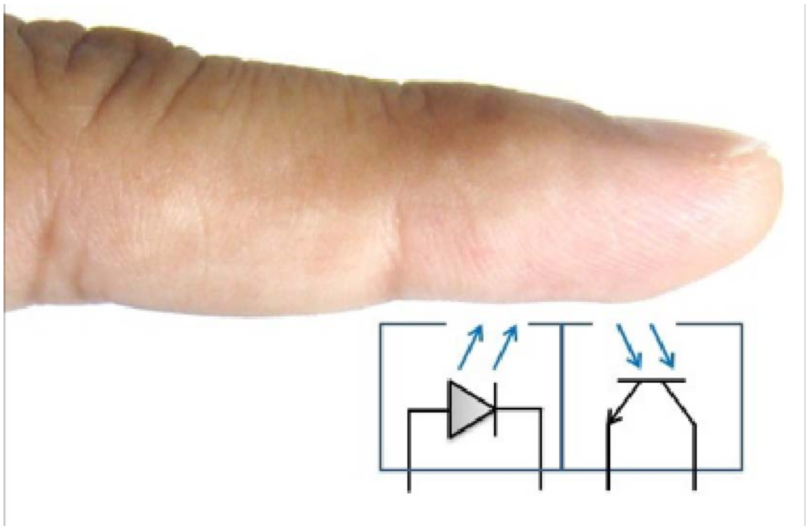


Figure 2.1 The Reflectance Approach Using Finger Tip [17]

can be applied to any parts of human body [17]. Therefore, the second approach is chosen in the design. A ray of infrared light is shed on a body part and the light reflected by tissue is measured. Figure 2.1 shows a fundamental reflectance probe to extract the pulse signal from the fingertip.

A pulse sensing unit with two red LEDs and a photo-sensor to measure one's heart rate through the change of blood reflectivity on a finger is proposed [14]. Our device uses the TCRT1000/1010 reflective optical sensor as the sensing unit instead. The use of TCRT1000 simplifies the build process of the sensor part, because the infrared light emitter diode and the detector are organized on the same side in a leaded package, thus blocking the surrounding ambient light, which can otherwise hurt the sensor performance [10]. TCRT1000/1010 sensor and the schematic circuit diagram are shown in Figure 2.2.

2.2. Signal Extraction and Pulse Amplification

The output signal of the sensor is a periodic physiological waveform attributed to small variations in the reflected infrared light, owing to the change of the HbO_2 volume in the pulsatile

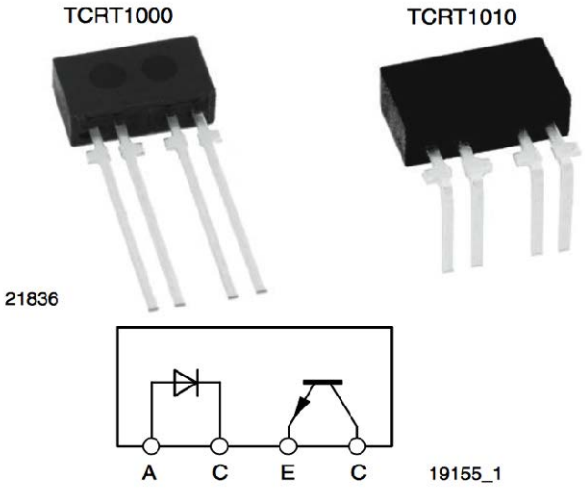


Figure 2.2 The TCRT1000/1010 Sensor and Its Inner Circuit [10]

blood. Two main components, AC and DC, compose the signal. The AC component is directly caused by pulsatile changes, which is simultaneous with the heartbeat, thus it can be used as a source of heart rate information. This AC component is superimposed onto a DC component that relates to the tissues and to the average blood volume. In order to measure the heart rate, the DC component should be removed to achieve a high signal-to-noise ratio. Since only a small portion of the original signal is the AC component with heart rate information, an effective amplification circuit is also required to extract desired signal. Figure 2.3 presents the circuit of first signal conditioning stage.

Two similar stages are used to refine the signal. Each stage has two filters to remove unnecessary interferences. The sensor output is first passed through a RC high-pass filter (HPF) to separate it from the DC component. The cut-off frequency of the HPF is set to 0.7Hz. Next, an active low-pass filter (LPF) will further process the signal, which is made of an Op-Amp circuit.

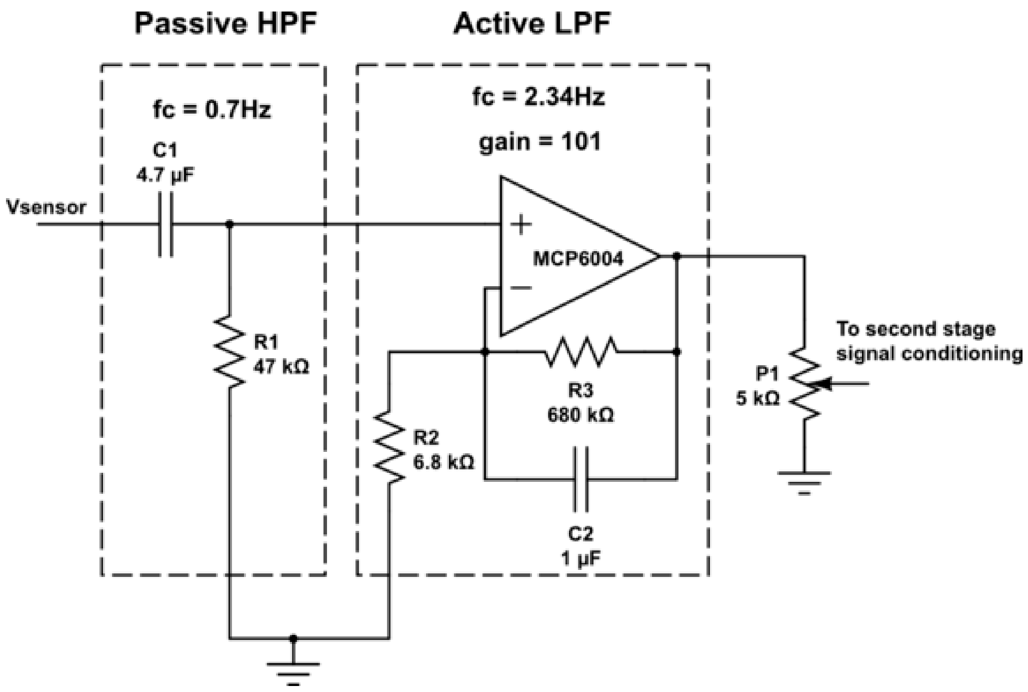
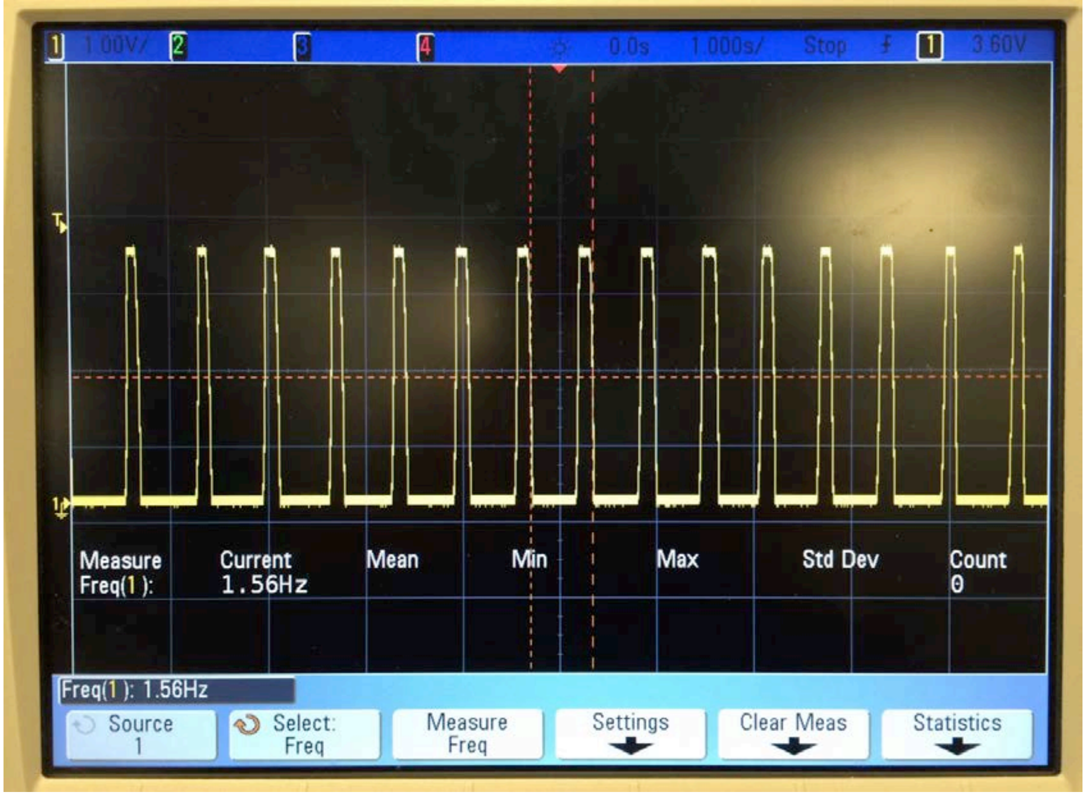


Figure 2.3 The First Stage of Signal conditioning

The gain and the cut-off frequency are set to 101 and 2.34 Hz, respectively. As a consequence, the combination of the HPF and LPF contributes to the removal of unwanted DC signal and high frequency noise, while amplifying the low amplitude pulse signal strength 101 times. The output of the first signal conditioning stage is sent to the next similar stage for further filtering and amplification. Therefore, the total voltage gain achieved from the two cascaded amplification stages is 101 * 101 = 10201. The two filtering and amplification stages convert the input signal to nearly 3V pulses. Figure 2.4 is the output signal on an oscilloscope.

2.3. The Microcontroller Interface

The measured pulse signal yielded from the filter stages is sent to a microcontroller (MCU) for frequency computing and data forwarding. Three main criteria are considered in



Measure Freq(1): Current 1.56Hz Mean Min Max Std Dev Count 0

Freq(1): 1.56Hz

Figure 2.4 The Output Signal of the Signal Conditioning Stage

choosing a proper microcontroller for the NFC Ring: (1) its power consumption should be as low as possible, (2) the computing ability and memory should be sufficient, and (3) the size of the chip should be as small as possible.

Microcontroller chip MSP430G2553 of Texas Instruments is selected as the ring's MCU. It belongs to the Texas Instruments MSP430 family of ultra-low-power microcontrollers. The architecture is specially optimized with five low-power modes to achieve extended battery life in portable measurement applications. A powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency are included in the device. The digitally controlled oscillator (DCO) makes it possible to wake-up the device from low-power modes to active mode in less than 1 μs [28]. The MSP430G2x53 series are ultra-low-power mixed signal microcontrollers having up to 24 I/O capacitive-touch enabled pins, a versatile analog comparator, built-in 16-bit timers, and the universal SCI. There is also a 10-bit analog-to-digital (A/D) converter in the MSP430G2x53 family members. Typical applications feature low-cost sensor systems that capture analog signals, convert them to digital values, and then display the data or transmit the data to a host.

First, the sensor and filter circuits need to be connected to the microcontroller. Figure 2.5 describes the pinout of MSP430G2553. The filtered output signal is sent to a digital I/O. PIN 5 in

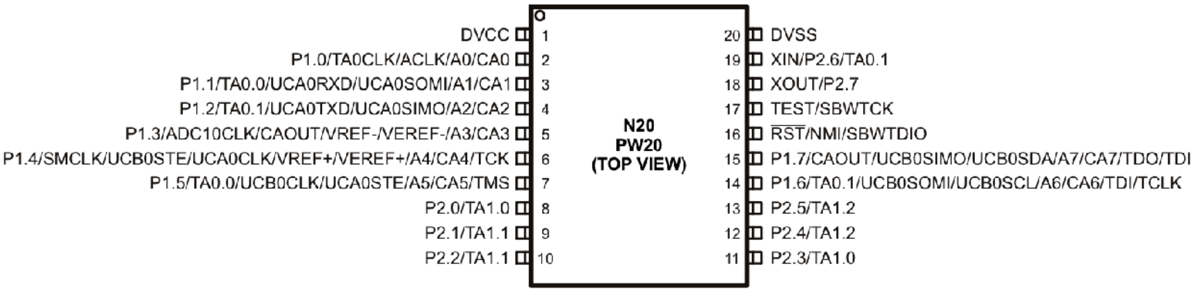

DVCC 1
P1.0/TA0CLK/ACLK/A0/CA0 2
P1.1/TA0.0/UCA0RXD/UCA0SOMI/A1/CA1 3
P1.2/TA0.1/UCA0TXD/UCA0SIMO/A2/CA2 4
P1.3/ADC10CLK/CAOUT/VREF-/VEREF-/A3/CA3 5
P1.4/SMCLK/UCB0STE/UCA0CLK/VREF+/VEREF+/A4/CA4/TCK 6
P1.5/TA0.0/UCB0CLK/UCA0STE/A5/CA5/TMS 7
P2.0/TA1.0 8
P2.1/TA1.1 9
P2.2/TA1.1 10
N20
PW20
(TOP VIEW)
20 DVSS
19 XIN/P2.6/TA0.1
18 XOUT/P2.7
17 TEST/SBWTCK
16 $\overline{\text{RST}}$/NMI/SBWTDIO
15 P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/CA7/TDO/TDI
14 P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6/TDI/TCLK
13 P2.5/TA1.2
12 P2.4/TA1.2
11 P2.3/TA1.0

Figure 2.5 The Device Pinout of MSP430G2x53, 20-Pin Devices [31]

Figure 2.5 is one of the 14 pins that can act as a general-purpose digital I/O pin. Therefore, PIN 5 reads the signal in so that the signal can be processed in microcontroller. Besides processing the pulse signal, the microcontroller also provides power supply to the sensor and filter circuit. As a consequence, PIN 0 and PIN 20 act as the power and ground respectively for the circuit.

2.4. Frequency Calculation

After the hardware connection is correctly built, the microcontroller can read the heart rate signal from PIN 5. The readings can then be leveraged to produce accurate heart rate per minute value. First the library code files of the microcontroller and the NFC transponder are imported: one is msp430.h, the other is RF430.h. These two are the header files for definitions of methods and constants. Common read and write register interfaces are provided in this file, and are implemented in RF430.c. These library code files assist in programming this microcontroller conveniently. The library files are imported as shown in Figure 2.6.

To implement the algorithm for computing frequency, a main.c file is also added to work with the library mentioned above. This file consists of five main parts: import of library files, variables definition, microcontroller initialization, frequency calculation and data forwarding.

Then a few significant global variables for signal processing and NFC communication are defined. The first three variables are used for computing pulse count. The get_new_pulse variable acts as a flag that is set to one when there is new pulse detected on PIN 5. An interrupt method named Port1_ISR listens to all the pins on Port1 that are enabled for interrupt. Whenever

```
#include "msp430.h"
#include "RF430.h"
```

Figure 2.6 Importing Library Files

there is a low to high transition on the input signal, the interrupt method will be triggered and set get_new_pulse to one. Another global variable named pulse_timer_counter is the value of how many times the timer interrupts are triggered between two pulses. Every time the interrupt method runs, it increments pulse_timer_counter by one. When a new pulse appears, the interrupt of PIN 5 is invoked and the value of pulse_timer_counter is passed to pulse_count. Then pulse_timer_counter is reset to zero for the next pulse. The pulse_count is particularly designed for temporarily storing the number of interrupts between two pulses in one calculation so that the calculation algorithm and data forwarding will not interfere with each other. After that, it is processed in the main loop of the microcontroller and serves as the input data of NFC tag. The last two arrays are for NFC data forwarding. These variables are presented in Figure 2.7.

Next, the microcontroller mode and pins are initialized for the two functions. The timer clock is set to 8000000/8192 Hz and the interrupt bit is set enabled as well. Then system clock is set to 8 MHz. After that, Port1 pins are configured for I2C connection with the NFC transponder and PIN 5 is adjusted to input pull-up mode. Other pins in Port1 are configured as output pull-down mode. Interrupt is enabled for PIN 5 to read upper edges in signal. Then USCI configurations for I2C are handled. Software reset and I2C mode are enabled here. Port2 pins are set as default because no pin is used in our application. Last, enable_interrupt() and __bis_SR_register(GIE) methods are called to bring all the settings above into effect.

```
// Heartbeat count
unsigned char get_new_pulse = 0;
unsigned short pulse_timer_counter = 0;
unsigned short pulse_count = 0;
// NFC data array
unsigned char NDEF_Application_Pre_Data[] = RF430_APP_DATA;
unsigned char NDEF_Application_Data[] = RF430_DEFAULT_DATA;
```

Figure 2.7 The Global Variables

```
void setup() {
    WDTCTL = WDTPW + WDTTMSEL + WDTCNTCL + WDTIS0; // watchdog clock
    IE1 |= WDTIE; //interrupt enable
    BCSCTL1 = CALBC1_8MHZ; // Set range
    DCOCTL = CALDCO_8MHZ; // SMCLK = MCLK = DCO = 8 MHz ACLK = LF oscillator
    // Configure pins for P1, I2C
    P1SEL = (unsigned char) (0x00 | BIT6 | BIT7); // Selecting I2C pin function
    P1SEL2 = (unsigned char) (0x00 | BIT6 | BIT7);
    P1OUT = (unsigned char) (0x00 | BIT3); // P1.3 pull up and others pull down
    P1DIR = (unsigned char) (0xFF & (~BIT3)); // P1.3 input and others output
    P1REN = (unsigned char) (0xFF & (~BIT0) & (~BIT6) & (~BIT7)); // P1.0 pull
up/down resistor disable for LED output, and P1.6 P1.7 disable for I2C.
    // Configure the P1 interrupt: Enable the P1.3 interrupt for signal input
    P1IE = (unsigned char) (0x00);
    P1IFG &= ~(BIT3); // Clear interrupt flag
    P1IES &= ~(BIT3); // Set interrupt trigger as low-to-high transition
    // configure USCI for I2C
    UCB0CTL1 |= UCSWRST; // Software reset enabled
    UCB0CTL0 |= UCMODE_3 + UCMST + UCSYNC; // I2C mode, Master mode, sync
    UCB0CTL1 |= UCSSEL_3 + UCTR; // SMCLK = 8MHz, transmitter
    UCB0BR0 = 80; // Baudrate = SMLK/80 = 100kHz
    UCB0I2CSA = 0x0028; // slave address - determined by pins E0, E1, and E2 on the
RF430CL330H
    UCB0CTL1 &= ~UCSWRST; // software reset released
    // Configure pins for P2
    P2SEL = (unsigned char) (0x00);
    P2SEL2 = (unsigned char) (0x00);
    P2OUT = (unsigned char) (0x00 | BIT4);
    P2DIR = (unsigned char) (0xFF & ~(BIT4));
    P2REN = (unsigned char) (0xFF);
    // Configure the P2 interrupt:
    P2IE = (unsigned char) (0x00);
    _enable_interrupt();
    __bis_SR_register(GIE);
    // Initialize
    __delay_cycles(1000);
    P2OUT |= BIT0; // Release the RF430CL330H from Reset
    Write_Register(CONTROL_REG, SW_RESET);
    __delay_cycles(4000000); // initialize launchpad should be 20ms or greater
    while (!(Read_Register(STATUS_REG) & READY));
}
```

Figure 2.8 The Setting Up Function

__enable_interrupt() enables interrupts by inserting the EINT instruction. EINT is an emulated instruction that is equivalent to BIS #8,SR. __bis_SR_register(GIE) is an intrinsic that is also equivalent to BIS #8,SR. The main difference between the two constructions is that __bis_SR_register() allows to set more than just the GIE bit, while __enable_interrupt() only sets GIE bit [29]. Finally, four seconds are set aside to initialize the NFC module and wait until it is ready. The setup function is displayed in Figure 2.8.

