Augustine Onubeze

Developing a Wireless Heart Rate Monitor with MAX30100 and nRF51822

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Thesis
31 October 2016

Helsinki Metropolia University of Applied Sciences

Author	Augustine Onubeze
Title	Developing a Wireless Heart Rate Monitor with MAX30100 and nRF51822
Number of pages	42 pages
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Software Engineering
Instructor	Kimmo Sauren, Principal Lecturer

The purpose of this project was to develop a wireless heart rate monitor based on the principle of photoplethysmography and using the Bluetooth Low Energy (BLE) technology for wireless transfer. The emphasis here was not on the hardware prototype but mainly on the software that drives the sensor and a compatible client mobile application.

The mobile client application is an Android application developed for smart phones powered by Android version 4.4.3 or higher. It enables the Android device to communicate with the sensor to recieve PPG data, process, store and display results. In this project, the mobile device used during the development and testing was an LG Nexus 5 smart phone.

The heart rate monitor developed in this project has two main units, the sensor unit, and the mobile client application. The sensor unit consists of two main components: a heart rate sensor integrated circuit (IC), max30100, which is a complete pulse oximeter and heart-rate sensor solution from Maxim Integrated, and a BLE chip, nRF51822, which is a multiprotocol system on chip for Bluetooth smart and 2.4 GHz ultra low-power wireless applications from Nordic Semiconductors. Both components work as a unit to collect photoplethysmograph (PPG) samples and stream them through an emulated UART/Serial port over BLE to a connected mobile device.

The result of this project includes a laboratory setup for a finger-based heart rate sensor, a firmware for the sensor, and an Android mobile application that works with the sensor. The sensor setup could be further developed into a finger wearable heart rate monitor prototype that can run the same firmware using MAX30100 and nRF51822. With some modifications in the firmware and the client application, the result of this project could also be further developed into a prototype of an oximeter that would also measure blood oxygen saturation.

Keywords	BLE, oxygen saturation, photoplethysmography, MAX30100, nRF51822, Android, UART service, oximeter, ultra low power, wearable

Contents

1 Introduction

The development of wearable technologies and Bluetooth low energy (BLE) aka Bluetooth smart has brought various innovations in health care and monitoring. Certain health monitoring and measurement devices such as Electrocardiography (ECG) devices, oximeters, and heart rate monitors, used to be confined mainly in the intensive care units of hospitals and clinics, and in some specialized hospital wards and laboratories. These devices are usually complex, expensive and largely nonportable. With the advent of wearables and BLE, monitoring and measurement of some vital signs in the body could be done in the comfort of one's bedroom, in the gym while working out, or even by athletes in the field while training.

Various sensors for monitoring and measuring some vital signs are incorporated into wearable devices that are very portable, can operate with very low energy, are affordable, and very easy to use. Some of these wearables are self-contained or stand-alone, having a in-built processing and control unit, and a display unit, while some others depend on smart phones for control, processing and output (display) and mostly utilize BLE for data communication with mobile devices.

The aim of this project was to develop a photoplethysmography-based heart-rate monitor that would communicate wirelessly with Android-powered smart devices to measure PPG and heart rate from a subject's finger. It would use max30100, a pulse oximeter and heart-rate sensor solution from Maxim Integerated, as the sensor IC; nRF51822, a BLE chip from Nordic Semiconductors; and the Android platform, for wireless control, processing and display.

2 Principles and Technologies

2.1 Photoplethysmography

Plethysmography refers to the measurement of changes in the volume of an organ of the body. These changes are usually due to changes in the blood flow to and from the organ, but could also be due to changes in the air flow. Photoplethysmography (PPG), therefore, is a form of plethysmography that uses optics to measure changes in the blood volume in the microvascular bed of body tissue. It is a non-invasive, simple and affordable technique of detecting the changes in the amount of blood in a body tissue. [1, 2.]

In PPG, a well perfused surface of the skin is illuminated by light from a light-emitting-diode (LED) preferably, infra red or red LED. The light transmitted or reflected by the skin surface, collected by means of a photodiode is then used to determine changes in blood volume. The photodiode is placed either on the other side of the skin surface to detect the transmitted light, or on the same side as the light source to detect the transmitted light.

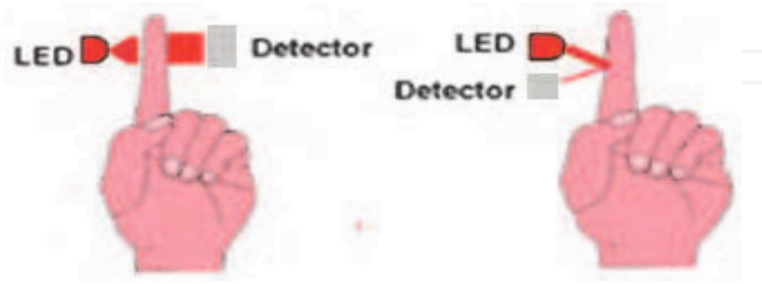


Figure 1: Operation of the pulse oximeter utilizing PPG to measure blood oxygen saturation. Copied from Omar and Armin [2, 160].

Figure 1 above shows the operation of a pulse oximeter. On the left is a depiction of transmission pulse oximetry and on the right is a depiction of reflection pulse oximetry. As shown in the figure, both the LED and the photodetector are placed on the same side of the finger in the reflection pulse oximetry while the LED and the photodetector are placed on the opposite sides of the finger in the transmission pulse oximetry. The PPG waveform consists of two components, the pulsatile AC component, which is attributed to the cardiac synchronous changes in the arterial blood volume per heart

beat, and the slowly varying or relatively constant DC component, which is attributed to respiration, sympathetic nervous system activity and thermoregulation [3].

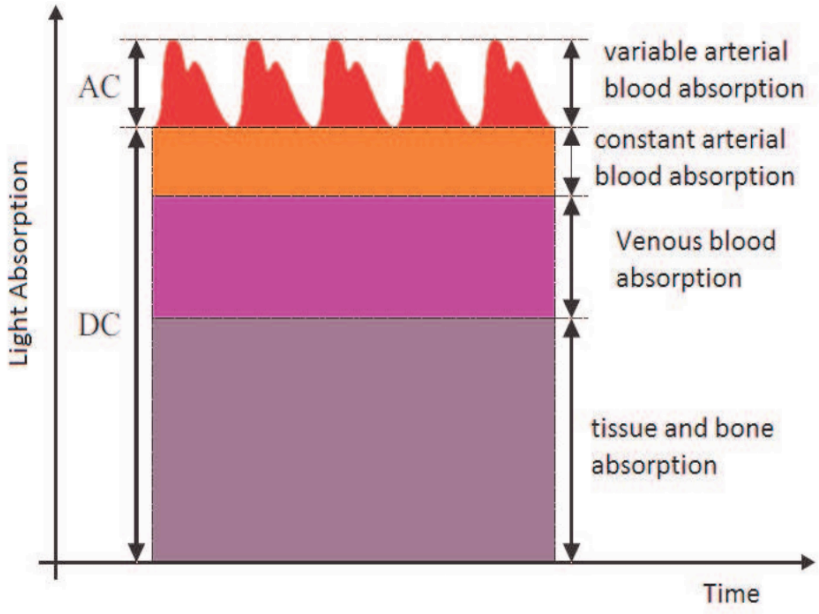


Figure 2: Light absorption by different tissues showing the PPG waveform based on arterial blood absortion at the top [2, 160].

Figure 2 above shows the PPG waveform with its AC and DC components. While the AC component represents variable arterial blood absorption, the DC component represents relatively constant blood absorption by the arteries, veins, tissues and bones. These two components are relevant for determining the amount of oxygen in the blood but only the AC component is needed for calculating the heart rate, where each spike in the wave represents a single heart beat. PPG is applied in the development of various commercially available health monitoring and measuring devices such as oximeter, for measuring blood oxygen saturation, heart rate meter for heart rate measurement, and blood pressure monitor.[3]

2.2 Bluetooth Low Energy

Bluetooth low energy (BLE), also known as Bluetooth Smart, is a wireless personal area network technology designed and marketed by Bluetooth Special Interest Group. It is part of the Bluetooth Core Specification from Bluetooth version 4.0 onwards built for the Internet of Things (IoT) and widely known for its ultra-low power consumption and low cost.[22]

Originally introduced by Nokia in 2006 with the brand name Wibree, BLE was developed as a solution to some scenarios that were not addressed by other wireless technologies. In their bid to address these scenarios, Nokia's researchers developed a wireless technology adapted from the Bluetooth standard, which is much more power-efficient and cost-effective, and yet not very different from the Bluetooth technology. Wibree was later included in the Bluetooth specification as a Bluetooth ultra-low-power technology and subsequently integrated with version 4.0 of the Bluetooth Core Specification in 2010. [4]

Key Features

The most outstanding feature of BLE is its ultra-low peak, average and idle mode power consumption. With this feature, it is possible to power a BLE device with a single standard coin cell battery for years. Other key features of BLE include:

- Standardized application development architecture which helps reduce the cost of development and operation.
- Interoperability across multiple vendors.
- Enhanced range
- Robust, secure and efficient.