The next stage is frequency calculation that is composed of two parts. The first part is to get the count of interrupts invoked during two pulses. The second part is to recover the value of heart rate per minute from the counts. The ring is only responsible for the first part.

The interrupt of a timer and PIN 5 are utilized to get the count. The process is clear: after all settings are initialized, every time a pulse invokes an interrupt at PIN 5, the count is stored to another variable while itself is reset to 0 for the next interval. Before the next pulse is detected, the timer interrupt will be triggered many times. Each time the count is added by one so that the total number of counts can be derived when the next low-to-high transition appears. To sum up, the interval between two pulses should be approximately equal to the total interval of that many timer interrupts. Equation 2.1 expresses this relation:

$$count * \frac{1}{\frac{8000000}{8192}} = 1 * T_{pulse} \qquad \text{(Eq. 2.1)}$$

With Equation 2.1, T_{pulse}, the period of the pulse signal, can be obtained. This is the data to be forwarded to the smartphone. A pulse costs T_{pulse} seconds, so $\frac{60}{T_{pulse}}$ represents the number of pulses within 60 seconds, which is exactly one minute. Equation 2.2 is the final equation between the processed output and count.

$$HPM = 60 * \frac{8000000}{count * 8192} \qquad \text{(Eq. 2.2)}$$

The reason why the count is not set directly in the ring with the MCU before sending is as follows. If the calculation is carried out on the MCU, it adds burden to the MCU and increases the power consumption of the NFC Ring. Due to the fact that there is no display requirement for the NFC Ring, shifting this calculation to the smartphone and then recovering it is a better

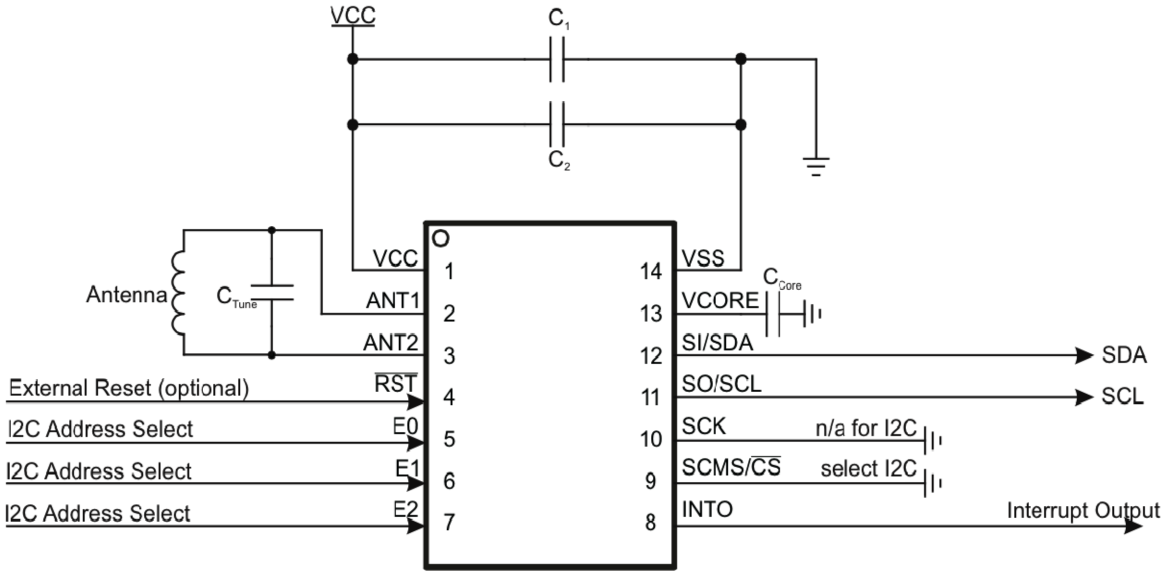


Figure 2.9 The RF430CL330H I2C Operation Diagram [26]

solution. This operation conforms to the low-energy and simple design of hardware and removes unnecessary calculations from the ring.

2.5. The Operations on The Transponder

Eventually, the data to be sent to a smartphone is generated by the algorithm in Section 2.4 and acts as the input of the last stage. First of all, the transponder should connect to the microcontroller correctly. In Section 2.4, it is mentioned that I2C protocol is selected in initialization stage for the two chips. Figure 2.9 presents the diagram for I2C operation of RF430CL330H.

According to the diagram, the connection between microcontroller and transponder can be built. C_1 and C_2 are 0.1 μF and 1 μF, respectively, which serve as decoupling capacitors on V_{cc}. The most significant pins are PIN 11 and PIN 12. PIN 11 is the SO/SCL terminal for the I2C clock. PIN 12 is the SI/SDA terminal for the I2C data. These two pins are routed to PIN 14 and PIN 15 separately.

Once the I2C channel between NFC transponder and MCU is correctly set up, the program can send the total number of timer interrupts to the NFC tag. First, two types of data are defined according to NDEF specifications. NDEF is a lightweight binary format used to encapsulate typed data and defines messages and records. It is composed of two main parts: the capability container and NDEF file [30]. To build an NDEF message, NDEF Application Data is selected by the name D2_7600_0085_0101h. Then the capability container is further chosen through its ID E103h. Parameters including the size of this capability container, mapping version, maximum R-APDU data size, maximum C-APDU data size, and NDEF file control type length value (TLV) should all be defined. The NDEF file control TLV is mandatory as well. The tag, length, file identifier, max NDEF size, read and write access condition are defined in it. Next, the NDEF file can be set through the file ID as well. It should be different from E103h while the same as the file identifier in control TLV field, E104h. The NDEF file contains the actual message to be written to the tag. It consists of the length of NDEF record data, the binary NDEF file content, and the unused bits if length is smaller than maximum length supported. Table 2.1 describes the structure of an NDEF application data.

Table 2.1 The NDEF Application Data

NDEF Application Selectable by Name = D2_7600_0085_0101h	Capability Container Selectable by File ID = E103h	2 Bits - CCLen		
		1 Bit - Mapping version		
		2 Bits - MLe = 000F9h		
		2 Bits - MLc = 000F6h		
		NDEF File Ctrl TLV	1 Bit - Tag = 04h	
			1 Bit - Len = 06h	
			6 Bits - Val	2 Bits - File Identifier
				2 Bits - Max file size
				1 Bit - Read access
				1 Bit - Write access
	NDEF File Selectable by File ID = xxyyh	2 Bits - Len		
		x Bits - Binary NDEF file content		
		y Bits - Unused if Len < Max file size in File Ctrl TLV		

```
#define RF430_APP_DATA {
\
/*NDEF Tag Application Name*/ \
0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01, \
\
/*Capability Container ID*/ \
0xE1, 0x03, \
0x00, 0x0F, /* CCLEN */ \
0x20,       /* Mapping version 2.0 */ \
0x00, 0xF9, /* MLe (49 bytes); Maximum R-APDU data size */ \
0x00, 0xF6, /* MLc (52 bytes); Maximum C-APDU data size */ \
0x04,       /* Tag, File Control TLV (4 = NDEF file) */ \
0x06,       /* Length, File Control TLV (6 = 6 bytes of data for this tag) */ \
0xE1, 0x04, /* File Identifier */ \
0x0B, 0xDF, /* Max NDEF size (3037 bytes of useable memory) */ \
0x00,       /* NDEF file read access condition */ \
0x00,       /* NDEF file write access condition */ \
\
/* NDEF File ID */ \
0xE1, 0x04, \
\
/* NDEF File for Hello World */ \
0x00, 0x1F, /* NLEN: NDEF length (20 byte long message, max length) */ \
\
/* NDEF Record (refer to NFC Data Exchange Format specifications)*/ \
0xD1,       /* MB(Message Begin), SR(Short Record) flags set, ME(Message End),
IL(ID length field present) flags cleared; TNF(3bits) = 1; */ \
0x01, 0x1B, /* Type Length = 0x01; Payload Length = 0x10 */ \
0x55,       /* Type = U (text) */ \
0x03,       /* http://www. */ \
\
/*me.freetymekiyan.smartring*/ \
'm','e','.','f','r','e','e','t','y','m','e','k','i','y','a','n','.','s','m','a','r'
,'t','r','i','n','g' \
} /* End of data */
```

Figure 2.10 Example Data Definition File

Two types of NDEF message are defined on the basis of the structure discussed previously. One is for bringing the APP to the front when it is not running on user’s smartphone, and the other is for forwarding heart rate data. Besides the data and data type, these two messages share most of the settings. The trigger message is an URL with the package name of the APP. On the other hand, the data message is pure text with four digits representing the pulse count. An example definition of trigger message is shown in Figure 2.10.

After the two NDEF messages are defined, the messages can be loaded to two arrays in main file, namely NDEF_Application_Pre_Data and NDEF_Application_Data. The data

```
if (pulse_count > 5000) { // range check
    pulse_count = 0;
} else if (get_new_pulse) {
    get_new_pulse = 0;
    unsigned short data = pulse_count;
    Write_Register(CONTROL_REG, SW_RESET);
    __delay_cycles(4000000);
    NDEF_Application_Data[35] = (data / 1000) + 0x30; // get 4 digits
    NDEF_Application_Data[36] = ((data % 1000) / 100) + 0x30;
    NDEF_Application_Data[37] = ((data % 100) / 10) + 0x30;
    NDEF_Application_Data[38] = (data % 10) + 0x30;
    Write_Continuous(0, NDEF_Application_Data, DATA_LENGTH); // write to tag
    Write_Register(CONTROL_REG, RF_ENABLE);
    __delay_cycles(4000000);
} else { // no new pulse
    __delay_cycles(1000000);
}
```

Figure 2.11 Example of Writing Data to NFC Module

forwarding begins after initialization is done. The MCU first enables its RF interface and writes trigger message to NFC tag every four seconds, and waits for a smartphone detected in range. When a smartphone is detected, the RF_BUSY flag will update to one in status register. Then the most recent pulse_timer_counter value is checked to prevent error. If it is over 5000, the measured value may be too high and will be discarded. If it is in normal range, whether there is a new pulse appeared is checked. The data will be converted to four digits with each digit represented by a character, and then written to NDEF application data's payload. The payload is then processed by the transponder so that it can be received successfully by a smartphone. Figure 2.11 provides example code to implement the logic.

3. NFC ANTENNA DESIGN

In order to transmit information from the NFC Ring to a smartphone, a customized antenna is designed to fit in the ring and work with the integrated circuit inside RF430CL330H. The Texas Instruments RF430CL330H Dynamic NFC Interface Transponder is a NFC Forum Type 4B Tag Platform operating at 13.56MHz. The device provides the flexibility to be used in combination with various antennas to meet the application performance requirements. The antenna should also be compatible with the NFC protocol to achieve a resonance frequency at 13.56MHz. This section describes the procedures to design an antenna for RF430CL330H.

3.1. The RF Interface Circuits

As shown in Figure 1.1 the functional block diagram of RF430CL330H, there is a RF module in the transponder. The RF Communication Interface is built on the ISO14443B specifications. It supports data rates from 106 kbps up to 848 kbps. The device is compliant with the NFC Tag Type 4B platform. With the antenna connection, an interface to the outside world is provided to the NFC Ring. There are two pins connected to the external antenna: ANT1 and

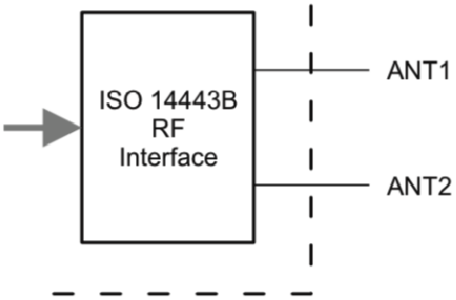


Figure 3.1 The RF Interface Module [26]

ANT2. Figure 3.1 shows the RF Interface module separately. The two pins, ANT1 and ANT2 in Figure 3.1, are Pin 2 and Pin 3 on the NFC transponder. The antenna is connected to these two pins, while being parallel to the internal resonance capacitor C_{int}. An additional external capacitor parallel to C_{ext} can be added in antenna design depending on the antenna inductance,.

3.2. The Inductance Coil

In fact, the NFC antenna is not the regular antenna used in most electronic devices. The broad goal of antenna design is to obtain a usable radiating structure by the coupling effect between two inductors [23]. A current will be induced to the second inductor when the magnetic field from the first inductor passes through it. As a result, the structure satisfies both the NFC transmission range and the requirement of contactless energy transfer.

Shape and size of the inductor are two main constraints to be taken into consideration in design. Firstly, the normal inductance range of an NFC antenna should be from 300 nH to 3 μH [24]. Although the upper limit is 3 μH approximately, the antenna inductance should be as large as possible, thus enable better interaction between itself and the other antenna in the smartphone. Normal shapes for antenna are circular, square, or rectangle. A rectangular inductor is chosen in this case to spread fully inside the ring. For this reason, the width and height cannot be too large. However, the antenna can be relatively smaller than the ring. The height of antenna can be the same as the width of ring and the width of antenna smaller than the diameter of a finger to fit in a ring shell.

The final dimension of the inductor is 1.6 cm in width (w) and 1.0 cm in height (h) according to previous discussion. The number of turns (N) is 5. The wire radius (a) is 10 mil, which is 0.0254 cm approximately. Copper wire is used on the PCB to build the antenna, so the relative permeability of the medium (μ_r) is 0.999994, almost equal to one [25]. Equation 3.1 provides a formula for rectangle antenna inductance calculation [26]:

$$L_{rect} = \frac{N^2 u_0 u_r}{\pi}\left[-2(w+h) + 2\sqrt{h^2+w^2} - h\ln\left(\frac{h+\sqrt{h^2+w^2}}{w}\right) - w\ln\left(\frac{w+\sqrt{h^2+w^2}}{h}\right) + h\ln\frac{2h}{a} + w\ln\frac{2w}{a}\right]$$

(Eq. 3.1)

With all the parameters provided, the inductance value of the customized antenna is 813 nH and this occurs within the recommended range. Figure 3.2 is the antenna inductance coil.

3.3. The External Capacitor

As shown in Figure 3.3, an on-chip resonance capacitor is placed inside RF430CL330H chip that has a typical value of 35 pF. A resonance circuit is built upon the external antenna and the on-chip resonance capacitor. An additional external resonance capacitor is added in the circuit to allow lower inductance antenna coils in this case. The resonance frequency is

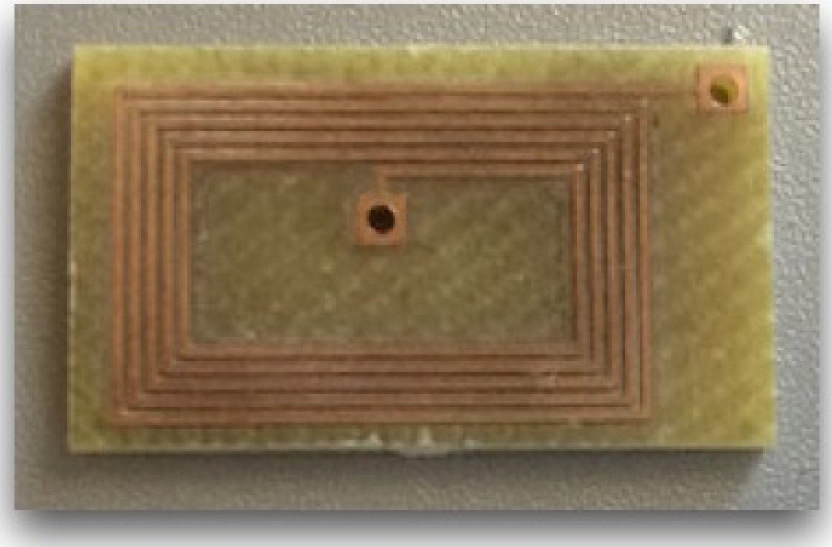


Figure 3.2 The Antenna Inductance Coil

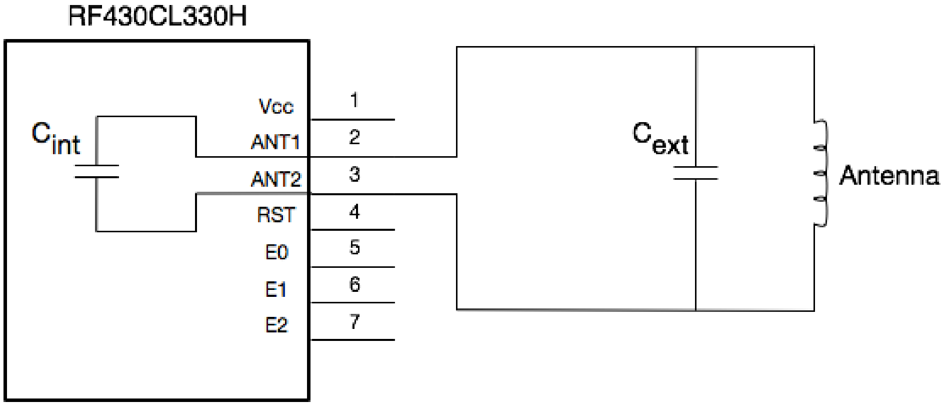


Figure 3.3 The Circuit of the NFC Transponder and Its Antenna

calculated using the formula in Equation 3.2:

$$f_{res} = \frac{1}{2 * \pi * \sqrt{L * C}} \qquad \text{(Eq. 3.2)}$$

Typical resonance frequency for NFC is 13.56 MHz. In practical design, recommended operating resonance frequency is approximately 13.7 MHz for optimum performance. Resonance frequencies greater than 13.7 MHz are regarded as too large and would lead to performance reduction. It is recommended to ensure that the resonance frequency, including all tolerances, stays above 13.56 MHz [26].

With antenna inductance L equal to 813 nH and resonance frequency f_{res} equal to 13.7 MHz in the formula, the total resonance capacitance is 166 pF. The theoretical value of external capacitance should be the difference of 166 pF and 35 pF, which is 131 pF.

The above calculations are not exactly accurate due to the possible tolerance of each part in the circuit. A variable capacitor for tuning is added in the circuit to compensate the tolerance and achieve best performance. This capacitor is also placed in parallel with the internal capacitor.

According to the theoretical value above, variable capacitor GZC15100 from Sprague-Goodman is chosen, and it has a capacitance range from 10 to 150 pF [27].

3.4. The Antenna Q Factor

To determine the frequency selectivity of our customized antenna, an experiment is conducted with a spectrum analyzer to measure the antenna’s Q factor. The Q factor can be calculated with the formula shown in Equation 3.3:

$$Q = \frac{f_{res}}{BW} \qquad \text{(Eq. 3.3)}$$

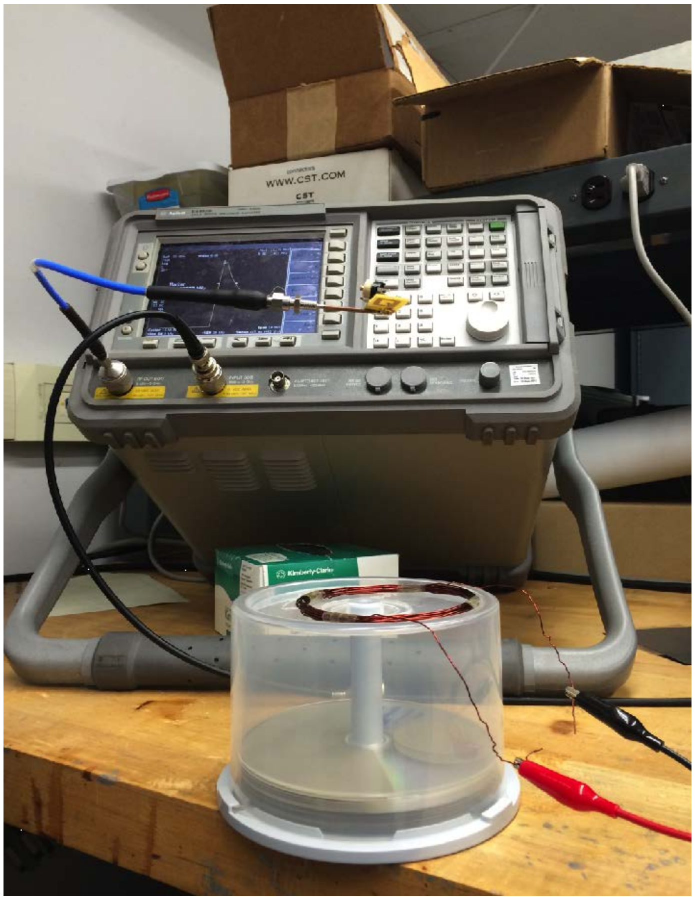


Figure 3.4 The Frequency and Bandwidth Measurement Experiment

The resonance frequency and bandwidth should be captured using a spectrum analyzer with a tracking generator. In our experiment, an Agilent E4402B spectrum analyzer is used. The test fixture should consist of a pickup coil connected to the input of the spectrum analyzer and the antenna connected to the output of the spectrum analyzer tracking generator as shown in Figure 3.4.

The steps to setup the spectrum analyzer are as follows:

(1) Connect the test fixture with the spectrum analyzer. The blue transmission line connects the inductance coil and the tuning circuit to the output of the tracking generator. The other connector joins a pickup copper coil with the input channel.

(2) Turn on the tracking generator output and set the output center to 13.56 MHz with a span of 10 MHz.

(3) Set resolution bandwidth to 9 kHz and video bandwidth to 30 kHz.

(4) Enable source power to 3 dBm and set attenuation to 0 dBm.

(5) At approximately -55 dBm reference level with a vertical scale of 1 dB/div, the resonance curve will be shown as in Figure 3.5.

(6) View the bandwidth by enabling the N dB three points through the peak search menu.

In an actual experiment, the span, reference level and vertical scale can be different according to the antenna and distance between antenna and pickup coil. These parameters should be adjusted based on the curve shown in spectrum analyzer.

The Q factor is calculated by dividing the resonant frequency by the measured bandwidth. From Figure 3.4 f_{res} is 13.135 MHz and BW is 897.8 kHz. Hence the Q factor is approximately 14.63. Typical Q values for the RF430CL330H range from approximately 30 to

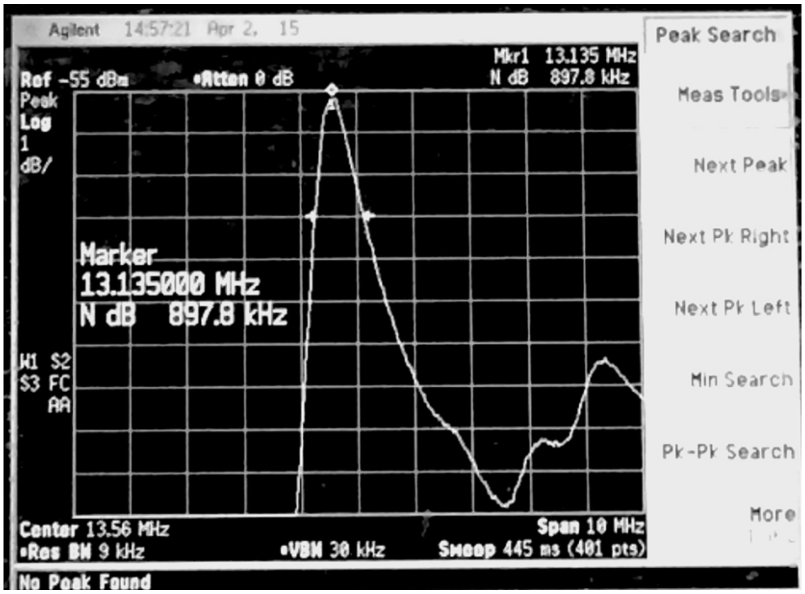


Figure 3.5 The Frequency Response of the Antenna on a Spectrum Analyzer

40. This indicates that antennas with small size at this level may encounter decrease in performance, such as narrowing down of transmission range or unstable connection.

4. THE SMARTRING ANDROID APPLICATION

In order to lower the power consumption of the ring, its functions are limited to merely data acquisition and forwarding. Other tasks, such as data storage and calculation, are handed over to the application on an Android phone. Due to the fact that almost everyone has one smartphone, it can serve as a powerful data computation unit to connect all other wearable sensors, whether it is a ring or a wristband. Therefore, the other wearables do not need to implement their own screen or user interface, and hence have a lower energy cost. An Android application called SmartRing was developed to receive data transmitted by the NFC module in the ring in order to store it, display it with a readable visual chart, and generate analysis reports according to the user's personal information.

4.1. The SmartRing Application Architecture

The basic architecture of the SmartRing app is shown in the Figure 4.1. There are three main modules - measurement, analysis and settings. The measure module is the core module to receive data forwarded by the NFC Ring from Android NFC module. The data is used in an

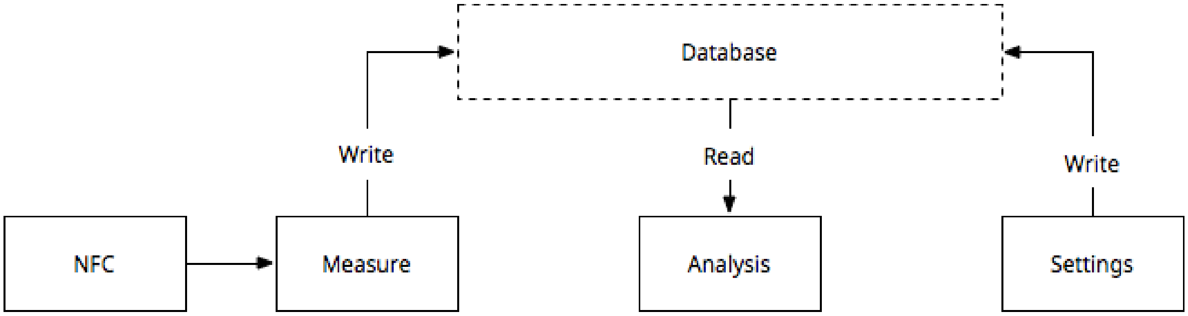


Figure 4.1 The SmartRing APP Architecture

algorithm to further compute an accurate value for user’s current heart rate. After the calculation is done, the user needs to choose his/her current workout state to save the result. Finally, the data will be inserted into the database.

When the user wants to check the analyzed data, the analysis module can read raw data from the database and render them into graph and table. The application will first read all measurements from the past seven days, including both rest and active states, and calculate their average daily values respectively. Then a graph showing average value of each day can be rendered. The table will read the user’s personal profile, which includes items like age and gender, to provide recommended heart rate ranges for various workout intensities.

The other important module is the settings module that stores the user’s profile and the preferences of the application. Basic user information can be set or updated in this page, such as the user’s name, gender, email, age, and weight. Also, the user can set a reminder of different frequency, daily, weekly or monthly, and at any time so that there will be a notification.

Figure 4.2 describes the workflow of this APP. The first main entry point of the APP is listening to NFC intent. After the APP is installed on an Android device, it checks whether there is an NFC module in the phone system and monitors the data in the NFC module in the Android device. Whenever the NFC Ring is within range of Android NFC module, it is automatically detected and sends out an encoded data to open the APP. After being triggered, the APP starts measuring to receive consecutive data from the ring and utilize all the data to generate a more precise result. If current measurement is not done, the APP will continue reading new data from NFC module. Once it is finished, the user will get the result and choose his/her current exercising state, either resting or working out. Adding this categorization to result will help categorize heart rate measurements and insert the new data to the database. If the measurement is not accurate or reasonable, the user can also discard it according to his/her own will.

The other one of the two main entrances is the normal entrance, which is started by user operation, when the user clicks the icon on the desktop launcher or the catalog of all APPs on device. Measure module will first show and by clicking the phone image in the middle, a new measuring process will start. By clicking it again the process can be stopped immediately. Due to

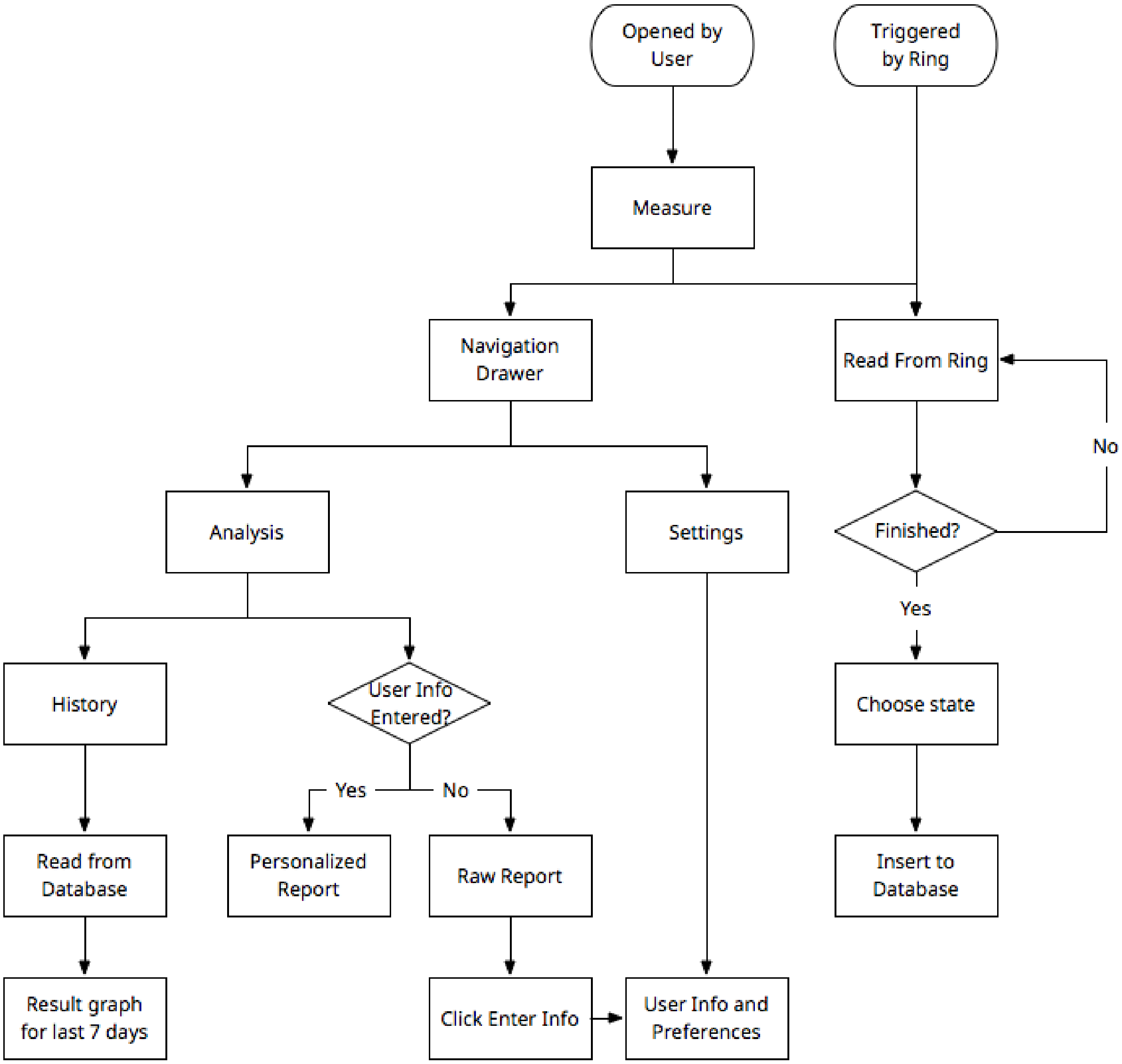


Figure 4.2 The Flowchart of the SmartRing APP

how the algorithm is defined, if the measurement is not completed, no result will show because the data received is not enough.

To see previous measurements and reports, the navigation drawer needs to be opened to access the analysis page. In this module, two kinds of analysis are offered to the user: one is a categorized bar graph showing the average heart rate of the last 7 days, the other is a table report of heart rate ranges and relative exercising levels. The application reads recent raw measurement data from the database first and processes them into the desired data structure. Data of the same type and same day will be calculated to gain an average value of that day. Each day will have two bars in the graph: one is for the rest state, which means the user is in a normal condition and not doing any workout; the other is for active state, which means the user is doing an exercise. The table report is different according to whether the user has already entered basic information, mainly gender and age. If there is not any required user information, the table will present only a percentage range of workout intensities and a button for switching to the settings page directly to set the user's age and weight. Otherwise, it will show the actual value predicted from the provided information.

The settings page is also accessible from the navigation drawer. It provides interfaces for the user to enter or update their personal profile and set up a friendly notification. The profile consists of the user name, gender, email address, age and weight. Gender, age and weight are vital for the personalization of history and reports. Arranging a system notification requires its frequency and the time it triggers. The user may choose anytime on a daily, weekly, and monthly basis.

4.2. Application Functions

In this section, the procedure of how to receive data from the NFC, the database design and how to produce personalized analysis based on user profile are explained.

4.2.1. Receiving Data from NFC

To develop NFC functions, the first step is to confirm that the Android phone supports NFC and the NFC module has been turned on. Some devices do not have the NFC module. Some devices with earlier Android system versions do not support NFC at software level, even if it has NFC hardware.

Before accessing the NFC hardware of an Android phone, it is necessary to add the following permissions in the project’s manifest file as shown in Figure 4.3 [18]:

- The NFC *<uses-permission>* item to access NFC hardware.
- The minimum SDK version number that the application can support.
- The *uses-feature* item so that the SmartRing application shows up in Google Play only for devices that have NFC hardware.

After adding the above permissions, the NFC module will be accessible in the application. To receive data from the NFC module, the SmartRing APP has to register for NFC

```
<uses-permission android:name="android.permission.NFC" />

<uses-sdk android:minSdkVersion=“10”/>

<uses-feature android:name=“android.hardware.nfc" android:required="true" />
```

Figure 4.3 Example Permissions in AndroidManifest.xml File

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data
         android:host="me.freetymekiyan.smartring"
         android:scheme="http" />
</intent-filter>
```

Figure 4.4 Example of Intent Filter

intents. There are three types of the NFC intents in Android: ACTION_NDEF_DISCOVERED, ACTION_TECH_DISCOVERED, and ACTION_TAG_DISCOVERED. The ACTION_NDEF_DISCOVERED intent covers most of the common NFC protocols. The ACTION_TECH_DISCOVERED intent needs an XML file to define NFC technology supported. The ACTION_TAG_DISCOVERED intent is the most fundamental intent that captures all NFC-related activities. ACTION_NDEF_DISCOVERED intent is adopted in this case for the NFC tag that is used to build the ring.

To filter ACTION_NDEF_DISCOVERED intents, the intent filter and the type of data should be specified in manifest file as well. The data type is an http address with the package name as host name. This data is predefined with hardware, so that when the ring sends a relative message, it will execute the application immediately. The details of the intent filter are presented in Figure 4.4.

After the setup steps are finished in the manifest file, the NFC module can be controlled

```
// Check for available NFC Adapter
mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
if (mNfcAdapter == null) {
    Toast.makeText(this, "NFC is not available", Toast.LENGTH_LONG).show();
    finish();
    return;
}
// Register callback
mNfcAdapter.setNdefPushMessageCallback(this, this);
```

Figure 4.5 Example of NFCAdapter Class

```
@Override
public void onResume() {
    super.onResume();
    // Check to see that the Activity started due to an Android Beam
    if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
        processIntent(getIntent());
    }
}
```

Figure 4.6 Example of onResume() Method

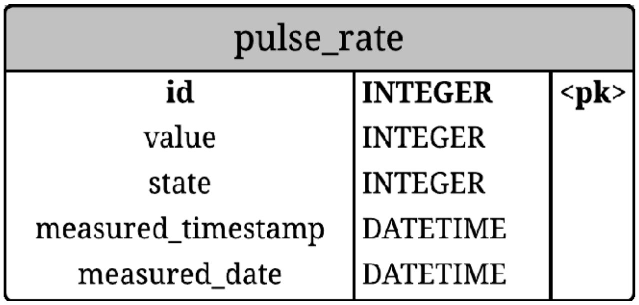
in the program. NFCAdapter class is imported to operate on the received NFC messages. With getDefaultAdapter() method in this class, a quick check on whether NFC is supported by this phone can be added. If the adapter exists, a callback for NDEF message will be registered in the onCreate method of the Activity. Figure 4.5 contains an example on the check and register process.

Then in onResume() method, the action of intent is examined to confirm that it is from ACTION_NDEF_DISCOVERED. After that, the intent is passed onto processIntent() method to extract NFC messages as shown in Figure 4.6.

The core step to receive NFC intents in running APP is to override onNewIntent()method [19]. This method executes whenever there is a new intent, including NFC intents. Therefore, when there is a new NFC intent, it can be set to the activity's intent. The getIntent() method is guaranteed to return the latest NFC intent in this way.