The BLE technology creates the possibility of enhancing the most basic electronic devices like toothbrushes, toys, or key rings with the Bluetooth wireless technology, and incorporating new functionalities into devices that are already Bluetooth-enabled. [5]

BLE Operation

The BLE radio operates in the 2.4 GHz ISM band split into 40 channels separated by 2 MHz. Three of the channels are for advertising while the rest (37) are used as data channels. To combat interference and fading, BLE employs a frequency hopping transceiver, and uses a shaped, binary frequency modulation to minimize transceiver complexity. With a symbol rate of 1 megasymbol per second, it supports a bit rate of up to 1 megabit per second. Data is transmitted between BLE devices in packets positioned in time units known as events, which can either be Advertising or Connection event. Devices that transmit advertising packets are known as advertisers while those that receive advertising packets with no intention of connecting to the advertiser are known as scanners. [6, 20.]

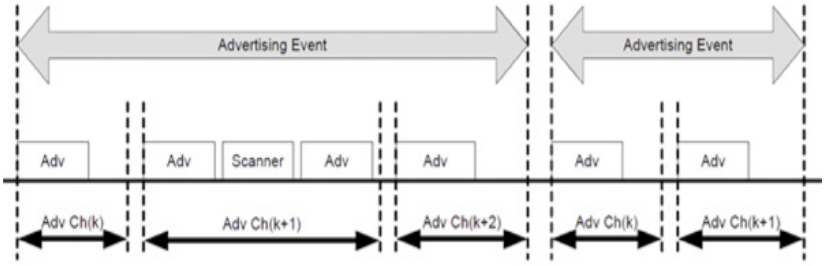


Figure 3: Advertising Events. Copied from Bluetooth SIG [6, 21].

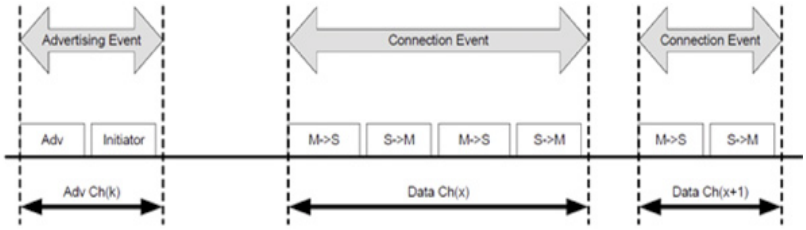


Figure 4: Connection Events. Copied from Bluetooth SIG [6, 21].

When there is the need for a connection between two BLE devices, one of the devices, known as the initiator, initiates the connection by making a connection request to the other which plays the role of the advertiser. Once a connection is established, the initiator becomes the master while the advertiser becomes the slave in what is known as a piconet. Data packets are transmitted between two connected BLE devices within the connection events. The master initiates a connection event and can end it at any time.[6, 20.] Figure 3 shows advertising events which occur at a specified interval called an advertisement interval. Figure 4, on the hand, shows connection events and the transition from advertising to connection. An advertiser stops advertising once a connection has been initiated.

BLE Application Architecture

In BLE, as well as other Bluetooth versions, application interoperability is realized using what is referred to as profiles. BLE profiles define the functions and features of each layer in the stack, the vertical communication between the layers in a device, the

horizontal communication between specific layers of different devices, data formats, and application behaviors. [6, 93.]

Generic Access Profile

All BLE devices implement a base profile known as the Generic Access Profile (GAP), which defines the basic requirements of a BLE device such as the Physical Layer, Link Layer, L2CAP, Security Manager, Attribute Protocol (ATT) and Generic Attribute Profile (GATT). GAP, in BLE, defines four specific roles: Broadcaster role for transmitter only applications, Observer role for receiver only applications, Peripheral role for less complex devices supporting only single connection, and Central role for devices that initiate all connections and can support multiple connections and more complex functions. Any BLE device can support multiple roles as long as its underlying controller supports them. However, only one role may be supported at any given time. [6, 93.]

Generic Attribute Profile

Generic Attribute Profile (GATT) defines how profile and user data are exchanged over a BLE connection. It also provides a framework that serves as a reference for all GAT-based profiles, defined for specific use cases, and ensures interoperability between devices from different manufacturers. GATT defines two roles: Client and Server roles. The client sends requests to a server and in turn receives responses from it. The server, on the other hand, receives clients' requests and responds to them. [6, 93.]
GATT Profile specifies a hierarchical structure for data exchanged over BLE. In this structure, a profile is at the topmost level of the hierarchy. A profile is made up of one or more conceptually related services required to realize a use case. A service consists of one or more characteristics or references to other services. A characteristic serves as a container for user data and can also contain information about the data known as descriptors. [6, 95-97.]

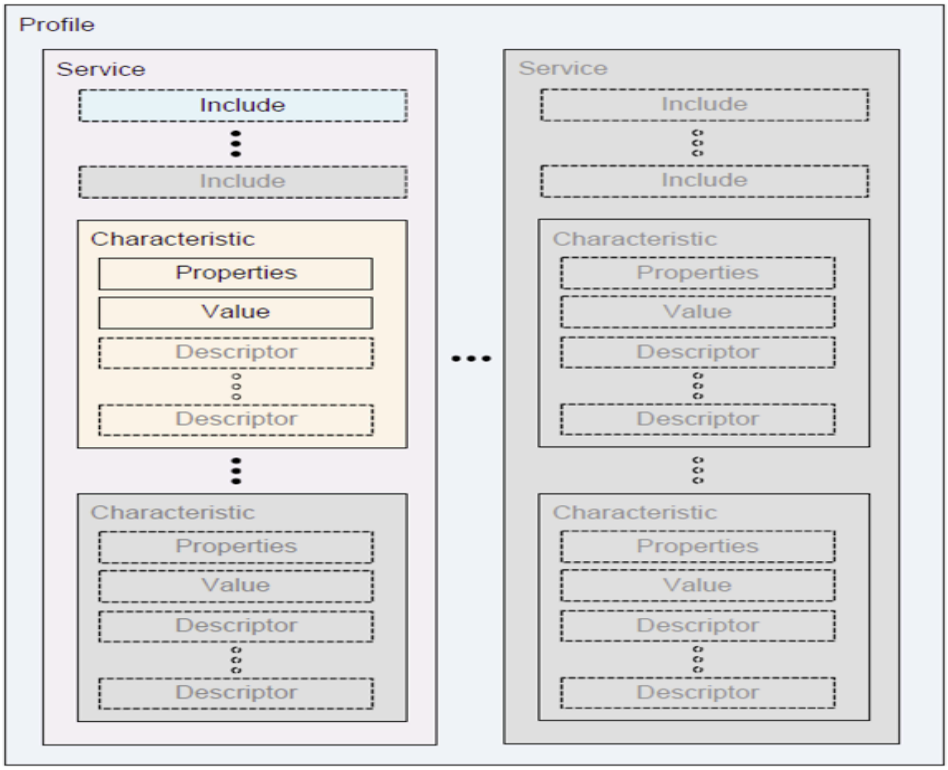


Figure 5: GATT-Based Profile Hierarchy. Copied from Bluetooth SIG [6, 96.]

Figure 5 describes the Generic Attribute Profile hierarchical organisation. From the figure, a profile is the highest in the hierarchy, and consists of one or more services. Each service has at least one characteristic and each characteristic has properties, descriptors and holds a value.

2.3 Android System

Android is an open-source software stack based on the Linux kernel that ships with operating system, middleware, native mobile applications, as well as sets of API libraries for third party applications development [7]. It was primarily designed for touchscreen mobile devices like smartphones and tablets but has, over the years, yielded some variants designed to power televisions, digital cameras, auto multimedia systems, notebooks, and game consoles. Android system was originally developed by Android, Inc., a Carlifornian based start-up founded in October 2003, which was later acquired by Google in 2005. Android was launched, in November 2007, as the first product of a consortium of technology companies led by Google called Open Handset Alliance (OHA) whose main goal was to develop open standards for mobile devices. Further development of Android eventually became the responsibility of OHA. [7]

Basic Features

A number of basic but very significant features of Android make it stand out among other mobile platforms. These features made Android the fastest growing mobile platform since it was launched. Thanks to these features, Android has the largest market share in the smartphone OS market. As of June 2016, Android's market share based on worldwide smartphone sales to end users is set at 85.2%. iOS, which is the second largest has only 13.8%. [8] Some of these features include open, equal status for all applications, seamless connectivity between applications, fast and easy application development.