```
void processIntent(Intent intent) {
    Parcelable[] rawMsgs =
intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
    // only 1 message sent during the beam
    NdefMessage msgs = (NdefMessage) rawMsgs[0];
    // record 0 contains the MIME type, record 1 is the Android Application Record,
if present
    String msg = new String(msgs.getRecords()[0].getPayload());
    // TODO update data and view
}
```

Figure 4.7 Example of processIntent() Method



pulse_rate		
id	**INTEGER**	**<pk>**
value	INTEGER	
state	INTEGER	
measured_timestamp	DATETIME	
measured_date	DATETIME	

Figure 4.8 The Pulse Rate Table Design

A method named processIntent() is built for parsing the intent as presented in Figure 4.7. The intent contains a Parcelable array for NDEF messages. If only one message is sent, it will be the first element in the Parcelable array. The payload of the first record of that NDEF message can be converted to a String for further usage.

4.2.2. Database Design

In the SmartRing Android APP, data transmitted from the NFC Ring is received and stored in a database. The default SQLite database provided in Android system is used. A simple data table is also designed to fulfill the requirements.

The main requirement is from the 7-day average history section bar graph. In this graph, each bar section is grouped by day and each day can be an aggregation of multiple measurements. For each bar in a section, the values are also categorized by two different states. Hence, a wrapper class is built for each measurement, adding an integer for states and a

```
SELECT measured_date, AVG(value) AS avg_value
    FROM pulse_rate
    WHERE state = 0
    GROUP BY measured_date
    HAVING measured_date >= DATE(‘now’, ‘-7 days’)
    ORDER BY measured_date
```

Figure 4.9 The Example SQL Sentence to Generate History Graph

timestamp to specify the measurement time. The final design of the pulse rate table is shown in Figure 4.8. The pulse rate of each measurement consists of an integer id, an integer value, an integer state, a timestamp and a date for the measurement.

To retrieve the information in the database for the graph, the SQLite SELECT statement is used with specific conditions. The graph only needs two fields: measured_date and average value of all measurements of the same state on that day. The return value is also limited to the last seven days. For example, the final SQL query for rest state for the last week is as shown in Figure 4.9.

Android system provides an SQLiteOpenHelper class for encapsulation of SQLite

```
public List<Pulse> getLast7Days(Pulse.State s) {
    List<Pulse> res = new ArrayList<Pulse>();
    SQLiteDatabase db = getReadableDatabase();

    String[] projection = {
            PulseEntry.COL_NAME_MEASURED_DATE,
            "AVG(" + PulseEntry.COL_NAME_VAL + ")" + " AS " +
PulseEntry.COL_NAME_AVG_VAL,
    };
    String selection = PulseEntry.COL_NAME_STATE + "=?";
    String[] selectionArgs = new String[]{s.ordinal() + ""};
    String groupBy = PulseEntry.COL_NAME_MEASURED_DATE;
    String having = PulseEntry.COL_NAME_MEASURED_DATE
            + " >= DATE('now', '-7 days')";
    String orderBy = PulseEntry.COL_NAME_MEASURED_DATE + " ASC";

    Cursor c = db.query(PulseEntry.TABLE_NAME, projection, selection,
selectionArgs, groupBy, having, orderBy);
    c.moveToFirst();
    while (!c.isAfterLast()) {
        String date =
c.getString(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_MEASURED_DATE));
        int value = c.getInt(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_AVG_VAL));
        Pulse p = new Pulse(value, date, Pulse.State.REST);
        res.add(p);
        c.moveToNext();
    }
    if (c != null && !c.isClosed()) {
        c.close();
    }
    return res;
}
```

Figure 4.10 The Java Code for Getting the Values of Last Seven Days

database operations [21]. To build an equivalent query as the above statement, the readable SQLiteDatabase instance is initialized first and query methods are called on it. It will return a cursor of returned table. Then the cursor is moved row by row for each value in that table row to generate the result for history graph. The equivalent implementation in Android is presented in Figure 4.10.

4.2.3. Personalized Analysis

Besides showing the recent history graph of the user, the SmartRing APP provides personalized analysis based on the user's profile as well. The analysis is composed of two parts. The first one is the reference line in the history graph that indicates maximum heart rate. This highest value should not be surpassed no matter how intense the workout is. The second one is the report table of various levels of intensity and their relative heart rate ranges. These ranges are percentages of the maximum heart rate value.

The formulas for max heart rate of male and female employed in this APP are different. A 2002 study [21] of 43 different formulas for max heart rate published concluded that the least objectionable formula was:

$$MHR = 205.8 - 0.685 * age \qquad \text{(Eq. 4.1)}$$

This had a standard deviation that, although large (6.4 beats per minute), was considered acceptable for prescribing exercise training heart rate ranges. Another study in 2010 conducted at Northwestern University by Martha Gulati, et al. suggested a maximum heart rate formula especially for woman [22]:

$$MHR = 206 - 0.88 * age \qquad \text{(Eq. 4.2)}$$

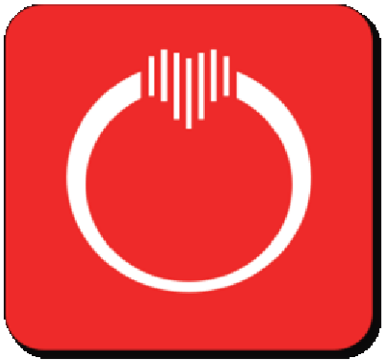


Figure 4.11 The Icon Design

In conclusion, for the max heart rate of male user Equation 4.1 is used. To provide more concise reference for different genders, Equation 4.2 is used for female users.

4.3. The SmartRing APP Interfaces

In this section, the user interface design of the SmartRing Android APP is presented, including the APP icon, three main views, and notification bar with both screenshots and explanations of supported actions.

Figure 4.11 is the icon of the APP, which is composed of a heart, vertical rectangles and a ring. The heart and the background red color indicate that this APP is a tool related to the heart. The rectangles that compose the heart is a symbol of data analysis, more specifically, the section bar graph for measurement history. Those bars also add a sense of technology to the icon. Finally, the ring shape conveys the idea that heart rate measurement is implemented with the NFC Ring hardware. Therefore, this icon combines all vital concepts and functions of the system we build: heart rate, ring and analysis. Figure 4.12 shows the icon in the Android System Launcher as well.

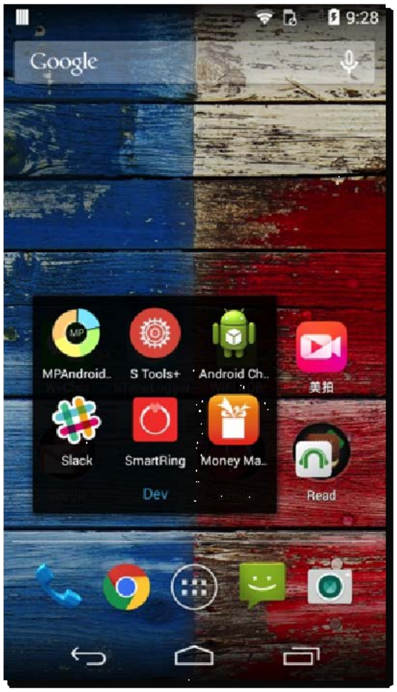


Figure 4.12 The Icon in Android System Launcher

The first activity is Measure Activity that contains full measure process as shown in Figure 4.13. An activity in Android APP means a user interface. In order to keep it simple and help the user focus, only one button is put in the center for starting measurement. Once the button is clicked, a toast (the round bar below in Figure 4.13) will pop up with “Reading pulse rate…” message, and the background of the center button will also start to ripple to indicate that the phone is now searching for the ring. When the ring is within range, it will forward the latest data to the phone, and the received value will be updated in a toast message. During the process, if the button is clicked again, it will stop current measurement and ignore all previous data. After the process is done, a dialog will be presented in the middle with the heart rate. The user is able to choose which exercising state he/she is in right now or cancel if the value is not reasonable enough. To open other activities, the user can either click on the top-left button with three horizontal rectangles or slide from the left edge of the screen to right. The navigation drawer will appear from the left side.

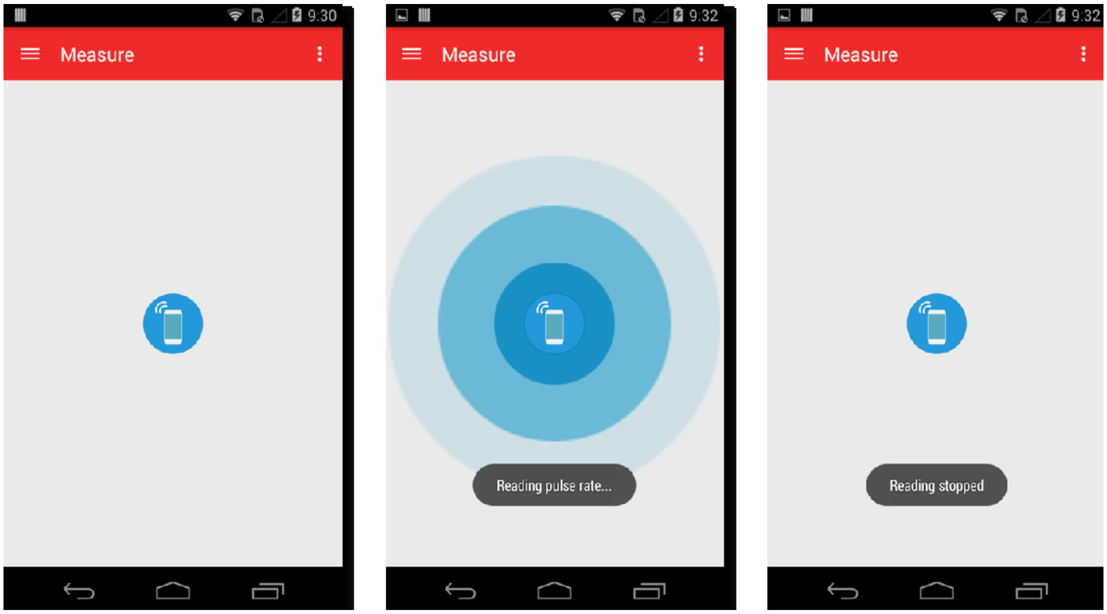


Figure 4.13 The Measuring Activity

Figure 4.14 is the user interface of navigation drawer. The navigation drawer is initially hidden. After being called out, current activity will be covered with a transparent gray image. The top part of the list shows the basic information on the user: profile, name and email. Below

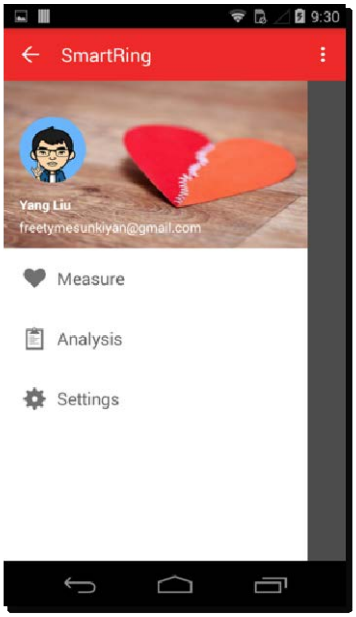
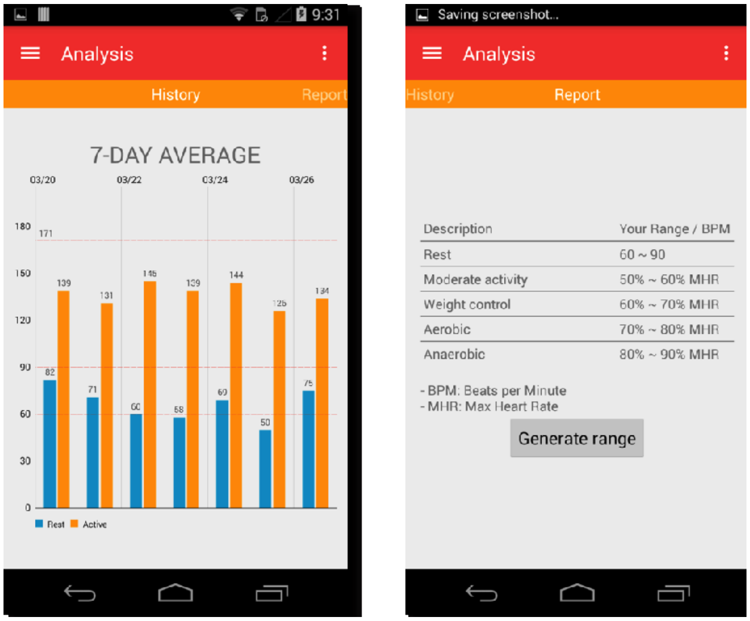


Figure 4.14 The Navigation Drawer

are all of the three activities provided in this APP. Clicking on any of them will open the specified activity.

If a user clicks the second item in the list, the Analysis Activity will show and the navigation drawer will hide again. The history graph shows first. An orange horizontal page indicator with title is added to the above. The indicator will inform the user of the current page and another page further right. The sectioned bar graph is placed under the indicator. The average heart rate value of different states measured each of the last seven days are shown in the graph. The x-axis is the date and the y-axis is the heart rate value. There are also two legends below on the left indicating two different colors: blue for rest state, orange for active state. To provide the user with an intuitive sense beside the value, three red dash reference lines are added. The highest line is 90% of user’s max heart rate, which is computed based on the gender and age saved in the APP. The other two lines form a range from 60 to 90 that is the common range for rest state. This graph also supports double tap and drag/zoom gestures. For example, if the value



Description	Your Range / BPM
Rest	60 ~ 90
Moderate activity	50% ~ 60% MHR
Weight control	60% ~ 70% MHR
Aerobic	70% ~ 80% MHR
Anaerobic	80% ~ 90% MHR

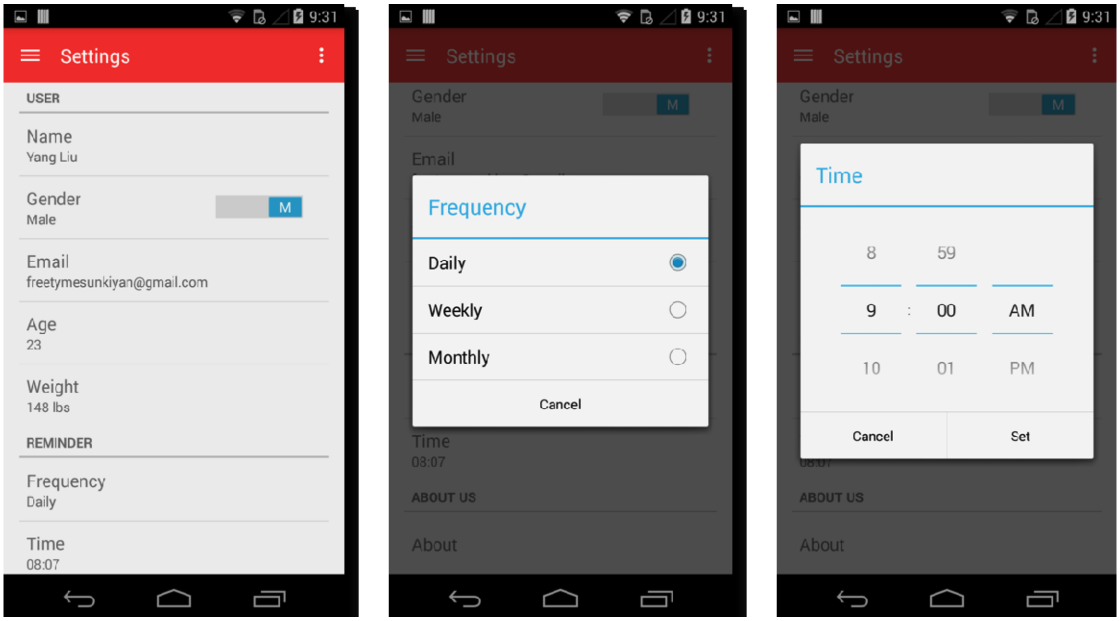
- BPM: Beats per Minute
- MHR: Max Heart Rate

Figure 4.15 The Analysis Activity: History and Report

is too small to see clearly, the user can tap twice continuously on that value to zoom in, and then drag the graph to move around for other values. Screenshots of the analysis activity are in Figure 4.15.

When sliding from the right to the left, the user can switch to the report page. This page contains a table for different exercising levels and their recommended heart rate range. If there is no gender and age information, the range will only be percentages of max heart rate and a button will show below the table for the user to open Settings Activity directly to set them. Otherwise, there will be actual values in a table instead and no button.

The last activity is the settings activity. In this activity the user’s basic information can be set and a reminder can also be set to notify the user to take measurements. If the user clicks on the setting items, a dialog will appear with a text field, options or a time picker. Relative values and functions will be updated after being set in the dialog. The dialogs for notification frequency and time are shown in Figure 4.16. For example, a reminder at 09:48 PM in the APP is set. When



Settings

USER

Name
Yang Liu

Gender
Male

Email
freetymesunkiyan@gmail.com

Age
23

Weight
148 lbs

REMINDER

Frequency
Daily

Time
08:07

Frequency

Daily ●

Weekly ○

Monthly ○

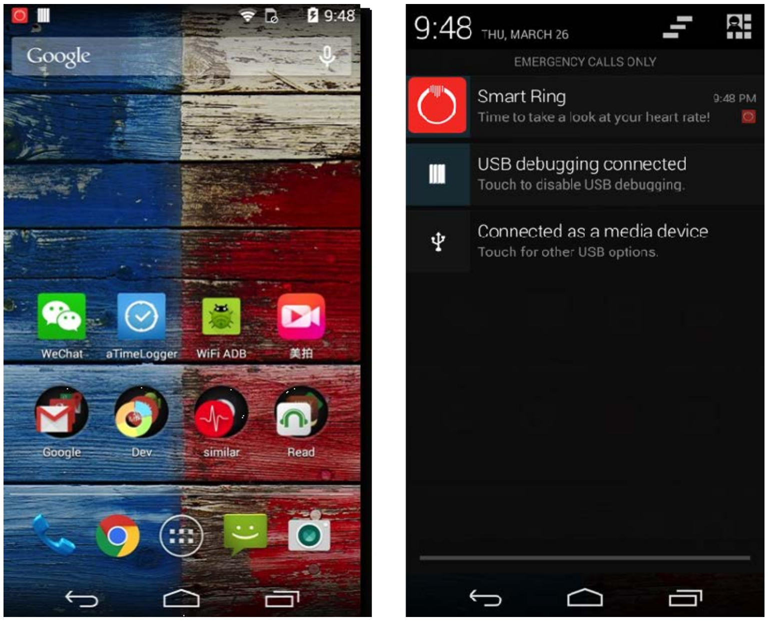
Time

8 59

9 : 00 AM

10 01 PM

Figure 4.16 The Settings Activity



9:48 THU, MARCH 26

EMERGENCY CALLS ONLY

Smart Ring 9:48 PM
Time to take a look at your heart rate!

USB debugging connected
Touch to disable USB debugging.

Connected as a media device
Touch for other USB options.

Figure 4.17 Screenshots of Example Notification

the time arrives, the phone will vibrate and beep with an icon showing on the top bar. If the user slides the bar at screen top downwards, a more detailed notification with a message and a bigger icon will be there. Figure 4.17 is the screenshots for the example notification view.

5. NFC RING PROTOTYPE AND COMPARISON WITH A RFID RING

Combining the methods illustrated in the previous sections, a prototype NFC Ring is designed, implemented, and compared with RFID ring as well. The prototype and entire flow are presented in Section 5.1. Section 5.2 presents comparison of the NFC Ring with a RFID ring.

5.1. A Prototype of the NFC Ring

An NFC Ring prototype is developed to implement the sensing, forwarding, and receiving procedure. The complete system consists of a ring-shape hardware named NFC Ring and an Android APP that connects to the NFC Ring through the NFC. Physical appearance of the prototype is presented in Figure 5.1.

The NFC Ring is used to collect raw heart rate data from user's finger, filter it, and then

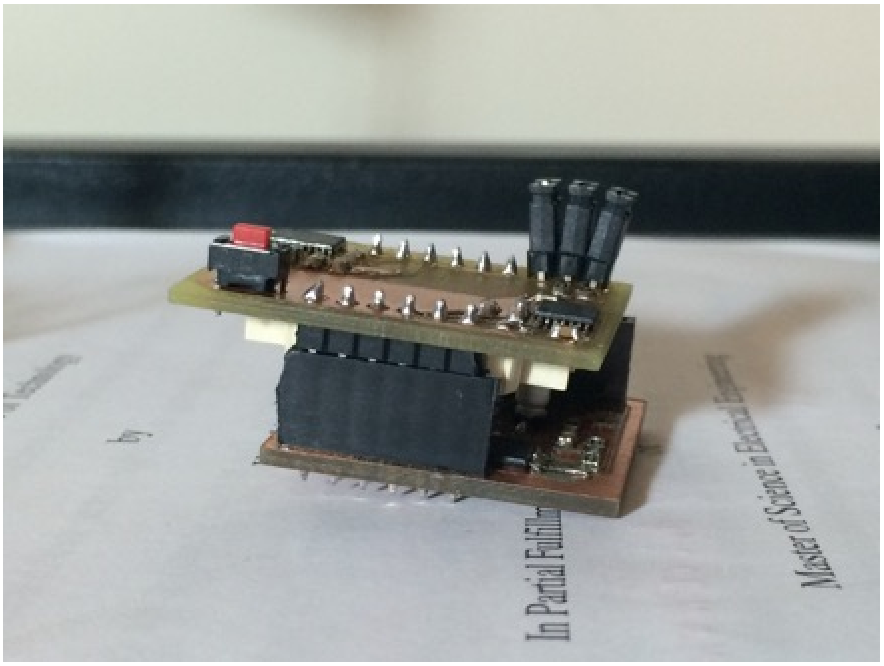


Figure 5.1 The Prototype

forward it to the NFC tag. A reflective optical sensor is chosen for the raw data acquisition, which is relevant to the change of luminance in pace with the flow of blood under skin. Then two filter stages are built for the signal, removing unrelated signals and making it readable. The frequency of the filtered signal is exactly the heart rate per minute value of the user. The signal is further sent into a microcontroller for frequency calculation. By detecting the low-to-high edge in the signal, the interval between two pulses can be identified accurately. The count of the timer interrupts during the interval is used later to compute the frequency.

Next, the NFC tag is initiated and enabled to encode and send the calculation result. The process of forwarding continues as data is generated, and whenever an electronic device with an NFC reader approaches within transmission range, it will discover the signal of the NFC module and present a notification to the user. If the APP developed for the ring is not yet installed on the device, it will open a webpage for the user to download the APP. If the APP is already installed, it will be opened directly. The measurement interface will appear and the measurement starts.

An active measurement will receive heart rate data from the NFC Ring continuously for about 15 seconds. Then all of the data is used to generate an accurate result and notify the user. The user can choose whether this result should be recorded or not, and which type of state the user is in right now. For example, the user could be exercising and would want to check how intense the exercise is. Or the user could be measuring in a resting state to gain an impression on how healthy is her overall cardiovascular system. Different states can have completely different meanings, so it is important to categorize them into various types. Once the type of state is decided, its measured result is stored in database for the APP to plot a history graph and to provide health suggestions.

5.2. Comparison with A RFID Ring

A mobile e-health-management system is proposed based on mobile physiological signal monitoring [13]. This system integrates a wearable ring-type pulse monitoring sensor and a portable bio-signal recorder with a cell phone. The ring-type pulse sensor transfers its data to a stationary RFID reader. Then the reader forwards the data to a cell phone via Bluetooth. If the connection between the cell phone and RFID reader is set up, the cell phone application can monitor real-time pulse rate. An interface is also provided for setting contact information so that whenever there is an emergency situation, for example, the pulse rate is irregularly low, that contact can be notified in time. This design is named the RFID ring in this thesis.

With an RFID antenna, the transmission distance of the RFID ring is longer than with NFC, but the RFID antenna must be connected to a RFID reader hub. Then, the hub can relay data to a cell phone. The hub is a large board with an antenna, an MCU, and a Bluetooth module on it. Therefore, it is not convenient to move it around often. The connection between the reader and the phone also needs additional setup to function. Although the RFID ring shows an available way to construct the data flow between the sensor and a mobile device through wireless connection, it is not very practical due to the complexity of system and lack of mobility.

Since the support of smartphone-to-NFC technology is already widespread and growing rapidly, replacing RFID technology with NFC to build a wireless connection between the ring and an NFC supported device is feasible. NFC simplifies the system dramatically by removing the complex RFID reader hub and enabling direct ring-phone interaction. The NFC antenna is also smaller than the RFID antenna, potentially allowing for a smaller ring. The only negative side of NFC is the close transmission distance. But this problem is avoided by conducting measurement with the ring on a finger and the phone in the same hand. Another improvement is

Table 5.1 The Comparison Between the RFID Ring and the NFC Ring

Aspects	The RFID Ring	The NFC Ring
System modules	Ring, RFID Reader, Phone	Ring, Phone
Wireless Technology	RFID, Bluetooth	NFC
Purpose	Continuous Monitoring	Measurement
Transmission Dist.	Long	Very Close
Mobility	Stationary	Wherever with an Android phone
Need Setup?	Yes	No
Device Support	Windows Phone 6.1	Android Phone with NFC

th ch must set up the connection manually, the NFC Ring makes the phone open the relevant APP and starts measurement. From a software perspective, the NFC Ring supports any Android devices with an NFC module. Table 5.1 presents the comparison between the two rings from various angles.

To sum up, a more practical system is built by substituting RFID with NFC. Removing the RFID reader hub not only reduces the system complexity, but also shrinks the ring to enhance mobility. Despite the short communication range, it is feasible and acceptable under normal user scenarios with the ring and the phone in the same hand.

6. CONCLUSION AND FUTURE WORK

An entire WBAN system that consists of an NFC ring, a customized NFC antenna, and an Android APP is presented. The NFC Ring is equipped with a sensor to convert heart rate into an electrical signal, a microcontroller to process the signal and write the result, and an NFC tag for wireless communication between the ring and a smartphone. The customized NFC antenna is made to fits the ring while maintaining stable connection. The APP controls the NFC module on an Android device to read, store, and analyze data. The system demonstrates that with a smartphone as a central unit, NFC-linked sensors can be embedded into daily accessories.

A prototype for the system is implemented to demonstrate the concept and explain the procedure. Also, the system is compared with a similar one that uses RFID for wireless communication. The RFID ring is excessively complex in comparison to the NFC Ring in terms of system modules and the setup process. Also, the device support is not in line with the current trend. In conclusion, the combination of the NFC Ring and an Android smartphone is more practical and preferable.

Due to the time constraints, only a prototype with fundamental features is implemented to explain the idea. Additional research can be performed to improve the NFC Ring. In terms of hardware, more experiments can be conducted on the NFC antenna to enhance the stability of data transmission. With respect to software, the measurement algorithm may be improved to generate more accurate data. There are potential research topics in processing data by digging deeper into what the data and its trend mean as well.

Experiments on how to enhance the NFC antenna are worth studying. To make the communication more stable, the antenna inductance should be higher, which means that the size of the inductance coil should be increased. First, different sizes or shapes can be tested to find out the maximum quality factor. Furthermore, to fit the shape of a ring, flexible PCB that stretches wider than regular ones can replace the original PCB. Using more than one antenna, for example, a combination of two antennas in an angle to form a group, is also appealing. Distinct magnetic fields would be formed. A best angle that contributes to the strongest field strength may be found.

Both the program that runs on the microcontroller to calculate the period of the pulse signal and the measurement process in the APP can be improved. The period calculation algorithm covers the time interval between two pulses with a timer. Basic error checking that removes abnormal high values could be added. Other errors could be considered to make it more robust, such as eliminating data when the variance is increasing vastly. Similarly, the same error handling idea could be applied to the measuring process. In addition, adding one more determination stage could refine the flow of measurement algorithm. The stage should be able to confirm automatically that the data is stable before using it for calculation. These improvements could all contribute to more accurate data.

As for data processing, besides displaying recent heart rate measurements with the proper graph to express the visual trend, the data has a deeper meaning that can be further revealed. For instance, the data indicates how healthy a person’s cardio-vascular system is, which could be categorized into different fitness levels sequentially. Another possible feature could be offering predictions based on the heart rate trend. Then a warning could be sent to users whose heart rate is developing towards a bad direction, or an encouraging notification if the user is heading toward a healthy condition. These new features require more knowledge in pattern recognition and data mining.

LIST OF REFERENCES

[1]. Nakano, Keisuke, and Toshiyuki Murakami. "An approach to guidance motion by gait-training equipment in semipassive walking." Industrial Electronics, IEEE Transactions on 55.4 (2008): 1707-1714.

[2]. Isern, David, David Sánchez, and Antonio Moreno. "Agents applied in health care: A review." International journal of medical informatics 79.3 (2010): 145-166.

[3]. Chen, Chih-Ming. "Web-based remote human pulse monitoring system with intelligent data analysis for home health care." Expert Systems with Applications 38.3 (2011).

[4]. Ullah, Sana, et al. "A comprehensive survey of wireless body area networks." Journal of medical systems 36.3 (2012): 1065-1094.

[5]. Chen, Min, et al. "Body area networks: A survey." *Mobile networks and applications* 16.2 (2011): 171-193.

[6]. Movassaghi, Samaneh, et al. "Wireless body area networks: a survey." (2014): 1-29.

[7]. Schmidt, Robert, et al. "Body Area Network BAN–a key infrastructure element for patient-centered medical applications." *Biomedizinische Technik/Biomedical Engineering* 47.s1a (2002): 365-368.

[8]. O'Donovan, Tony, et al. "A context aware wireless body area network (BAN)." *Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on*. IEEE, 2009.

[9]. Bennett, Stuart. A history of control engineering, 1930-1955. No. 47. IET, 1993.

[10]. Semiconductors, Vishay. "Reflective Optical Sensor with Transistor Output."

[11]. Nosowitz, Dan. "Everything you need to know about near field communication." Popular Science Magazine. Popular Science 1 (2011).

[12]. Coskun, Vedat, Busra Ozdenizci, and Kerem Ok. "A survey on near field communication (NFC) technology." *Wireless personal communications* 71.3 (2013): 2259-2294.

[13]. Wu, Yu-Chi, et al. *A mobile-phone-based health management system*. INTECH Open Access Publisher, 2011.

[14]. Hashem, M. M. A., et al. "Design and development of a heart rate measuring device using fingertip." *Computer and Communication Engineering (ICCCE), 2010 International Conference on*. IEEE, 2010.

[15]. Wikipedia, “Heart rate”, Available at: http://en.wikipedia.org/wiki/Heart_rate [December 27, 2009]

[16]. Embedded Lab, “Introducing Easy Pulse: A DIY photoplethysmographic sensor for measuring heart rate”, Available at: http://embedded-lab.com/blog/?p=5508 [September 12, 2012]

[17]. Embedded Computing Design, “Measuring heart rate and blood oxygen levels for portable medical and wearable devices”, http://embedded-computing.com/articles/measuring-levels-portable-medical-wearable-devices/ [April 23, 2014]

[18]. Android Developers, “NFC Basics”, Available at: https://developer.android.com/guide/topics/connectivity/nfc/nfc.html

[19]. Android Developers, “Advanced NFC”, Available at: https://developer.android.com/guide/topics/connectivity/nfc/advanced-nfc.html

[20]. Android Developers, “Reference Documents android.database.sqlite.SQLiteOpenHelper”, Available at: http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

[21]. Robergs, Robert A., and Roberto Landwehr. "The surprising history of the “HRmax= 220-age” equation." *J Exerc Physiol* 5.2 (2002): 1-10.

[22]. Gulati, Martha, et al. "Heart Rate Response to Exercise Stress Testing in Asymptomatic Women The St. James Women Take Heart Project." Circulation 122.2 (2010): 130-137.

[23]. Antenna Theory.com, “NFC Antennas”, Available at: http://www.antenna-theory.com/definitions/nfc-antenna.php

[24]. NXP Semiconductors, “AN1445 Antenna design guide for MFRC52x, PN51x and PN53x”, Rev. 1.2 — 11 October 2010, 144512.

[25]. Wikipedia, “Permeability (electromagnetism)”, Available at: http://en.wikipedia.org/wiki/Permeability_(electromagnetism)

[26]. Texas Instruments, “RF430CL330H Practical Antenna Design Guide”, Application Report, SLOA197 — June 2014.

[27]. Sprague- Goodman, “FILMTRIM® PLASTIC DIELECTRIC CAPACITORS”, Engineering Bulletin SG-402H.1, Supercedes SG-402G.

[28]. Reid, Wes. "Mixed-Signal Microcontroller." HEWLETT-PACKARD JOURNAL 48.2 (1997): 8-8.

[29]. TI E2E Community, “Difference between __enable_interrupt() and __bis_SR_register(GIE)”, Available at: https://e2e.ti.com/support/microcontrollers/msp430/f/166/t/252264

[30]. Type 4 Tag Operation Specification, Technical Specification NFC Forum T4TOP 2.0 NFCForum-TS-Type-4-Tag_2.0 20011-06-28.

[31]. Instruments, Texas. "Mixed Signal Microcontroller." MSP430G2x53 datasheet, Apr (2011).

APPENDICES

Appendix A. Circuit Diagrams

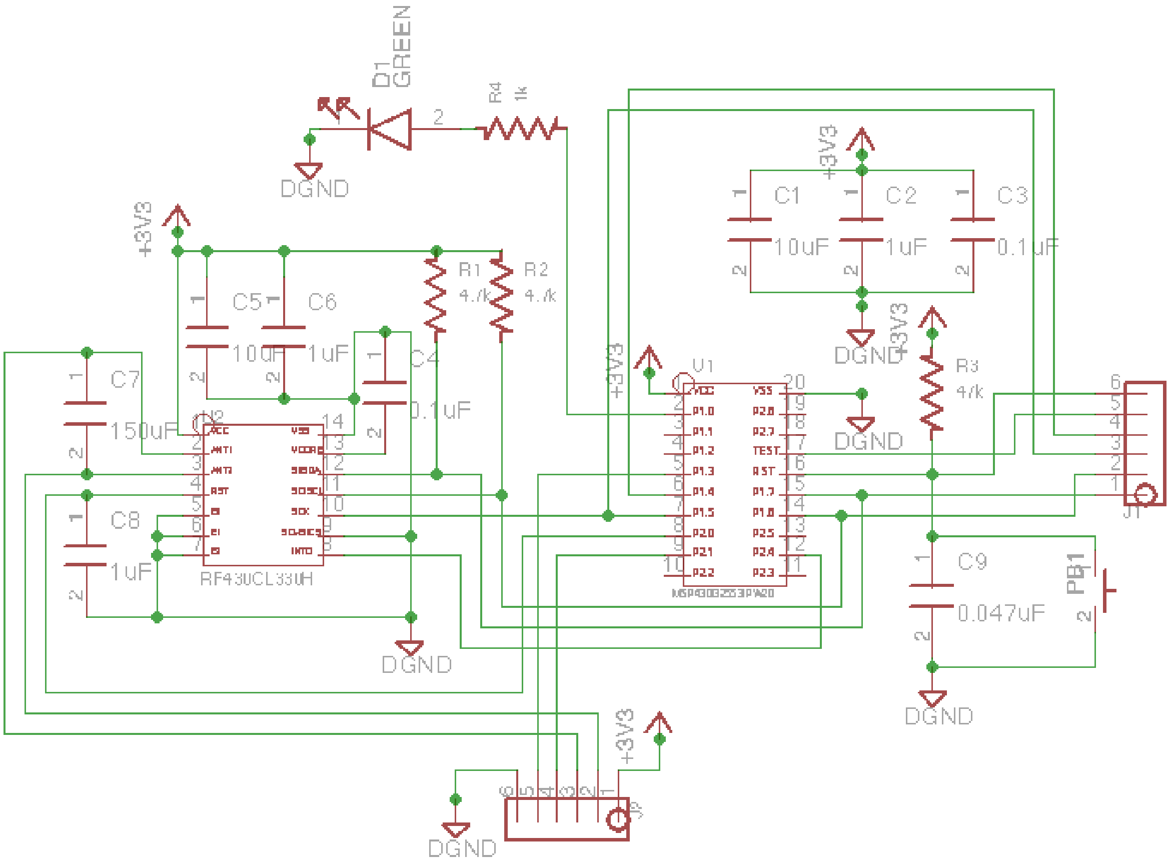


Figure A.1 Ring control board schematic

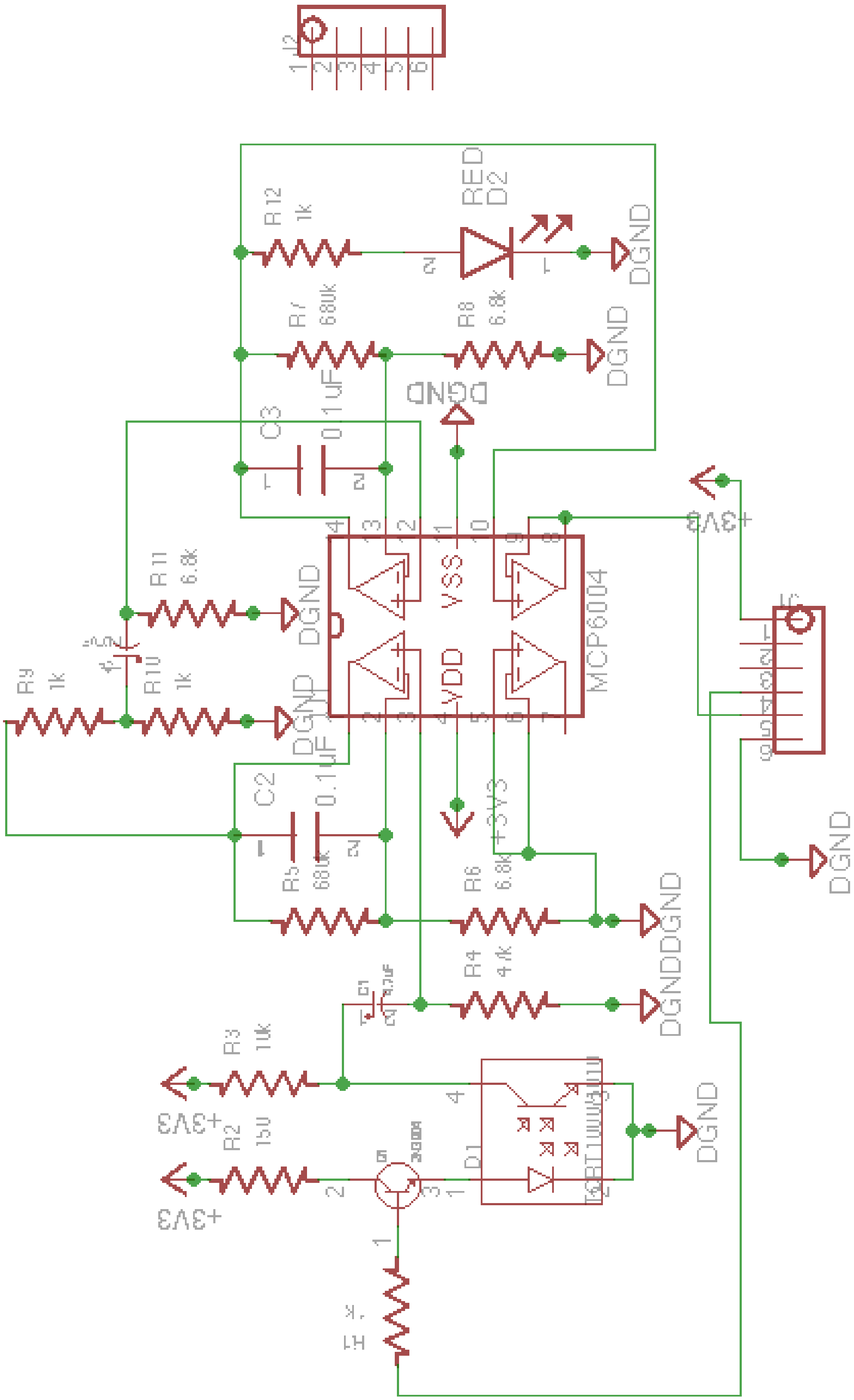


Figure A.2 Ring sensing board schematic

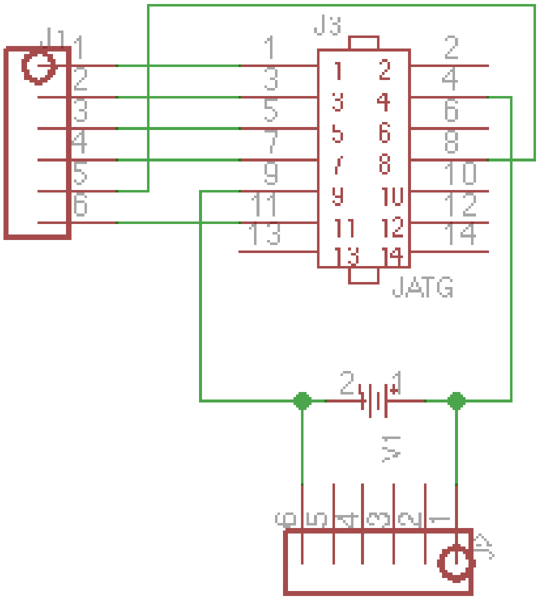


Figure A.3 Ring testing board schematic

Appendix B. PCB Layout Plots

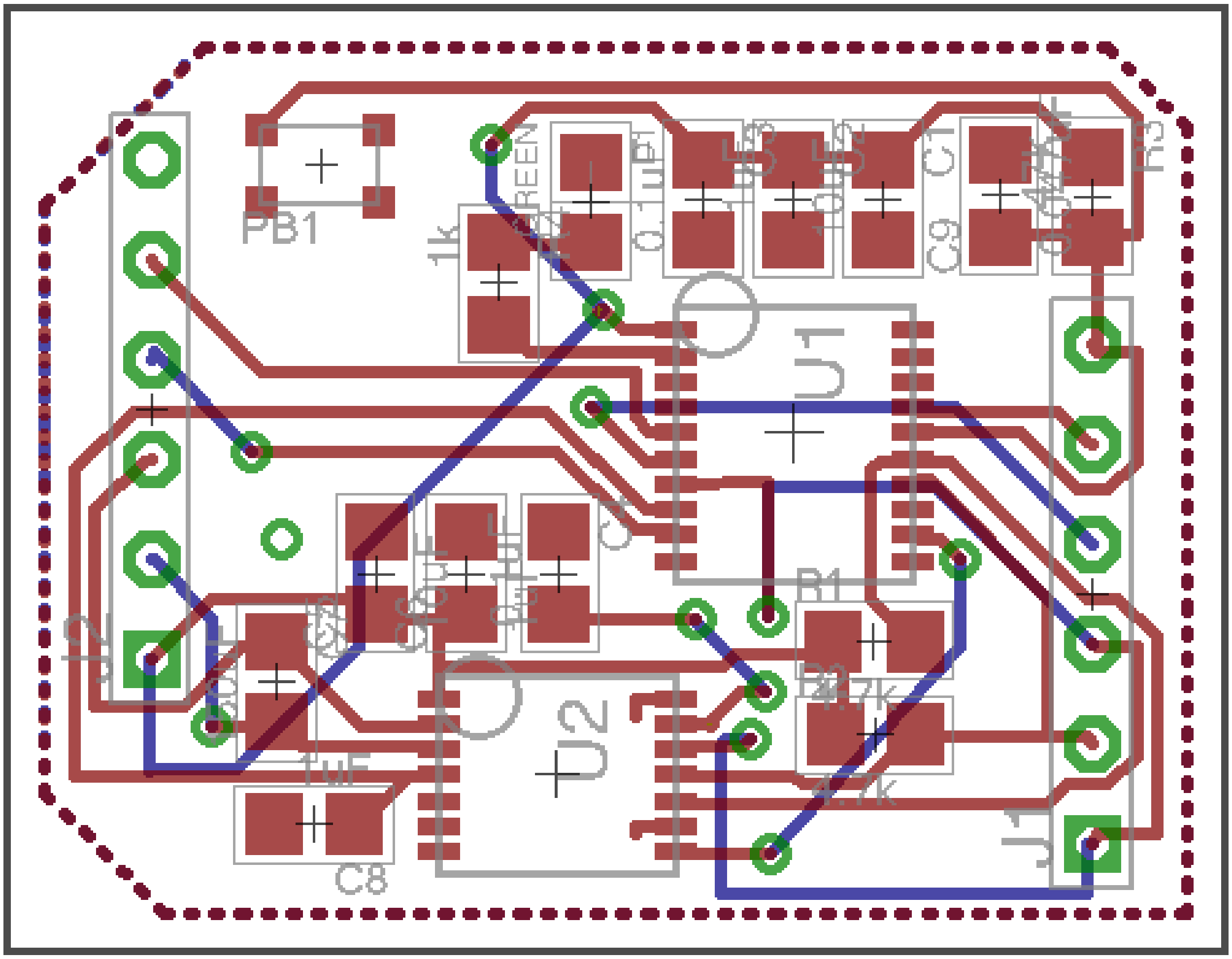


Figure B.1 Control board layout

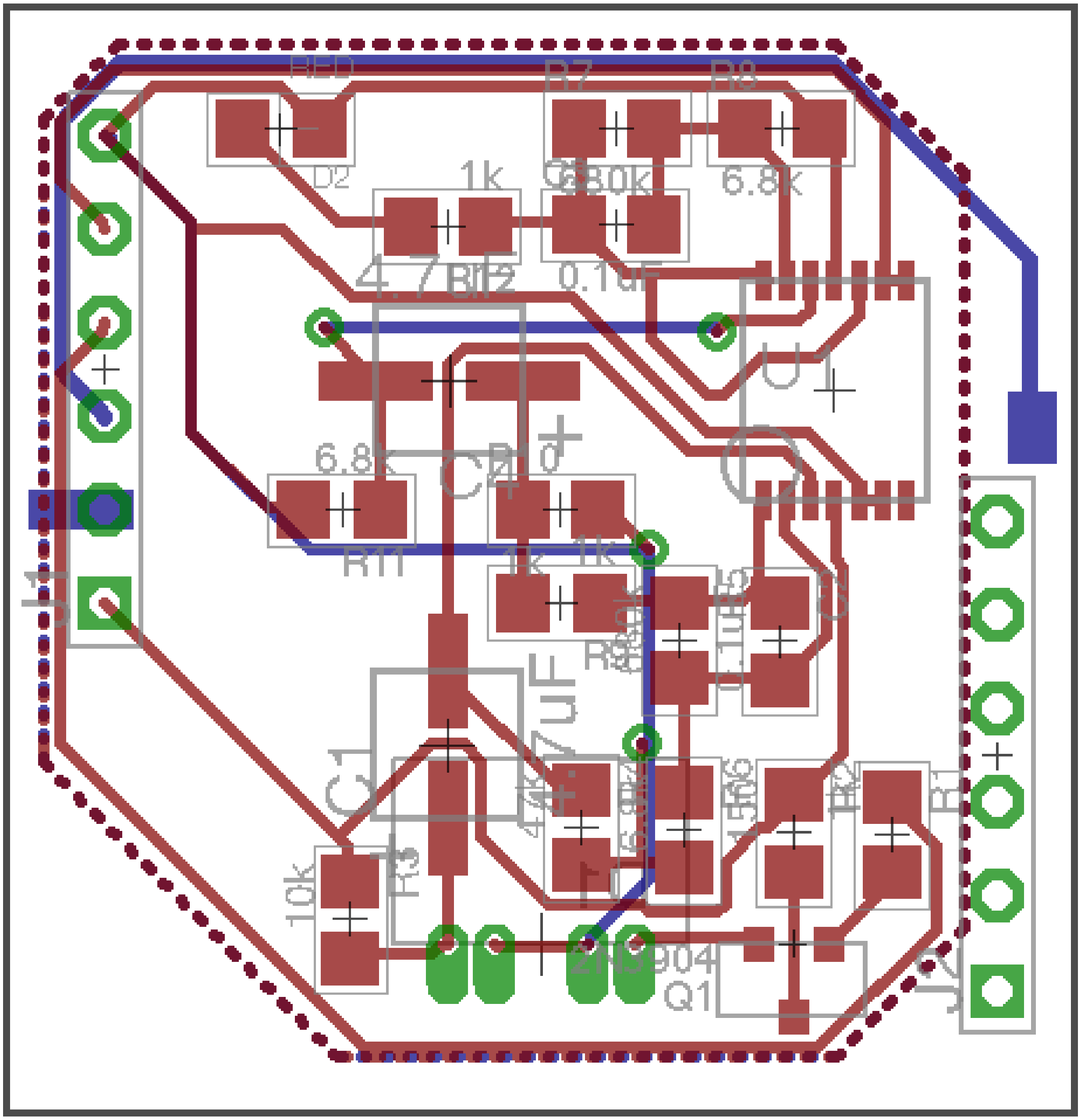


Figure B.2 Sensor board layout

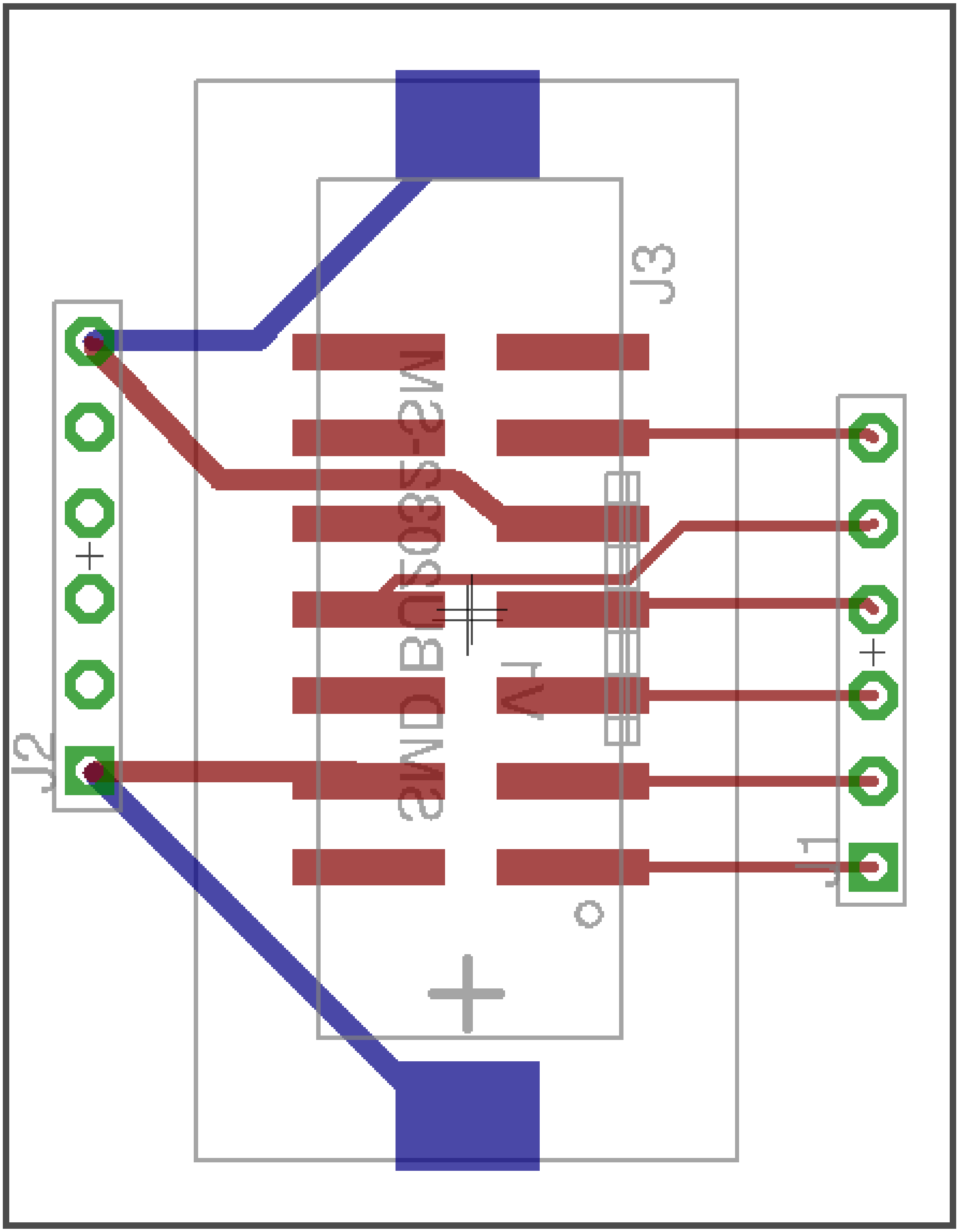


Figure B.3 Testing board layout

Appendix C. Software on Microcontroller

main.c