Android was developed in such a way as to give developers the full advantage of the power of any mobile device. An Android developer can incorporate a device's core functionality such as taking pictures, video recording, text messaging, and making calls, into his/her own application to create very compelling mobile applications for richer and more cohesive experiences for users. Android is also open source, which means that it could be further customized by device manufacturers or even technology-minded users to suit varied needs and requirements. [9]

There is no difference between a phone's core applications and other third-party applications. They all share the same status and can be substituted one for the other based on the user's preferences. For instance, users can substitute Android's own camera application with another third-party camera application developed for Android phones. In other words, Android gives users the opportunity to fully tailor their mobile phones to suit their interests. [9]

Android makes it easy to build new and innovative applications that can easily connect or communicate with other applications to achieve richer and more relevant user experience. It enables developers to build applications that dismantle the barrier between applications. For instance, with Android, it is possible to develop an application that updates the user's contacts with data from the web, or an application that presents users with various services they can access based on their location. [9]

Android ships with a very rich tool-set, useful libraries, tutorials, and examples that can be used to develop a fully working application with little or no experience within a very short time. With the tool-set and and documentation that Android provides, it is easy to

learn Android application development for beginners without experience, and even much easier for application developers coming from other platforms. [9]

Architecture

Android OS software stack is basically composed of four layers which include the Linux kernel, Libraries, Application Framework, and Applications. As we can see in figure 6, which describes the anatomy of the Android software stack, the lowest layer, the Linux kernel, is the closest to the hardware and consists of the various peripheral drivers and power management software, while the topmost layer, Android applications, is the closest to the user and consists of both native and third party Android applications with which the user interacts with the Android device. [11]

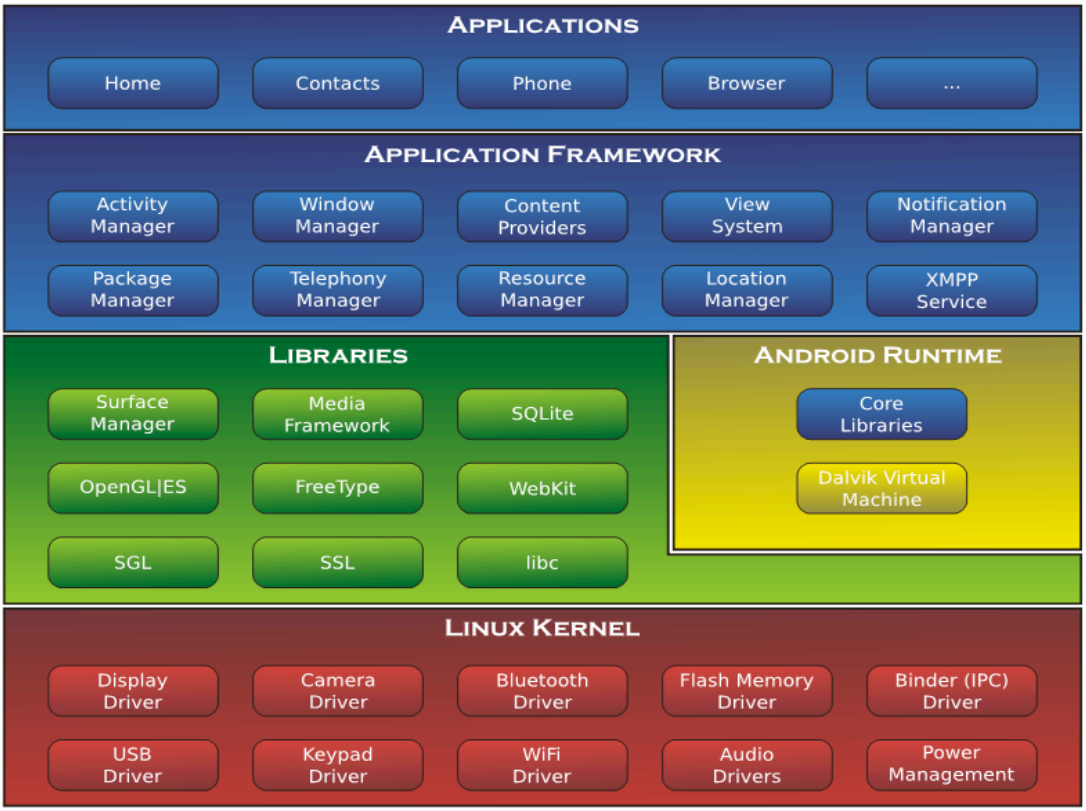


Figure 6: Android Software Stack. Copied from Smieh [10].

Linux Kernel

Linux kernel lies at the bottom of the stack providing a level of abstraction from the hardware. As presented in the figure above, it contains all the essential hardware drivers like USB driver, Audio drivers, and Bluetooth Driver. Among other functions, the

kernel is responsible for preemptive multitasking and low-level core system services such as memory, process and power management. It also provides a network stack. [11]

Libraries

Directly above the Linux kernel is a set of C/C++ libraries also referred to as native libraries. These fulfill a wide and diverse range of functions such as 2D and 3D graphics drawing, Secure Sockets Layer (SSL) communication, and SQLite database management, typically assessed through Java-based Android core library Application Programming Interfaces (API), or the Android Native Development Kit (NDK). [12]

Also at this layer is the Android Runtime Environment consisting of the Core Libraries and Dalvik Virtual Machine. The Core Libraries enable developers to write Android applications using the standard Java programming language. The Dalvik Virtual Machine is a kind of Java Virtual Machine specially designed and optimized for Android, which provides an environment for Android applications to run, enabling each application to run in its own process and with its own virtual machine instance. [11]

Application Framework

The Application framework sits at the third layer of the stack just above the Libraries. It provides various higher-level services to applications in the form of Java classes [10] implementing the concept that Android applications are built from reusable, interchangeable and replaceable components. Some of the Application Framework services include: Activity manager, which controls application lifecycle and activity stack, Content Providers, which enables applications to publish and share data with other applications, Package Manager, which enables applications to access information about other applications installed on a device, etc. [12]

Applications

Applications are located at the topmost layer of the stack. They include both the native Android applications that ship with a particular Android implementation and all other third-party applications that were either developed by the device manufacturer or installed by the user.

3 Materials

The main materials used in this project include nRF51822 system-on-chip (SoC) from Nordic Semiconductors, which served as the controller and BLE communication unit, MAX30100 from Maxim Integrated, which served as the sensor unit, and an LG Nexus 5 smart phone powered by Android Lollipop 5.1, which runs the mobile application client to the sensor. These will be briefly discussed here along the line of what they are, what they offer and their suitability for the project.

3.1 nRF51822

nRF51822 belongs to the nRF51 Series from Nordic Semiconductors, which is a family of multi-protocol Soc devices for wireless applications distinguished by their high flexibility and ultra-low power consumption. It is designed ideally for BLE and 2.4GHz ultra low-power wireless applications and supports both BLE and a proprietary Nordic 2.4 GHz protocol stack, Gazell. It also has a built-in set of analog and digital peripherals that can work together through what is known as the Programmable Peripheral Interconnect (PPI) system without the intervention of CPU. Its ultra-compact form factor makes it possible for it to be used in really small smart sensor applications. It is also available in a bigger 6X6 mm 48-pin QFN packages. Nordic Semiconductor provides its BLE protocol stack as a pre-compiled, pre-linked binary files called SoftDevices. [13] Figure 7 below is a schematic of nRF51822 chip showing both the general purpose input-output pins and special purpose pins with their numbers and names.

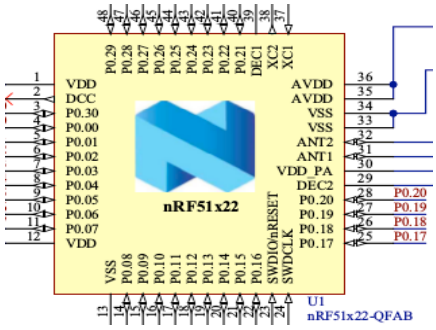


Figure 7 Schematic of nRF51x22 chip showing its pinouts. Copied from Nordic Semiconductor [14, 79].

In this project, an evaluation board for nRF51822, nRF51822-EK, was used for easy access to the chips periherals and for easy and fast application development. As shown in figure 8 below, the board comes with a programmer for the chip's programming, an antenna, some push buttons, current measurement pins, a mini usb for power supply and programming, some LEDs and other components.

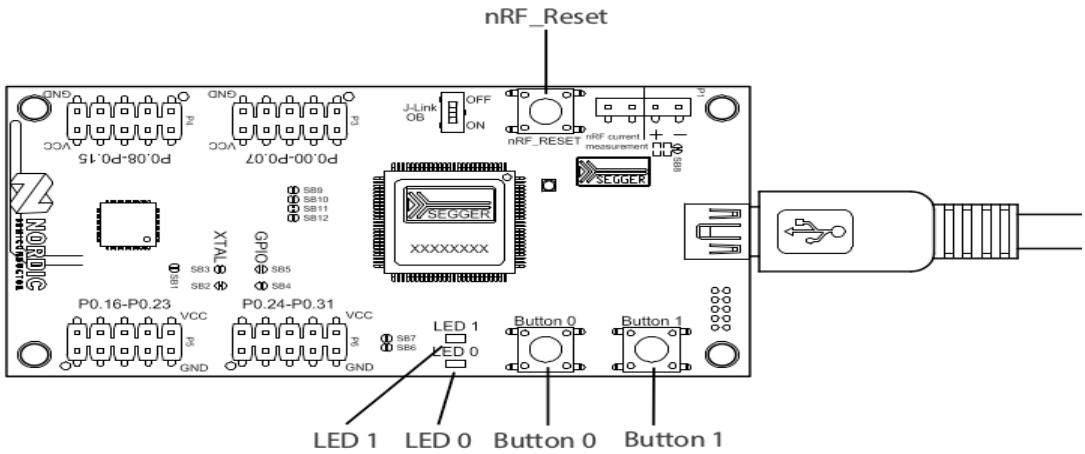


Figure 8: Schematic of nRF51822 Evaluation Kit. Copied from Nordic Semiconductor [15,7].

Main Features

nRF51822 is a highly flexible 2.4 GHz multi-protocol complete system on chip device. It comes with a 32-bit ARM Cortex M0 CPU core and either 256kB flash + 32kB RAM or 128kB flash + 16kB RAM. It is thread safe and run-time protected, and is equipped with a well-designed event-driven API. [15]

For data transmission, nRF51822 provides three different data rates 2 Mbps, 1 Mbps, and 250 kbps for different application purposes. It has an output power of up to +4 dBm in all modes and up to -93 dBm RX sensitivity while in BLE mode. With an advanced power management scheme, which enables each system block to independently control its own clock and to be powered on or off independently, and a Programmable Peripheral Interconnect (PPI) system, which enables autonomous interactions among peripherals independent of the procesor, it greatly minimizes power consumption. [15]

Suitablility for Project

In the first place, nRF51822 is suitable for this project because it is a BLE chip, and one of the requirements for the project is a BLE chip for ultra-low power wireless

communication. Any BLE chip, therefore, would be suitable for this project given this first reason. However, in addition to the features discussed earlier, this chip's suitability for the project was due to its availability in a very compact form factor that suited the purposes of the project, the rich application development support and tools provided by the manufacturer, and a relatively large and very active development community formed around it, which helps to make application development easier and faster.

3.2 MAX30100

MAX30100 is a pulse oximeter and heart-rate sensor integrated circuit (IC) for wearable health monitoring systems or devices. It detects pulse oximetry and heart rate signals with a combination of two LEDs (red and infra red), a photodetector, optimized optics, and low-noise analog signal processing techniques. It can operate from either 1.8V or 3.3V power supplies and can be powered down programatically by software with negligible standby current, thereby presenting the possibility of leaving the power supply connected all the time. It is typically used in fitness assistant devices, medical monitoring devices and wearable devices. [16, 1.] Figure 9 below shows the pin configuration of MAX30100 and the top and bottom views of the chip. The LEDs (red and infra red) are located on top while the pins are located at the bottom.

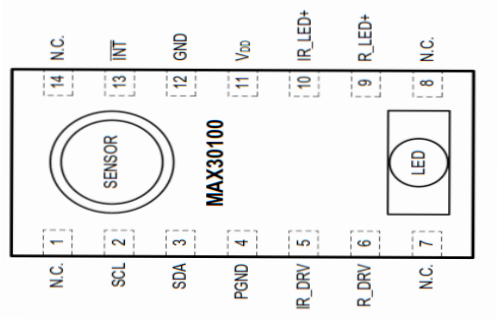


Figure 9: MAX30100. On the left is the Pin Configuration Diagram. Copied from MAX 30100 data sheet [16, 8] and on the right is the top and bottom of MAX 30100. Copied from Mouser Electronics [19].

Main Features [16, 1]

MAX30100 is a complete pulse oximeter and heart-rate sensor solution on a single die. With its LEDs (red and infra red), photo sensor and high-performance Analog Front-

End all integrated in a single chip, it is very simple and convenient to use. Its form factor, 5.6mm x 2.8mm x 1.2mm 14-Pin Optically Enhanced System-in-Package , makes it suitable for wearable devices. [16, 1]. For efficient power management and savings, it has programmable sample rate and LED current , and ultra-low shutdown current (typically 0.7μA). Other important features include high Signal to Noise Ratio (SNR), integrated ambient light cancellation, and high sample rate capability.

Suitability for Project

At the beginning of this project, MAX30100 was about the only heart-rate monitor sensor solution I could access that offered an integrated LED, photodetector and signal filtering and processing units, all in a single die. This made it preferable among other similar sensors available. Furthermore, it comes in a relatively small form packaging and promises a high signal to noise ratio(SNR) which is very important for clear and more reliable signal. All this, together with all the other features listed above, formed the basis for its suitability for the project.

3.3 Nexus 5

Nexus 5 is an Android-powered smartphone manufactured by LG Electronics for Google. It is co-developed with and marketed by Google Inc. as part of its Nexus line of flagship devices. Nexus 5 and its other siblings in the Nexus family are considered the original Android phones and they get their relevant Android updates before every other Android powered smart phones. It was the launch device for Android 4.4 “KitKat”, which offered a refreshed interface, performance improvements, increased Google Now integration, better battery life and other features. [17]

Main Features

Nexus 5 is relatively high-end Android smart phone with 4.95 inches touchscreen display of 1080 x 1920 pixels resolution (445 pixels per inch), a 2.26GHz quad-core Qualcomm Snapdragon 800 chipset, 2GB RAM, 16GB internal storage, and powered by Android Lollipop 5.1. Other features include 8/1.3 megapixel rear/front cameras and a number of sensors: Compass/Magnetometer, Proximity sensor, Accelerometer, Ambient light sensor, Gyroscope, Barometer. For connectivity, it is equipped with Wi-Fi, GPS, Bluetooth 4.0, NFC, Micor-Sim, 3G, 4G/LTE, GSM/CDMA.[17]

Suitabilitity for Project

This mobile device is suitable for the project simply because its version of Android, 5.1, supports BLE in the central role and provides APIs that applications can use to discover devices, query for services, and read/write characteristics. Since built-in platform support for BLE in the central role was introduced in Android version 4.3 Jelly Bean, it means that any Android mobile device powered by Android version 4.3 or higher would equally be suitable for this project.

4 Methods

4.1 Sensor Unit

The development of the sensor unit was done in three major steps. The first step was the preparation of the hardware setup, which included designing and building a break-out board for max30100 and connecting the appropriate pins from nRF51 evaluation board to max30100 board with jumper wires. The second step was to setup the development environment for firmware development. The step was the actual implementation of the firmware.

Preparing the Hardware setup for the Sensor

The heart-rate sensor IC used in this project, max30100, comes in a very small form (5.6mm x 2.8mm x 1.2mm) and therefore needs at least a break-out board to make it usable for development process. It also requires some pull-up resistors and capacitors to work properly. So the first task was to make a break-out board based on the specifications provided in the chip's data sheet as shown in figure 11 below.

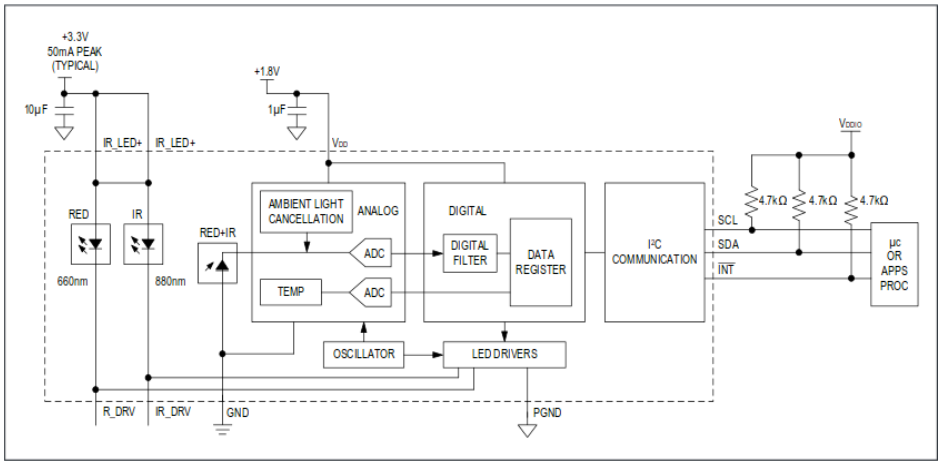


Figure 11: MAX30100 Schematic of a typical application circuit [16, 26].