```
#include "msp430.h"
#include "RF430.h"

#define APP_DATA_LENGTH 59
#define DATA_LENGTH     40

/*
 * main.c
 */
// Global variables
// Heartbeat count
unsigned char get_new_pulse = 0;
unsigned short pulse_timer_counter = 0;
unsigned short pulse_count = 0;
// NFC data array
unsigned char NDEF_Application_APP_Data[] = RF430_APP_DATA;
unsigned char NDEF_Application_Data[] = RF430_DEFAULT_DATA;

void setup() {
    WDTCTL = WDTPW + WDTTMSEL + WDTCNTCL + WDTIS0; // watchdog clock
    IE1 |= WDTIE; //interrupt enable

    BCSCTL1 = CALBC1_8MHZ; // Set range
    DCOCTL = CALDCO_8MHZ; // SMCLK = MCLK = DCO = 8 MHz ACLK = LF oscillator

    // Configure pins for P1
    // Configure pins for I2C
    P1SEL = (unsigned char) (0x00 | BIT6 | BIT7); // Selecting I2C pin function
    P1SEL2 = (unsigned char) (0x00 | BIT6 | BIT7);
    P1OUT = (unsigned char) (0x00 | BIT3); // P1.3 pull up and others pull down
    P1DIR = (unsigned char) (0xFF & (~BIT3)); // P1.3 input and others output
    P1REN = (unsigned char) (0xFF & (~BIT0) & (~BIT6) & (~BIT7)); // P1.0 pull
up/down resistor disable for LED output, and P1.6 P1.7 disable for I2C.
    // Configure the P1 interrupt: Enable the P1.3 interrupt for signal input
    P1IE = (unsigned char) (0x00);
    P1IFG &= ~(BIT3); // Clear interrupt flag
    P1IES &= ~(BIT3); // Set interrupt trigger as low-to-high transition

    // configure USCI for I2C
    UCB0CTL1 |= UCSWRST; // Software reset enabled
    UCB0CTL0 |= UCMODE_3 + UCMST + UCSYNC; // I2C mode, Master mode, sync
    UCB0CTL1 |= UCSSEL_3 + UCTR; // SMCLK = 8MHz, transmitter
    UCB0BR0 = 80; // Baudrate = SMLK/80 = 100kHz
    UCB0I2CSA = 0x0028; // slave address - determined by pins E0, E1, and E2 on the
RF430CL330H
    UCB0CTL1 &= ~UCSWRST; // software reset released

    // Configure pins for P2
    P2SEL = (unsigned char) (0x00);
    P2SEL2 = (unsigned char) (0x00);
    P2OUT = (unsigned char) (0x00 | BIT4);
    P2DIR = (unsigned char) (0xFF & ~(BIT4));
    P2REN = (unsigned char) (0xFF);
    // Configure the P2 interrupt:
    P2IE = (unsigned char) (0x00);
    _enable_interrupt();
    __bis_SR_register(GIE);
    // Initialize
    __delay_cycles(1000);
    P2OUT |= BIT0; // Release the RF430CL330H from Reset
    Write_Register(CONTROL_REG, SW_RESET);
    __delay_cycles(4000000); // Leave time for the RF430CL33H to get itself
initialized; should be 20ms or greater
    while (!(Read_Register(STATUS_REG) & READY))
        ; // Wait until READY bit has been set
}

int main(void) {
    unsigned short status = 0;
    unsigned char IE_heartbeat = 0;
    setup();

    while (1) {
        Write_Continuous(0, NDEF_Application_APP_Data, APP_DATA_LENGTH);
        Write_Register(CONTROL_REG, RF_ENABLE);
        __delay_cycles(4000000);

        status = Read_Register(STATUS_REG);
        pulse_timer_counter = 0;
        while (status & RF_BUSY) {
            if (!IE_heartbeat) {
                P1IE |= BIT3;
                P1IFG &= ~(BIT3);
                IE_heartbeat = 1;
            }
            P1OUT ^= BIT0;
            status = Read_Register(STATUS_REG);
            if (pulse_count > 5000) {
                pulse_count = 0;
            } else if (get_new_pulse) {
                get_new_pulse = 0;
                unsigned short data = pulse_count;
                Write_Register(CONTROL_REG, SW_RESET);
                __delay_cycles(4000000);
                NDEF_Application_Data[35] = (data / 1000) + 0x30;
                NDEF_Application_Data[36] = ((data % 1000) / 100) + 0x30;
                NDEF_Application_Data[37] = ((data % 100) / 10) + 0x30;
                NDEF_Application_Data[38] = (data % 10) + 0x30;
                Write_Continuous(0, NDEF_Application_Data, DATA_LENGTH);
                Write_Register(CONTROL_REG, RF_ENABLE);
                __delay_cycles(4000000);
            } else { // no new pulse
                __delay_cycles(1000000);
            }
        }
        if (IE_heartbeat) {
            P1IE &= ~(BIT3);
            P1IFG &= ~(BIT3);
            IE_heartbeat = 0;
        }
        Write_Register(CONTROL_REG, SW_RESET);
        __delay_cycles(4000000);
    }
    return 0;
}