Figure 11 shows the schematic of a typical application circuit for MAX 30100. The break-out board was made based on this schematic with the resistors and capacitors connected to the specified pins. Figure 12 below shows the break-out board with the sensor IC attached to it, which would be referred to as sensor IC board henceforth.

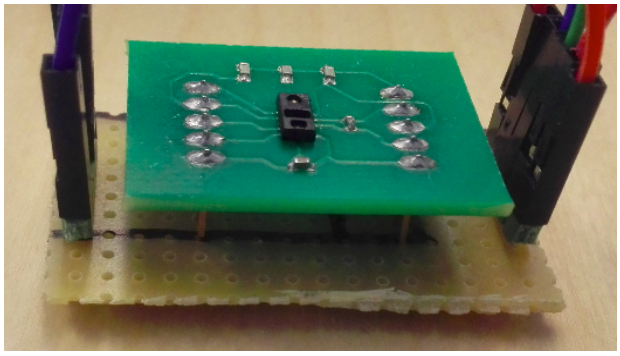


Figure 12: Break-out board with sensor IC (MAX30100) attached.

In the break-out board, the pins are attached in a way that they could be easily reachable without hindering easy access to the sensor. In this project, the sensor was intended for finger-based heart-rate measurement, so there is enough room for the finger to be placed on top of the sensor.

The next step was to connect the sensor IC to the controller and to voltage supply. The controller, nRF51 evaluation board, unlike the sensor IC, has everything needed for it to operate installed on the board. The connection was done according to figure 13.

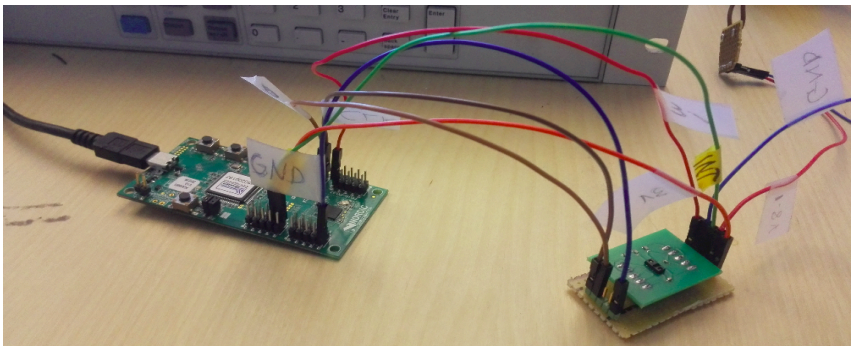


Figure 13: Connection setup for sensor IC and controller

As shown in figure 13, the nRF51 evaluation board, which serves as the controller board, is powered through a mini USB connector. This connector also serves as the programming point for the BLE chip nRF51822. In the course of this project, the sensor is powered from a PC from which it is also programmed for the sake of convenience. The sensor IC board gets its power both from the controller board and an external voltage supply. For easy description of the connected pins and the connections in the sensor connection setup, the schematic of sensor IC board and the pin headers of the controller board are shown figures 14 and 15 below.

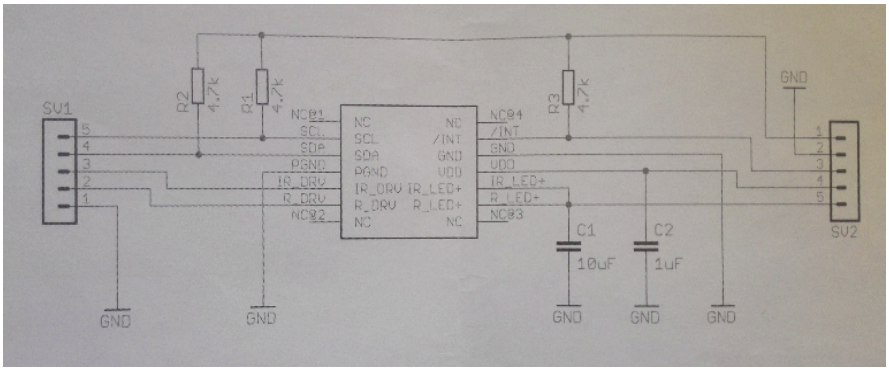


Figure 14: Schematic of the sensor IC board

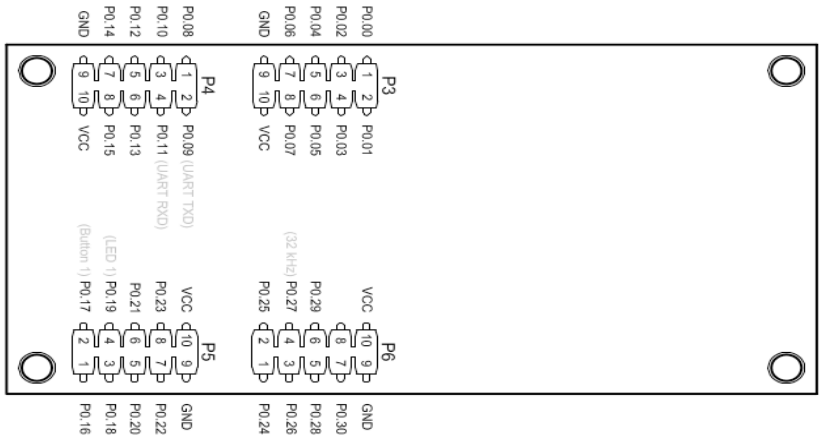


Figure 15: Pin headers of the controller board. Copied from Nordic Semiconductor [15,18].

In the sensor connection setup, pin SV2.5(red and infra red LEDs power supply pin) of the sensor IC board, is connected to pin P3.10 (vcc) of the controller board for 3V voltage supply to the LEDs; pin SV1.1 (Power Ground of the LEDs) of the sensor IC board is connected to pin P4.9 (Ground) of the controller board; pin SV2.1 of the sensor IC board is connected to pin P5.10 of the controller board for 3V voltage supply to the serial clock line (SCL), the serial data line (SDA) and the interrupt (INT) pins via 4.7k pull-up resistors; pins SV2.4 (Analog Power Supply input) and SV2.2 (Analog Ground) are connected to an external voltage supply for 1.8V input voltage.

Communication between the sensor IC (slave) and the controller (master) is via a 2-wire (I2C) which requires a serial clock line that drives the communication, and the serial data line that carries the data to be transfered. So pin SV1.5 (SCL) of the sensor IC board is connected to pin P6.1 (SCL) of the controller board for serial clock

communication and pin SV1.4 (SDA) of the sensor IC board is connected to pin P6.2 (SDA) of the controller board for serial data transfer. Pin SV2.3 (INT) of the sensor IC board is connected to pin P3.8 (INT) of the controller board for active low interrupt operation.

Firmware Application Development Environment Setup

The first step in the firmware development was to set up the development environment. A Linux machine running Ubuntu version 14.04 was used for the application development and the source code was written with vim text editor, which was already installed on the machine. The tools required to develop applications for nRF51822 were downloaded and installed: gcc cross compiler for ARM processors (gcc-arm-none-eabi), required to compile the source code written for nRF51822 on a Linux machine; Jlink GDBServer, required to program the chip; nRF51822 SDK with compatible softdevice, which provided the API for nRF51822 application development. With these tools downloaded, installed and configured, the environment was fully set up for the firmware application development.

Firmware Implementation

The basic logic of the firmware is found in the main routine :

```
int main(void){
    twi_master_init(); init_sensor();
    ble_stack_init(); timers_init();
    gpio_init(); gap_params_init();
    services_init(); advertising_init();
    conn_params_init(); sec_params_init();
    advertising_start();
    while(true){
        power_manage();
    }
}
```

Listing 1. Main function of the firmware.

As listing 1 shows, the function starts with initializing the various relevant software and hardware components both in the sensor IC board and the controller board and then starts the BLE advertisement, after which it enters a forever loop in which the function power_manage() is called, which basically waits for an event to occur. The events could be any of the BLE events registered to be handled, an event that announces that data is ready to be read from the sensor IC, depicted by the INT pin being pulled low, or a power down or system error event which causes a shutdown or deep sleep, as the case may be. The first function in the main routine initializes the controller board (master) for I2C transactions.

```
bool twi_master_init(void) {
    TWI_SDA_STANDARD0_NODRIVE1();
    TWI_SCL_STANDARD0_NODRIVE1();
    TWI_SCL_HIGH(); TWI_SCL_OUTPUT();
    TWI_SDA_HIGH(); TWI_SDA_OUTPUT();
    return twi_master_clear_bus();
}
```

Listing 2. Initialize I2C master. Copied from Nordic Semiconductor nRF51 SDK [21].

Listing 2 basically sets the SCL and SDA pins on the controller board to high, configures them as output and clears any data in the bus. The SCL and SDA are considered active when low and inactive when high, hence they are set to high at initialization. The second initialization function init_sensor(), initializes the sensor IC board with initial configuration parameters.

```
static void init_sensor(){
    config.spo2_config = (SPO2_HI_RES_EN | SPO2_SR_100);
    config.mode_config = HR_ONLY_EN;
    config.led_config =  (RED_PA_50_0 | IR_PA_50_0);
    initial_config(config);
}
```

Listing 3. Initialize sensor function

The sensor IC chip could be configured to operate either in heart-rate only mode or in Blood Oxygen Saturation (SPO2) mode. The LED pulse width, LED current level, ADC resolution and Sample Rate of the sensor IC could also be configured. [16,15-17] In

listing 3 above, the mode is set to heart-rate only mode, the LED pulse width is set to 1.6ms, the current levels of the infra red and red LEDs are set to 50.0mA, and the ADC resolution is set to 100 samples per second. These settings could also be modified according to application needs. However, in this project, the initial settings remained unchanged. The next function, ble_stack_init(), initializes the BLE stack, which is called softdevice in Nordic Semiconductor's terms.

```
static void ble_stack_init(void){
    uint32_t err_code;
    SOFTDEVICE_HANDLER_INIT(NRF_CLOCK_LFCLKSRC_XTAL_20_PPM,
          false);
    err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
    APP_ERROR_CHECK(err_code);
    err_code = softdevice_sys_evt_handler_set(sys_evt_dispatch);
    APP_ERROR_CHECK(err_code);
}
```

Listing 4. BLE stack initialization function. Copied from Nordic Semiconductor nRF51 SDK [21].

Listing 4 uses the softdevice API to set event handlers for BLE events and system events, passing them over to the stack so that it knows how to handle such events.
The next function, timers_init(), initializes the timers needed for the application:

```
static void timers_init(void){
    APP_TIMER_INIT(APP_TIMER_PRESCALER,     APP_TIMER_MAX_TIMERS,
                   APP_TIMER_OP_QUEUE_SIZE, false);
}
```

Listing 5. Initialize timers function. Copied from Nordic Semiconductor nRF51 SDK [21].

Listing 5 calls SDK wrapper function for a softdevice API to initialize the only timer used in this application, setting the value of the RTC prescaler register (first parameter) to 0, the maximum number of simultaneously created timers (second parameter) to 3 and the size of timer operation queues (third parameter) to 4.

The next function, gpio_init(), initializes the gpio pins:

```
static void gpio_init(void){
    simple_uart_config(RTS_PIN_NUMBER, TX_PIN_NUMBER,
          CTS_PIN_NUMBER,     RX_PIN_NUMBER, HWFC);
    NRF_GPIO->PIN_CNF[INTERRUPT_PIN] =
        (GPIO_PIN_CNF_SENSE_Low << GPIO_PIN_CNF_SENSE_Pos)
        |(GPIO_PIN_CNF_DRIVE_S0S1 << GPIO_PIN_CNF_DRIVE_Pos)
        |(NRF_GPIO_PIN_NOPULL << GPIO_PIN_CNF_PULL_Pos)
        |(GPIO_PIN_CNF_INPUT_Connect << GPIO_PIN_CNF_INPUT_Pos)
        |(GPIO_PIN_CNF_DIR_Input << GPIO_PIN_CNF_DIR_Pos);
    NVIC_EnableIRQ(GPIOTE_IRQn);
    NRF_GPIOTE->INTENSET = GPIOTE_INTENSET_PORT_Set <<
          GPIOTE_INTENSET_PORT_Pos;
    sd_nvic_SetPriority(GPIOTE_IRQn, NRF_APP_PRIORITY_LOW);
}
```

Listing 6. Function for Initializing GPIO pins.

Listing 6 uses the SDK library functions and the softdevice API to configure the UART pins for debugging purposes and one GPIO pin, Pin 7, as the INT pin for GPIO interrupt. This pin is connected to the INT pin of the sensor IC board so that it triggers an interrupt when it is pulled low, indicating data ready to be read from the sensor IC board. When this interrupt happens, the interrupt handler is called to handle the interrupt:

```
void GPIOTE_IRQHandler(void){
    if ((NRF_GPIOTE->EVENTS_PORT != 0)){
        NRF_GPIOTE->EVENTS_PORT = 0;
    }
    int_status = get_interrupt_status();
    if((int_status & HR_RDY) == HR_RDY){
       if(read_max30100_data(DEVICE_ADDRESS, FIFO_DATA_REG,
                             data, SIZE_OF_SAMPLE)){
          format_max30100_sample();send_string(sample);
        }
    }
}
```

Listing 7. GPIOTE interrupt handler.

Listing 7 clears the event that triggered the gpiote interrupt and gets the interrupt status. If the status indicates that heart-rate data is ready, the function reads the heart-rate data from the sensor IC, formats and forwards them to a connected client device. The next function, gap_params_init(), initializes the BLE gap parameters:

```
static void gap_params_init(void){
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
    sd_ble_gap_device_name_set(&sec_mode,
                (const uint8_t *)DEVICE_NAME,
                strlen(DEVICE_NAME));
    sd_ble_gap_appearance_set(
        BLE_APPEARANCE_GENERIC_HEART_RATE_SENSOR);
    memset(&gap_conn_params, 0, sizeof(gap_conn_params));
    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
    sd_ble_gap_ppcp_set(&gap_conn_params);
}
```

Listing 8. Function to initialize GAP paremeters.

Listing 8 uses the softdevice API and the SDK library functions to set the device name (a string defined by the programmer to identify the device), the device's category (selected from a list defined in the SDK), connection parameters (min connection interval, max connection interval, slave latency and connection supervision timeout). The connection interval is the time between two connection events and must be between 7.5 milliseconds and 4 seconds. Data transfer between two connected BLE devices is done within a connection. The minimum and maximum intervals in this project were set to 7.5 milliseconds and 4 seconds respectively. Slave latency refers to the number of times the slave or peripheral device can neglect requests from the master or central device. This was set to 0 in this project. The connection supervision timeout refers to the timeout from the last data exchange till a link is considered lost. This was set to 4 seconds in this project. [20]

The next initialization function is services_init(). This function initializes the BLE services that the sensor provides:

```
static void services_init(void){
    memset(&nus_init, 0, sizeof(nus_init));
    nus_init.data_handler = nus_data_handler;
    err_code = ble_nus_init(&m_nus, &nus_init);
}
```

Listing 9. Function to initialization services.

Listing 9 initializes the only service utilized in the firmware, the Nordic UART service (NUS), which is a custom BLE service developed by Nordic Semiconductor to emulate serial UART over BLE. It basically defines the needed variables, allocates some memory to the initialization data, sets the event handler and then calls an SDK library function, which does the actual initialization. The NUS has two characteristics: TX for data transfer and RX for data reception. These two characteristics are the interfaces between the sensor and a connected client for data communication. In this project, the TX was set to send data as notification to the connected mobile client, which has registered for notification on TX characteristic. The RX was configured with read and write permission such that a connected mobile client can write some value to it, which is in turn read by the sensor. This value could be a command or some other information.

The next function is the advertising_init(), which initializes the advertisement parameters and sets advertisement data.

```
static void advertising_init(void){
    uint8_t  flags =
        BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
    ble_uuid_t adv_uuids[] =
        {
            {BLE_UUID_NUS_SERVICE, m_nus.uuid_type}
        };
    memset(&advdata, 0, sizeof(advdata));
    advdata.name_type               = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance      = true;
```

```
    advdata.flags.size              = sizeof(flags);
    advdata.flags.p_data            = &flags;
    memset(&scanrsp, 0, sizeof(scanrsp));
    scanrsp.uuids_complete.uuid_cnt =
        sizeof(adv_uuids) / sizeof(adv_uuids[0]);
    scanrsp.uuids_complete.p_uuids  = adv_uuids;
    err_code = ble_advdata_set(&advdata, &scanrsp);
    APP_ERROR_CHECK(err_code);
    memset(&m_adv_params, 0, sizeof(m_adv_params));
    m_adv_params.type = BLE_GAP_ADV_TYPE_ADV_IND;
    m_adv_params.p_peer_addr = NULL;
    m_adv_params.fp          = BLE_GAP_ADV_FP_ANY;
    m_adv_params.interval    = APP_ADV_INTERVAL;
    m_adv_params.timeout     = APP_ADV_TIMEOUT_IN_SECONDS;
}
```

Listing 10. Advertisement parameters initialization function.