#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    pulse_timer_counter++;
}

#pragma vector=PORT1_VECTOR
__interrupt void Port1_ISR(void) {
    if (P1IFG & BIT3) {
        get_new_pulse = 1; // get the heartbeat pulses
        pulse_count = pulse_timer_counter;
        pulse_timer_counter = 0;
        P1IFG &= ~(BIT3); // clear interrupt flag
    }
}
```

RF430.h

```
#ifndef RF430_H_
#define RF430_H_

unsigned int Read_Register(unsigned int reg_addr);
unsigned int Read_Register_BIP8(unsigned int reg_addr);
void Read_Continuous(unsigned int reg_addr, unsigned char* read_data, unsigned int
data_length);

void Write_Register(unsigned int reg_addr, unsigned int value);
void Write_Continuous(unsigned int reg_addr, unsigned char* write_data, unsigned int
data_length);
void Write_Register_BIP8(unsigned int reg_addr, unsigned int value);

//define the values for Granite's registers we want to access
#define CONTROL_REG         0xFFFE
#define STATUS_REG          0xFFFC
#define INT_ENABLE_REG      0xFFFA
#define INT_FLAG_REG        0xFFF8
#define CRC_RESULT_REG      0xFFF6
#define CRC_LENGTH_REG      0xFFF4
#define CRC_START_ADDR_REG  0xFFF2
#define COMM_WD_CTRL_REG    0xFFF0
#define VERSION_REG         0xFFEE //contains the software version of the ROM
#define TEST_FUNCTION_REG   0xFFE2
#define TEST_MODE_REG       0xFFE0

//define the different virtual register bits
//CONTROL_REG bits
#define SW_RESET        BIT0
#define RF_ENABLE       BIT1
#define INT_ENABLE      BIT2
#define INTO_HIGH       BIT3
#define INTO_DRIVE      BIT4
#define BIP8_ENABLE     BIT5
#define STANDBY_ENABLE  BIT6
#define TEST430_ENABLE  BIT7
//STATUS_REG bits
#define READY           BIT0
#define CRC_ACTIVE      BIT1
#define RF_BUSY         BIT2
//INT_ENABLE_REG bits
#define EOR_INT_ENABLE      BIT1
#define EOW_INT_ENABLE      BIT2
```

```
#define CRC_INT_ENABLE      BIT3
#define BIP8_ERROR_INT_ENABLE   BIT4
#define NDEF_ERROR_INT_ENABLE   BIT5
#define GENERIC_ERROR_INT_ENABLE    BIT7
//INT_FLAG_REG bits
#define EOR_INT_FLAG    BIT1
#define EOW_INT_FLAG    BIT2
#define CRC_INT_FLAG    BIT3
#define BIP8_ERROR_INT_FLAG BIT4
#define NDEF_ERROR_INT_FLAG BIT5
#define GENERIC_ERROR_INT_FLAG  BIT7
//COMM_WD_CTRL_REG bits
#define WD_ENABLE   BIT0
#define TIMEOUT_PERIOD_2_SEC    0
#define TIMEOUT_PERIOD_32_SEC   BIT1
#define TIMEOUT_PERIOD_8_5_MIN  BIT2
#define TIMEOUT_PERIOD_MASK     BIT1 + BIT2 + BIT3

#define RF430_APP_DATA {
\
/*NDEF Tag Application Name*/ \
0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01, \
\
/*Capability Container ID*/ \
0xE1, 0x03, \
0x00, 0x0F, /* CCLEN */ \
0x20,       /* Mapping version 2.0 */ \
0x00, 0xF9, /* MLe (49 bytes); Maximum R-APDU data size */ \
0x00, 0xF6, /* MLc (52 bytes); Maximum C-APDU data size */ \
0x04,       /* Tag, File Control TLV (4 = NDEF file) */ \
0x06,       /* Length, File Control TLV (6 = 6 bytes of data for this tag) */ \
0xE1, 0x04, /* File Identifier */ \
0x0B, 0xDF, /* Max NDEF size (3037 bytes of useable memory) */ \
0x00,       /* NDEF file read access condition, read access without any security */
\
0x00,       /* NDEF file write access condition; write access without any security
*/ \
\
/* NDEF File ID */ \
0xE1, 0x04, \
\
/* NDEF File for Hello World */ \
0x00, 0x1F, /* NLEN: NDEF length (20 byte long message, max. length for RF430CL) */\
\
/* NDEF Record (refer to NFC Data Exchange Format specifications)*/ \
0xD1,       /*MB(Message Begin), SR(Short Record) flags set, ME(Message End), IL(ID
length field present) flags cleared; TNF(3bits) = 1; */ \
0x01, 0x1B, /*Type Length = 0x01; Payload Length = 0x10 */ \
0x55,       /* Type = U (text) */ \
0x03,       /*http://www.*/ \
\
/*me.freetymekiyan.smartring*/ \
'm','e','.','f','r','e','e','t','y','m','e','k','i','y','a','n','.','s','m','a','r',
't','r','i','n','g' \
} /* End of data */

#define RF430_DEFAULT_DATA
{                                                       \
/*NDEF Tag Application Name*/
\
0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01,
\
\
/*Capability Container ID*/
\
0xE1, 0x03,
\
0x00, 0x0F, /* CCLEN */
\
0x20,       /* Mapping version 2.0 */
\
0x00, 0xF9, /* MLe (49 bytes); Maximum R-APDU data size */
\
0x00, 0xF6, /* MLc (52 bytes); Maximum C-APDU data size */
\
0x04,       /* Tag, File Control TLV (4 = NDEF file) */
\
0x06,       /* Length, File Control TLV (6 = 6 bytes of data for this tag) */
\
0xE1, 0x04, /* File Identifier */
\
0x0B, 0xDF, /* Max NDEF size (3037 bytes of useable memory) */
\
0x00,       /* NDEF file read access condition, read access without any security */
\
0x00,       /* NDEF file write access condition; write access without any security
*/   \

\
/* NDEF File ID */
\
0xE1, 0x04,
\

\
/* NDEF File for Hello World */
\
0x00, 0x0C, /* NLEN: NDEF length (20 byte long message, max. length for RF430CL) */
\

\
/* NDEF Record (refer to NFC Data Exchange Format specifications)*/
\
0xD1,       /*MB(Message Begin), SR(Short Record) flags set, ME(Message End), IL(ID
length field present) flags cleared; TNF(3bits) = 1; */ \
0x01, 0x08, /*Type Length = 0x01; Payload Length = 0x10 */
\
0x54,       /* Type = T (text) */
\
0x02,       /* 1st payload byte: "Start of Text", as specified in ASCII Tables */
\
0x65, 0x6E, /* 'e', 'n', (2nd, 3rd payload bytes*/
\
0x41, 0x42, 0x43, 0x44, 0x03
\
} /* End of data */

#endif
```

Appendix D. Software on Android

The files presented here are from the Android SmartRing APP project. Only significant files are listed in this section, such as the main interfaces, the database helper, the preference helper, and the alarm receiver for notification.

MainActivity.java

```
package me.freetymekiyan.smartring.views;

public class MainActivity extends ActionBarActivity implements
OnFragmentInteractionListener,
        MeasureOneFragment.OnMeasureListener {

    public static final String KEY_TITLE = "title";

    public static final int MEASURE_FRAGMENT = 1;

    public static final int ANALYSIS_FRAGMENT = 2;

    public static final int SETTINGS_FRAGMENT = 3;

    private final int profile = R.drawable.photo;

    private final int[] icons = {R.drawable.ic_measure, R.drawable.ic_history,
            R.drawable.ic_settings};

    private Toolbar toolbar;

    RecyclerView mRcView;

    DrawListAdapter mDrawerAdapter;

    RecyclerView.LayoutManager mLayoutManager;

    DrawerLayout drawer;

    ActionBarDrawerToggle mDrawerToggle;

    private int page = MEASURE_FRAGMENT;

    private NfcAdapter mAdapter;

    private PendingIntent mPendingIntent;

    private IntentFilter[] mIntentFilters;

    private int count = 0;

    public static final int START_INDEX = 3;

    private float sum;

    private float result;
    private boolean enabled = false;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        toolbar = (Toolbar) findViewById(R.id.tool_bar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setTitle(R.string.measure);

        final SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(this);
        String name = sp.getString(getString(R.string.key_name),
getString(R.string.ph_name));
        String email = sp.getString(getString(R.string.key_email),
getString(R.string.ph_email));

        mRcView = (RecyclerView) findViewById(R.id.recycle_view);
        mRcView.setHasFixedSize(true);
        mDrawerAdapter = new
DrawListAdapter(getResources().getStringArray(R.array.titles), icons,
                name,
                profile, email);
        mRcView.setAdapter(mDrawerAdapter);
        mLayoutManager = new LinearLayoutManager(this);
        mRcView.setLayoutManager(mLayoutManager);

        drawer = (DrawerLayout) findViewById(R.id.drawer);
        mDrawerToggle = new ActionBarDrawerToggle(this, drawer, toolbar,
R.string.open_drawer,
                R.string.close_drawer) {
            @Override
            public void onDrawerOpened(View drawerView) {
                super.onDrawerOpened(drawerView);
                toolbar.setTitle(R.string.app_name);
            }

            @Override
            public void onDrawerClosed(View drawerView) {
                super.onDrawerClosed(drawerView);
                switch (page) {
                    case MEASURE_FRAGMENT:
                        toolbar.setTitle(R.string.measure);
                        break;
                    case ANALYSIS_FRAGMENT:
                        toolbar.setTitle(R.string.analysis);
                        break;
                    case SETTINGS_FRAGMENT:
                        toolbar.setTitle(R.string.settings);
                        break;
                    default:
                        break;
                }
            }
        };
        drawer.setDrawerListener(mDrawerToggle);
        mDrawerToggle.syncState();

        mRcView.addOnItemTouchListener(new
RecyclerItemClickListener(MainActivity.this,
                new RecyclerItemClickListener.OnItemClickListener() {
                    @Override
                    public void onItemClick(View view, int position) {
                        switch (position) {
                            case MEASURE_FRAGMENT:
                                if (page != MEASURE_FRAGMENT) {

getSupportFragmentManager().beginTransaction().replace(
                                            R.id.content_frame,
MeasureOneFragment.newInstance())
                                            .commit();
                                    page = MEASURE_FRAGMENT;
                                    toolbar.setTitle(R.string.measure);
                                }
                                break;
                            case ANALYSIS_FRAGMENT:
                                if (page != ANALYSIS_FRAGMENT) {
                                    getSupportFragmentManager().beginTransaction()
                                            .replace(R.id.content_frame,

AnalysisFragment.getInstance()).commit();
                                    page = ANALYSIS_FRAGMENT;
                                    toolbar.setTitle(R.string.analysis);
                                }
                                break;
                            case SETTINGS_FRAGMENT:
                                if (page != SETTINGS_FRAGMENT) {
                                    getSupportFragmentManager().beginTransaction()
                                            .replace(R.id.content_frame,
                                                    new
MyPreferenceFragment()).commit();
                                    page = SETTINGS_FRAGMENT;
                                    toolbar.setTitle(R.string.settings);
                                }
                                break;
                            default:
                                break;
                        }
                        drawer.closeDrawer(Gravity.LEFT);
                    }
                }));
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.content_frame,
MeasureOneFragment.newInstance()).commit();
        // NFC Adapter
        mAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mAdapter == null) {
            Toast.makeText(this, R.string.toast_nfc_not_available,
Toast.LENGTH_LONG).show();
        } else {
            mPendingIntent = PendingIntent.getActivity(this, 0,
                    new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
            IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
            try {
                ndef.addDataType("text/plain");
            } catch (IntentFilter.MalformedMimeTypeException e) {
                throw new RuntimeException("fail", e);
            }
            mIntentFilters = new IntentFilter[]{ndef,};
        }
        if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {}
    }
    @Override
    protected void onNewIntent(Intent intent) {
        setIntent(intent);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.action_db) {
            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.content_frame, new TestDbFragment()).commit();
            page = 0;
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onFragmentInteraction(Uri uri) {}

    @Override
    protected void onResume() {
        super.onResume();
        if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
            processIntent(getIntent());
        }
        if (mAdapter != null)
            mAdapter.enableForegroundDispatch(this, mPendingIntent, mIntentFilters,
null);
        EventBus.getDefault().register(this);
    }

    void processIntent(Intent intent) {
        Parcelable[] rawMsgs =
intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        if (rawMsgs != null) {
            NdefMessage[] msgs = new NdefMessage[rawMsgs.length];
            for (int i = 0; i < rawMsgs.length; i++) {
                msgs[i] = (NdefMessage) rawMsgs[i];
                String msg = new String(msgs[i].getRecords()[0].getPayload());
                if (!msg.equals("\u0003me.freetymekiyan.smartring")) {
                    if (enabled) {
                        if (count <= 10) {
                            float rate = Float.valueOf(msg.substring(START_INDEX,
START_INDEX + 4))
                                    * 8192 / 8000000;
                            sum += 60 / rate;
                            count++;
                            Toast.makeText(this, "Current Pulse: " + 60 / rate,
Toast.LENGTH_SHORT)
                                    .show();
                        } else {
                            EventBus.getDefault().post(new MeasureEvent((int) sum /
count));
                            result = sum / count;
                        }
                    }
                }
            }
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (mAdapter != null)
            mAdapter.disableForegroundDispatch(this);
        EventBus.getDefault().unregister(this);
    }

    @Override
    public void onMeasureStateChanged(boolean enabled) {
        this.enabled = enabled;
        if (enabled == true) {
            sum = 0;
            count = 0;
        }
    }

    public void onEvent(PrefChangedEvent event) {
        switch (event.prefKey) {
            case R.string.key_name:
                mDrawerAdapter.setName(event.newValue);
                break;
            case R.string.key_email:
                mDrawerAdapter.setEmail(event.newValue);
                break;
        }
        mDrawerAdapter.notifyItemChanged(0);
    }
}
```

AnalysisFragment.java

```
package me.freetymekiyan.smartring.views;

public class AnalysisFragment extends Fragment {

    private static AnalysisFragment instance;

    public AnalysisFragment() {}

    public static AnalysisFragment getInstance() {
        if (instance == null) {
            instance = new AnalysisFragment();
        }
        return instance;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.fragment_report_wrapper, container,
false);
        MeasurePagerAdapter mPagerAdapter = new MeasurePagerAdapter(
                getChildFragmentManager());
        ViewPager mViewPager = (ViewPager) view.findViewById(R.id.pager);
        mViewPager.setAdapter(mPagerAdapter);
        return view;
    }

    public class MeasurePagerAdapter extends FragmentPagerAdapter {

        public MeasurePagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public Fragment getItem(int position) {
            if (position == 0) {
                return (Fragment)
HistoryFragment.newInstance(getString(R.string.history));
            } else {
                return (Fragment)
ReportFragment.newInstance(getString(R.string.report));
            }
        }

        @Override
        public int getCount() {
            return 2;
        }

        @Override
        public CharSequence getPageTitle(int position) {
            return
getItem(position).getArguments().getString(MainActivity.KEY_TITLE);
        }
    }
}
```

MeasureOneFragment.java

```
package me.freetymekiyan.smartring.views;

public class MeasureOneFragment extends Fragment implements View.OnClickListener {

    private RippleBackground rippleBkg;

    private MySqlDbHelper mDbHelper;

    private OnMeasureListener mListener;

    public interface OnMeasureListener {
        public void onMeasureStateChanged(boolean enabled);
    }

    public static MeasureOneFragment newInstance() {
        MeasureOneFragment fragment = new MeasureOneFragment();
        return fragment;
    }

    public MeasureOneFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mDbHelper = new MySqlDbHelper(getActivity());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.fragment_measure_one, container,
false);
        rippleBkg = (RippleBackground) view.findViewById(R.id.rbkg_measure);
        ImageView imageView = (ImageView) view.findViewById(R.id.centerImage);
        imageView.setOnClickListener(this);
        return view;
    }

    public void onNFCStateChanged(boolean enabled) {
        if (mListener != null) {
            mListener.onMeasureStateChanged(enabled);
        }
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnMeasureListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                    + " must implement OnFragmentInteractionListener");
        }
        EventBus.getDefault().register(this);
    }

    public void onEvent(final MeasureEvent event) {
        stopUpdateNfc();
        final MaterialDialog.Builder builder = new
MaterialDialog.Builder(getActivity());
        builder.title(R.string.dialog_state_title)
                .content(R.string.dialog_state_content, event.result)
                .positiveText(R.string.active)
                .negativeText(R.string.rest)
                .neutralText(android.R.string.cancel)
                .callback(new MaterialDialog.ButtonCallback() {
                    @Override
                    public void onPositive(MaterialDialog dialog) {
                        super.onPositive(dialog);
                        mDbHelper.addPulseRate(event.result,
Pulse.State.ACTIVE.ordinal());
                    }

                    @Override
                    public void onNegative(MaterialDialog dialog) {
                        super.onNegative(dialog);
                        mDbHelper.addPulseRate(event.result,
Pulse.State.REST.ordinal());
```

```
                    }

                    @Override
                    public void onNeutral(MaterialDialog dialog) {
                        super.onNeutral(dialog);
                    }
                })
                .show();
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mListener = null;
        EventBus.getDefault().unregister(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.centerImage:
                if (isReceiving()) {
                    stopUpdateNfc();
                } else {
                    startUpdateNfc();
                }
                break;
            default:
                break;
        }
    }

    private void stopUpdateNfc() {
        rippleBkg.stopRippleAnimation();
        Toast.makeText(getActivity(), R.string.toast_stop_reading, 
Toast.LENGTH_SHORT).show();
        onNFCStateChanged(false);
    }

    private void startUpdateNfc() {
        rippleBkg.startRippleAnimation();
        Toast.makeText(getActivity(), R.string.toast_start_reading, 
Toast.LENGTH_SHORT).show();
        onNFCStateChanged(true);
    }

    private boolean isReceiving() {
        return rippleBkg.isRippleAnimationRunning();
    }
}
```

HistoryFragment.java