Listing 10 sets advertisement data and initializes advertisement parameters. Here the sensor is set to include its full name in the advertisement data. The Universally Unique Identifiers (UUID) of the services and their characteristics are also included in the advertisement data. The sensor was set to advertise to all (undirected advertisement) every 25ms and to stop advertising (and go to sleep) after 180s of advertisement if no client has connected to it. It stops advertising once it connects to a client.

The next function, conn_params_init(), initializes the connection parameters:

```
static void conn_params_init(void){
    memset(&cp_init, 0, sizeof(cp_init));
    cp_init.p_conn_params = NULL;
    cp_init.first_conn_params_update_delay =
        FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  =
        NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   =
        MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    =
```

```
        BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail = false;
    cp_init.evt_handler = on_conn_params_evt;
    cp_init.error_handler = conn_params_error_handler;
}
```

Listing 11. Connection parameters initialization function. Copied from Nordic Semiconductor nRF51 SDK [21].

In listing 11, the function defines the variables required for the connection parameters initialization, allocates some memory to the initialization data, sets the initialization parameters and then calls an SDK library function that does the actual initialization. The connection parameters initialized here concern the connection events in which actual data transfer takes place between the sensor and a connected client. The next function, sec_params_init() initializes the sensor's security parameters:

```
static void sec_params_init(void){
    m_sec_params.timeout      = SEC_PARAM_TIMEOUT;
    m_sec_params.bond         = SEC_PARAM_BOND;
    m_sec_params.mitm         = SEC_PARAM_MITM;
    m_sec_params.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    m_sec_params.oob          = SEC_PARAM_OOB;
    m_sec_params.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    m_sec_params.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
}
```

Listing 12. Sensor security parameters initialization function. Copied from Nordic Semiconductor nRF51 SDK [21].

Listing 12 initializes the security parameters, which include timeout for pairing request or security request, bonding, man in the middle protection, input/output capabilities, out of band data, and minimum and maximum encription sizes. Here the sensor is configured with a relatively loose security policy. This needs to be further improved in the next versions of the firmware. The next function, which comes immediately before the main loop is advertising_start():

```
static void advertising_start(void){
```

```
    uint32_t err_code;
    err_code = sd_ble_gap_adv_start(&m_adv_params);
    APP_ERROR_CHECK(err_code);
}
```

Listing 13. Function for starting sensor advertisement. Copied from Nordic Semiconductor nRF51 SDK [21].

Listing 13 starts the advertisement, in which the sensor advertises itself as a connectable peripheral together with the services it offers. As this listing shows, a softdevice function, which starts the advertisement, is called with a pointer to the already set advertisement parameters as an arguement.

4.2 Android Mobile Client Application

The client application communicates with the sensor through BLE to read PPG samples at the rate of 100 samples per second. The samples recieved when a finger is placed on the sensor are considered valid samples while others are considered invalid samples or noise. The valid samples pass through two stages of filtering. The first stage, detrending, serves to remove the trend in the samples, and the second stage, smoothening, serves to reduce or eliminate high frequency noise from the signal. After filtering, the samples are visualized on a dynamic graph and the heart rate is calculated from the filtered samples. The application starts to record PPG samples and heart rate values once it starts recieving valid samples and stops recording after about 120 seconds or once the finger is removed from the sensor.

The mobile application implementation would be discussed along the following major functions: BLE communication with sensor, sample filtering, heart-rate calculation, visualization and recording.

BLE Communication with Sensor

BLE communication involves scanning, connection, registering for notifications, recieving notifications, disconnection. The application gets from the user the id of the remote sensor to scan for and performs a targeted scan according to the following code:

```
private void scanLeDevice(final boolean enable) {
  if (enable) {
    mHandler.postDelayed(new Runnable() {
      public void run() {
        mLEScanner.stopScan(mLeScanCallback);
          if(mTargetSensorAddress == null){
            setResult(DEVICE_NOT_FOUND);
              finish();
          }
      }
    }, SCAN_PERIOD);
    mLEScanner.startScan(filters, settings, mLeScanCallback);
  } else {
    mLEScanner.stopScan(mLeScanCallback);
  }
}
private ScanCallback mLeScanCallback = new ScanCallback() {
  public void onScanResult(int callbackType,
  final ScanResult result) {
    BluetoothDevice device = result.getDevice();
    if (device.getName().equalsIgnoreCase(mSensorId)){
      sensorAddr = device.getAddress();
      scanLeDevice(false);
      result.putExtra(EXTRA_ADDR, sensorAddr);
      setResult(Activity.RESULT_OK, result);
      finish();
    }
  }
};
```

Listing 14. Scanning for sensor.

The code snippet in listing 14 scans for remote sensors based on specified UUID. During the scanning, the application searches the scan result for a sensor with the id specified by the user. Once the sensor of interest is found, the scanning ends, otherwise, the scanning lasts for 10 seconds (SCAN_PERIOD). The application then gets the address of the remote sensor of interest and initiates a connection.

```
public boolean connect(final String address) {
  if (mBluetoothAdapter == null || address == null)return false;
  final BluetoothDevice device = mBluetoothAdapter
                                .getRemoteDevice(address);
  if (device == null)return false;
  mBluetoothGatt = device.connectGatt(this, false,
                                      mGattCallback);
  mBluetoothDeviceAddress = address;
  mConnectionState = STATE_CONNECTING;
  return true;
}
```

Listing 15. Connecting to remote sensor.

The code snippet in listing 15 initiates a connection with the specified remote sensor. It checks that the Bluetooth adapter of the mobile device is turned on and that the remote sensor address is available. It then gets a handle for the remote sensor and starts the connection transaction. After the connection is established, the application will search for the BLE services implemented by the remote sensor. The service of interest here is the UART service. When this is discovered, the application will then get the RX characteristic from the service and registers for notification on this characteristic as shown in the following code snippet.

```
public void enableRXNotification(){
    mBluetoothGatt.setCharacteristicNotification(
                              mRXCharacteristic, true);
    BluetoothGattDescriptor descriptor = mRXCharacteristic
              .getDescriptor(CCCD_UUID);
    descriptor.setValue(BluetoothGattDescriptor
              .ENABLE_NOTIFICATION_VALUE);
    mBluetoothGatt.writeDescriptor(descriptor);
}
```

Listing 16. Register for Notification

The code snippet in listing 16 registers for notification on RX characteristic, which carries PPG samples from the sensor, so that each time the value is updated, the

application will get a notification of the update and read the value from the characteristic.

```
public void onCharacteristicChanged(BluetoothGatt gatt,
            BluetoothGattCharacteristic characteristic) {
    if(characteristic.getUuid().equals(RX_CHAR_UUID)) {
        int value = Integer
        .parseInt(characteristic.getStringValue(0).trim());
        if(value > MIN_Y ){
            if(mSamples4HRCalc.size() == TEN_SECONDS_SAMPLES){
                calculateHR(new ArrayList<>(mSamples4HRCalc));
                mSamples4HRCalc.clear();
            }
            value = iIRFilter(detrend(value));
            if(value > 0) {
               mSamples4HRCalc.add(value);
               broadcastUpdate(ACTION_RX_DATA_AVAILABLE, value);
            }
        }
        else {
            broadcastUpdate(ACTION_RX_DATA_AVAILABLE, 0);
            mSamples.clear();
        }
    }
}
```

Listing 17. Reading PPG samples from sensor.

The code snippet in listing 17 is called by the BLE stack each time a new sample is read by the sensor. It reads the sample, passes it through the filtering stages and then sends the filtered sample onward for visualization and recording. The next BLE transaction to be discussed here is disconnection, which comes when the user wants to end BLE communication. The user initiates the disconnection by pressing the disconnection button on the application's user interface. Once this button is pressed, the application will initiate disconnection from the remote sensor in the following snippet.

```
public void disconnect() {
    if (mBluetoothAdapter == null || mBluetoothGatt == null) {
        return;
    }
    mBluetoothGatt.disconnect();
}
```

Listing 18. Disconnecting from remote sensor.

The code snippet in listing 18 initiates the disconnection event. It first checks if there is still a valid handle to a remote sensor, and if a valid handle still exists, it will initiate disconnection from the connected remote sensor.