```
package me.freetymekiyan.smartring.views;

public class HistoryFragment extends Fragment implements 
OnChartValueSelectedListener {

    private static final String X_VAL_FORMAT = "MM/dd";
    private BarChart mChart;

    private MySqlDbHelper db;

    public HistoryFragment() {}

    public static HistoryFragment newInstance(String title) {
        HistoryFragment fragment = new HistoryFragment();
        Bundle args = new Bundle();
        args.putString(MainActivity.KEY_TITLE, title);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        db = new MySqlDbHelper(getActivity());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_history, container, false);
        mChart = (BarChart) view.findViewById(R.id.chart_history);
        mChart.setOnChartValueSelectedListener(this);
        mChart.setDrawLegend(true);
        mChart.setDescription("");
        mChart.setDrawYValues(false);
        mChart.setPinchZoom(true);
        mChart.setDrawBarShadow(false);
        mChart.setDrawGridBackground(false);
        mChart.setDrawHorizontalGrid(false);
        mChart.setDrawYValues(true);
        generateDataSet();
        mChart.animateXY(1500, 1500);
        return view;
    }

    /**
     * Read last 7 days data from database
     */
    private void generateDataSet() {
        ArrayList<String> xVals = new ArrayList<String>();
        ArrayList<BarEntry> rest = new ArrayList<BarEntry>();
        ArrayList<BarEntry> active = new ArrayList<BarEntry>();

        final Calendar c = Calendar.getInstance();
        final Calendar c2 = Calendar.getInstance();
        c.clear();
        c.set(c2.get(Calendar.YEAR), c2.get(Calendar.MONTH),
c2.get(Calendar.DAY_OF_MONTH));
        c.add(Calendar.DATE, -6); // 7 days ago
        final DateFormat format = new SimpleDateFormat(X_VAL_FORMAT,
Locale.ENGLISH);

        final List<Pulse> rawRest = db.getLast7DaysRest();
        final List<Pulse> rawActive = db.getLast7DaysActive();
        for (int i = 0; i < 7; i++) {
            xVals.add(format.format(c.getTime()));

            Pulse p = new Pulse();
            p.setDate(c.getTime());
            if (rawRest.contains(p)) {
                rest.add(new BarEntry(rawRest.get(rawRest.indexOf(p)).getValue(),
i));
            } else {
                rest.add(new BarEntry(0, i));
            }
            if (rawActive.contains(p)) {
                active.add(new
BarEntry(rawActive.get(rawActive.indexOf(p)).getValue(), i));
            } else {
                active.add(new BarEntry(0, i)); // insert 0 if no measurement for
that day
            }

            c.add(Calendar.DATE, 1);
        }

        BarDataSet set1 = new BarDataSet(rest, getString(R.string.rest));
        set1.setColor(getResources().getColor(R.color.BlueBkg));
        BarDataSet set2 = new BarDataSet(active, getString(R.string.active));
        set2.setColor(getResources().getColor(R.color.AccentColor));

        ArrayList<BarDataSet> dataSets = new ArrayList<BarDataSet>();
        dataSets.add(set1);
        dataSets.add(set2);

        BarData data = new BarData(xVals, dataSets);
        data.setGroupSpace(110f);

        LimitLine lowerLimit = new LimitLine(60f);
        lowerLimit.setDrawValue(false);
        lowerLimit.setLineColor(getResources().getColor(R.color.PrimaryColor));
        lowerLimit.setLineWidth(0.5f);
        lowerLimit.enableDashedLine(12f, 2f, 0f);
        data.addLimitLine(lowerLimit);

        LimitLine lowerLimit2 = new LimitLine(90f);
        lowerLimit2.setDrawValue(false);
        lowerLimit2.setLineColor(getResources().getColor(R.color.PrimaryColor));
        lowerLimit2.setLineWidth(0.5f);
        lowerLimit2.enableDashedLine(12f, 2f, 0f);
        data.addLimitLine(lowerLimit2);

        final float v = getUpperLimitValue();
        if (v != 0) {
            LimitLine upperLimit = new LimitLine(v);
            upperLimit.setLineColor(getResources().getColor(R.color.PrimaryColor));
            upperLimit.setLineWidth(0.5f);
            upperLimit.enableDashedLine(12f, 2f, 0f);
            upperLimit.setLabelPosition(LimitLine.LimitLabelPosition.LEFT);
            data.addLimitLine(upperLimit);
        }
        mChart.setData(data);
    }

    private float getUpperLimitValue() {
        final SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(getActivity());
        String age = sp.getString(getString(R.string.key_age), "");
        boolean gender = sp.getBoolean(getString(R.string.key_gender), true);
        if (age.isEmpty()) return 0f;
        return gender ? getMaleHr(age) : getFemaleHr(age);
    }

    /**
     * Gulati method, for women, 2010
     * @param age
     * @return
     */
    private float getFemaleHr(String age) {
        return 0.9f * (206 - 0.88f * Integer.valueOf(age));
    }

    /**
     * Least objectionable formula
     * @param age
     * @return
     */
    private float getMaleHr(String age) {
        return 0.9f * (205.8f - 0.685f * Integer.valueOf(age));
    }

    /**
     * Generate test data for the graph
     */
    private void generateTestDataSet() {
        ArrayList<String> xVals = new ArrayList<String>();
        for (int i = 0; i < 7; i++) {
            xVals.add(i + "");
        }

        ArrayList<BarEntry> yVals1 = new ArrayList<BarEntry>();
        ArrayList<BarEntry> yVals2 = new ArrayList<BarEntry>();
        Random r = new Random();
        for (int i = 0; i < 7; i++) {
            yVals1.add(new BarEntry(50 + r.nextInt(41), i));
            yVals2.add(new BarEntry(120 + r.nextInt(41), i));
        }

        BarDataSet set1 = new BarDataSet(yVals1, "Static");
        set1.setColor(getResources().getColor(R.color.BlueBkg));
        BarDataSet set2 = new BarDataSet(yVals2, "Workout");
        set2.setColor(getResources().getColor(R.color.AccentColor));

        ArrayList<BarDataSet> dataSets = new ArrayList<BarDataSet>();
        dataSets.add(set1);
        dataSets.add(set2);

        BarData data = new BarData(xVals, dataSets);
        data.setGroupSpace(110f);
        mChart.setData(data);
    }

    @Override
    public void onValueSelected(Entry e, int dataSetIndex) {}

    @Override
    public void onNothingSelected() {}
}
```

ReportFragment.java

```
package me.freetymekiyan.smartring.views;

public class ReportFragment extends Fragment {

    public static ReportFragment newInstance(String title) {
        ReportFragment fragment = new ReportFragment();
        Bundle args = new Bundle();
        args.putString(MainActivity.KEY_TITLE, title);
        fragment.setArguments(args);
        return fragment;
    }

    public ReportFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_report, container, false);
        final TableLayout tlReport = (TableLayout)
view.findViewById(R.id.tl_report);
        return view;
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
    }

    @Override
    public void onDetach() {
        super.onDetach();
    }
}
```

MySqlDbHelper.java

```
package me.freetymekiyan.smartring.models;

public class MySqlDbHelper extends SQLiteOpenHelper {

    public static final String DB_NAME = "SmartRing.db";

    public static final int VER = 2;

    private static final String SQL_CREATE_TABLE = "CREATE TABLE "
            + PulseEntry.TABLE_NAME + "("
            + PulseEntry._ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + PulseEntry.COL_NAME_VAL + " INTEGER, "
            + PulseEntry.COL_NAME_STATE + " INTEGER, "
            + PulseEntry.COL_NAME_MEASURED_DATE + " DATETIME DEFAULT
(date('now','localtime')), "
            + PulseEntry.COL_NAME_MEASURED_TIMESTAMP
            + " DATETIME DEFAULT (datetime('now','localtime'))" +
            ")";
    private static final String SQL_DELETE_TABLE = "DROP TABLE IF EXISTS " +
PulseEntry.TABLE_NAME;

    public MySqlDbHelper(Context context) {
        super(context, DB_NAME, null, VER);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL(SQL_DELETE_TABLE);
    }

    @Override
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }

    public long addPulseRate(int rate, int state) {
        return addPulseRateWithDate(rate, state, null);
    }

    public long addPulseRateWithDate(int rate, int state, Date date) {
        SQLiteDatabase db = getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(PulseEntry.COL_NAME_VAL, rate);
        values.put(PulseEntry.COL_NAME_STATE, state);
        if (date != null) {
            final SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd",
Locale.ENGLISH);
            values.put(PulseEntry.COL_NAME_MEASURED_DATE, df.format(date));
            final SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss",
Locale.ENGLISH);
            values.put(PulseEntry.COL_NAME_MEASURED_TIMESTAMP, df2.format(date));
        }
        // Insert the new row, returning the primary key value of the new row
        long rowId = db.insert(PulseEntry.TABLE_NAME, null, values);
        if (rowId == -1) {
            Log.d("DEBUG", "addPulseRate failed");
        }
        return rowId;
    }

    public List<Pulse> getLast7DaysRest() {
        return getLast7Days(Pulse.State.REST);
    }

    public List<Pulse> getLast7DaysActive() {
        return getLast7Days(Pulse.State.ACTIVE);
    }

    public List<Pulse> getLast7Days(Pulse.State s) {
        List<Pulse> res = new ArrayList<Pulse>();
        SQLiteDatabase db = getReadableDatabase();

        String[] projection = {
                PulseEntry.COL_NAME_MEASURED_DATE,
                "AVG(" + PulseEntry.COL_NAME_VAL + ")" + " AS " +
PulseEntry.COL_NAME_AVG_VAL,
        };
        String selection = PulseEntry.COL_NAME_STATE + "=?";
        String[] selectionArgs = new String[]{s.ordinal() + ""};
        String groupBy = PulseEntry.COL_NAME_MEASURED_DATE;
        String having = PulseEntry.COL_NAME_MEASURED_DATE
                + " >= DATE('now', '-7 days')";
        String orderBy = PulseEntry.COL_NAME_MEASURED_DATE + " ASC";

        Cursor c = db.query(PulseEntry.TABLE_NAME, projection, selection,
selectionArgs, groupBy,
                having, orderBy);
        c.moveToFirst();
        while (!c.isAfterLast()) {
            String date =
c.getString(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_MEASURED_DATE));
            int value =
c.getInt(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_AVG_VAL));
            Pulse p = new Pulse(value, date, Pulse.State.REST);
            res.add(p);
            c.moveToNext();
        }
        if (c != null && !c.isClosed()) c.close();
        return res;
    }

    public String getAllPulses() {
        SQLiteDatabase db = getWritableDatabase();
        String sortOrder = PulseEntry.COL_NAME_MEASURED_DATE + " ASC";
        Cursor c = db.query(PulseEntry.TABLE_NAME, null, null, null, null, null,
sortOrder);
        c.moveToFirst();
        StringBuilder res = new StringBuilder();
        while (!c.isAfterLast()) {
            long id = c.getLong(c.getColumnIndexOrThrow(PulseEntry._ID));
            int rate = c.getInt(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_VAL));
            String state =
c.getInt(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_STATE)) == 0
                    ? "static" : "workout";
            String date = c
                    .getString(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_MEASURED_
DATE));
            String time = c
                    .getString(c.getColumnIndexOrThrow(PulseEntry.COL_NAME_MEASURED_
TIMESTAMP));
            res.append(id);
            res.append('\t');
            res.append(rate);
            res.append('\t');
            res.append(state);
            res.append('\t');
            res.append(date);
            res.append('\t');
            res.append(time);
            res.append('\t');
            res.append('\n');
            c.moveToNext();
        }
        if (c != null && !c.isClosed()) c.close();
        return res.toString();
    }

    public void addLast7Days() {
        final Random r = new Random();
        final Calendar c = Calendar.getInstance();
        final Calendar c2 = Calendar.getInstance();
        c.clear();
        c.set(c2.get(Calendar.YEAR), c2.get(Calendar.MONTH),
c2.get(Calendar.DAY_OF_MONTH));
        c.add(Calendar.DATE, -6);
        for (int i = 0; i < 7; i++) {
            int rate = 50 + r.nextInt(41);
            addPulseRateWithDate(rate, Pulse.State.REST.ordinal(), c.getTime());
            rate = 120 + r.nextInt(41);
            addPulseRateWithDate(rate, Pulse.State.ACTIVE.ordinal(), c.getTime());
            c.add(Calendar.DATE, 1);
        }
    }
}
```

MyPreferenceFragment.java

```
package me.freetymekiyan.smartring.views;

public class MyPreferenceFragment extends PreferenceFragment implements
        SharedPreferences.OnSharedPreferenceChangeListener {

    public static final long INTERVAL_WEEK = AlarmManager.INTERVAL_DAY * 7;

    public static final long INTERVAL_MONTH = INTERVAL_WEEK * 4;

    public static final long[] INTEVALS = {AlarmManager.INTERVAL_DAY, INTERVAL_WEEK,
            INTERVAL_MONTH};

    public MyPreferenceFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.pref_general);

getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListene
r(this);
    }

    @Override
    public void onPause() {
        super.onPause();

getPreferenceScreen().getSharedPreferences().unregisterOnSharedPreferenceChangeListe
ner(
                this);
    }

    @Override
    public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        Preference prefAbout = findPreference(getString(R.string.key_about));
        prefAbout.setOnPreferenceClickListener(new
Preference.OnPreferenceClickListener() {

            @Override
            public boolean onPreferenceClick(Preference preference) {
                new MaterialDialog.Builder(getActivity())
                        .title(R.string.about)
                        .content(R.string.about_content)
                        .positiveText(android.R.string.ok)
                        .show();
                return true;
            }
        });
    }

    @Override
    public void onResume() {
        super.onResume();
        for (int i = 0; i < getPreferenceScreen().getPreferenceCount(); i++) {
            Preference pref = getPreferenceScreen().getPreference(i);
            if (pref instanceof PreferenceGroup) {
                PreferenceGroup preferenceGroup = (PreferenceGroup) pref;
                for (int j = 0; j < preferenceGroup.getPreferenceCount(); j++) {
                    updatePreference(preferenceGroup.getPreference(j), false);
                }
            } else {
                updatePreference(pref, false);
            }
        }
    }

    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
String key) {
        updatePreference(findPreference(key), true);
    }

    void updatePreference(Preference pref, boolean fromUser) {
        if (pref instanceof ListPreference) {
            ListPreference listPref = (ListPreference) pref;
            listPref.setSummary(
                    listPref.getEntry() != null ? listPref.getEntry() :
getString(R.string.daily));
            if (fromUser) setRecurringAlarm(getActivity(), listPref.getValue());
        }
        if (pref instanceof EditTextPreference) {
            String title = pref.getTitle().toString();
            String text = ((EditTextPreference) pref).getText();
            if (title.equals(getString(R.string.name))) {
                pref.setSummary(
                        (text == null || text.isEmpty()) ?
getString(R.string.name_summary) : text);
                EventBus.getDefault().post(
                        new PrefChangedEvent(pref.getSummary() + "",
R.string.key_name));
            } else if (title.equals(getString(R.string.weight))) {
                pref.setSummary(
                        (text == null || text.isEmpty()) ?
getString(R.string.weight_summary)
                                : text + " " + getString(R.string.weight_unit));
            } else if (title.equals(getString(R.string.age))) {
                pref.setSummary(
                        (text == null || text.isEmpty()) ?
getString(R.string.age_summary) : text);
            } else if (title.equals(getString(R.string.email))) {
                pref.setSummary(text == null || text.isEmpty() ?
getString(R.string.email_summary) : text);
```

```
                EventBus.getDefault()
                        .post(new PrefChangedEvent(pref.getSummary() + "",
R.string.key_email));
            }
        }
        if (pref instanceof SwitchPreference) {
            pref.setSummary(
                    ((SwitchPreference) pref).isChecked() ? ((SwitchPreference)
pref).getSummaryOn()
                            : ((SwitchPreference) pref).getSummaryOff());
        }
        if (pref instanceof TimePickerPreference) {
            TimePickerPreference t = (TimePickerPreference) pref;
            if (fromUser) setRecurringAlarm(getActivity(), t.getHour(),
t.getMinute());
        }
    }

    private void setRecurringAlarm(Context context, int hour, int min) {
        final String intervalVal = getPreferenceScreen().getSharedPreferences()
                .getString(getString(R.string.key_frequency),
getString(R.string.freq_daily));
        setAlarm(getActivity(), hour, min, INTEVALS[Integer.valueOf(intervalVal)]);
    }

    private void setRecurringAlarm(Context context, String value) {
        long interval = INTEVALS[Integer.valueOf(value)];
        final String time = getPreferenceScreen().getSharedPreferences().getString(
                getString(R.string.key_time), TimePickerPreference.DEFAULT_VALUE);
        setAlarm(getActivity(), TimePickerPreference.getHour(time),
                TimePickerPreference.getMinute(time), interval);
    }

    private void setAlarm(Context context, int hour, int min, long interval) {
        Calendar c = Calendar.getInstance();
        c.clear(Calendar.SECOND);
        c.set(Calendar.HOUR_OF_DAY, hour);
        c.set(Calendar.MINUTE, min);
        if (c.getTimeInMillis() < System.currentTimeMillis()) {
            c.set(Calendar.DAY_OF_YEAR, c.get(Calendar.DAY_OF_YEAR) + 1);
        }

        PendingIntent pi = PendingIntent.getBroadcast(context,
                0, new Intent(context, AlarmReceiver.class),
PendingIntent.FLAG_UPDATE_CURRENT);
        AlarmManager alarms = (AlarmManager)
getActivity().getSystemService(Context.ALARM_SERVICE);
        alarms.setRepeating(AlarmManager.RTC_WAKEUP, c.getTimeInMillis(), interval,
pi);
    }
}
```

AlarmReceiver.java

```
package me.freetymekiyan.smartring.receivers;

public class AlarmReceiver extends BroadcastReceiver {

    public AlarmReceiver() {}
    @Override
    public void onReceive(Context context, Intent intent) {
        Bitmap icon = BitmapFactory.decodeResource(context.getResources(),
R.drawable.ic_big_noti);
        NotificationCompat.Builder mBuilder =
                new NotificationCompat.Builder(context)
                        .setSmallIcon(R.drawable.ic_notification)
                        .setLargeIcon(icon)
                        .setContentTitle(context.getString(R.string.noti_title))
                        .setAutoCancel(true)
                        .setDefaults(Notification.DEFAULT_ALL)
                        .setContentText(context.getString(R.string.noti_text));
        Intent resultIntent = new Intent(context, MainActivity.class);

        TaskStackBuilder stackBuilder = TaskStackBuilder.create(context);
        stackBuilder.addParentStack(MainActivity.class);
        stackBuilder.addNextIntent(resultIntent);
        PendingIntent resultPendingIntent =
                stackBuilder.getPendingIntent(
                        0,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );
        mBuilder.setContentIntent(resultPendingIntent);
        mBuilder.setAutoCancel(true);
        NotificationManager mNotificationManager =
                (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
        mNotificationManager.notify(0, mBuilder.build());
    }
}
```