Sample Filtering

The PPG samples recieved from the connected sensor are filtered to make the peaks in the signal as distinct as possible, and to further lower the signal-to-noise ratio. The first stage of the filtering process is a simple detrending function which returns the absolute value of a sample subtracted from the mean of the sample and the preceeding 99 samples.

```
private int detrend(int sample){
    int detrended = 0;
    mFilteredSamples.add(sample);
    if(mFilteredSamples.size() < 100)
        return detrended;
    for(int i = 0; i < 100; i++)
        detrended += mFilteredSamples.get(i);
    mFilteredSamples.remove(0);
    detrended = detrended/100;
    return Math.abs(sample - detrended);
}
```

Listing 19. Detrending function.

Listing 19 is a simple implementation of detrending. It basically removes the trend in the signal. As shown in the snippet, the application remembers the preceeding 99 samples in order to perform the detrending. Thus for the first 99 samples recieved, the

function returns 0. The second stage of filtering, shown in listing 20 below, is a simple low pass filter implemented as a 31-point moving average to remove the high frequency noise.

```
private int iIRFilter(int sample){
    int filtered = 0;
    mSamples.add(sample);
    if(mSamples.size() < 31)
        return filtered;
    for(int i = 0; i < 31; i++)
        filtered += mSamples.get(i);
    mSamples.remove(0);
    return filtered/31;
}
```

Listing 20. A simple low pass filter for noise reduction.

Heart Rate Calculation

In PPG-based heart rate measurment, heart rate is determined by the number of peaks detected in the PPG signal per minute. Thus each peak in the signal represents one heart beat. The application calculates the heart rate for every 1000 samples. Since the sampling rate of the sensor is set at 100 samples per second, the application calculates the heart rate every 10 seconds. The peak detection algorithm used in the application was defined and implemented in MATLAB by Eli Billauer.[22]

```
private void calculateHR(final ArrayList<Integer> samples){
   for(int i = 0; i < samples.size(); i++){
      for(int i = 0; i < samples.size(); i++) {
          double cur = samples.get(i);
             if (cur > mx) {
                 mx = cur; mxPos = i;
             }
             if (cur < mn) {
                 mn = cur; mnPos = i;
             }
             if(lookForMax == 1){
```

```
                    if(cur < (mx - delta )){
                        mMaxPoints.add(mx);
                        mMaxPos.add(mxPos);
                        mn = cur; mnPos = i; lookForMax = 0;
                    }
                }else{
                    if(cur > (mn + delta)){
                        mMinPoints.add(mn);
                        mMinPos.add(mnPos);
                        mx = cur; mxPos = i; lookForMax = 1;
                    }
                }
            }
            double peaks = mMaxPoints.size();
            double duration = samples.size()/SAMPLING_RATE;
            mHr = (int)Math.round((peaks*60)/duration);
        }
}
```

Listing 19. Heart rate calculation.

In listing 19, the snippet calculates the heart rate by detecting the number of peaks in a set of 1000 samples. It then determines how long it takes to get one peak in seconds (T). The heart rate value is given as 60 seconds (a minute) divided by T.

Visualization and Recording

The application displays the filtered samples on a graph for visualization. The y-axis of the graph represents the PPG samples while the x-axis represents time. The heart rate value is also displayed on the screen and is updated each time the value changes.

```
private void updateGraph(int value) {
    if(mCounter > X_RANGE){
        mCounter = 0;
    }
    if(mLineGraph.getItemCount() > mCounter){
        mLineGraph.removeValue(mCounter);
```

```
    }
    mLineGraph.addValue(mCounter, mCounter, value);
    mCounter++;
    mGraphView.repaint();
}
private void setHeartRateValue(int value) {
    if (value != 0) {
        hrView.setText(value + "BPM");
    } else {
        hrView.setText(" ");
    }
}
```

Listing 20. PPG samples and heart rate display functions.

The PPG samples and heart rate values are recorded as a text file named as the patient's id underscore date and time of recording(<patientId>_<time of recording>) and saved in the device's local storage.

```
ppgM.addToData(rxInt, 0); ppgM.addToData(hr, 1);
private void saveRecord(){
   File file;
   String start = new SimpleDateFormat("yyMMddHHmmss",
                  Locale.US).format(record.getStart());
   String fileName = record.getPatientId()+"_"+start+".txt";
   file = new File(dir, fileName);
   FileWriter fw = new FileWriter(file, true);
   fw.append(record.toJson());
   fw.flush();
   fw.close();
}
```

Listing 21. Record and save PPG samples and heart rate values.

The code snippet in listing 21 records PPG samples and hr values by adding them to a collection. This collection is then saved as a text file in the local storage of the mobile device for future reference. The file contains a maximum of 2 minutes of records.

5. Results

The outcome of this project is a laboratory-based prototype for heart rate sensor that works with an Android mobile device to measure and record heart rate from the finger. The following is a pictorial overview of how the device works.

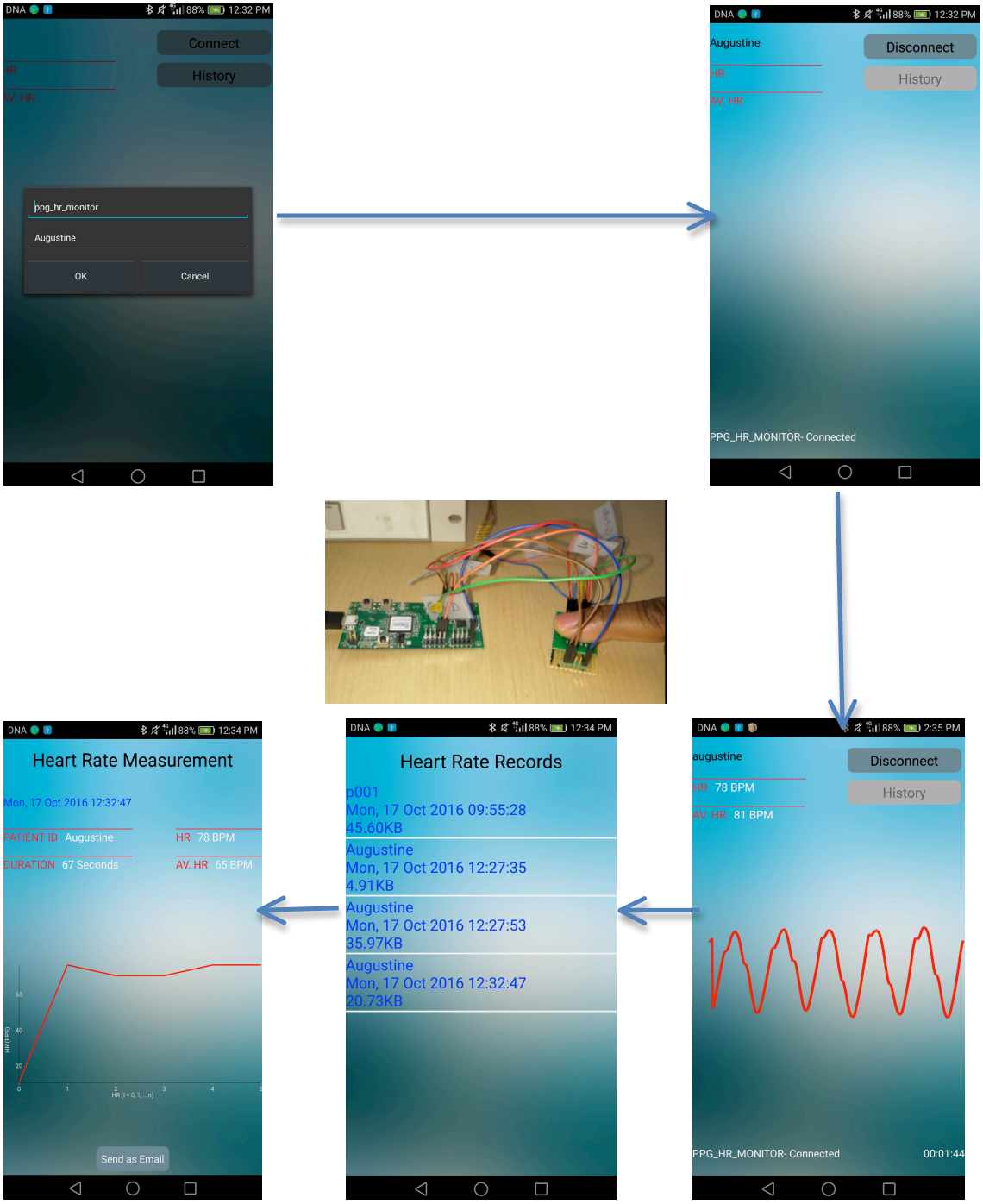
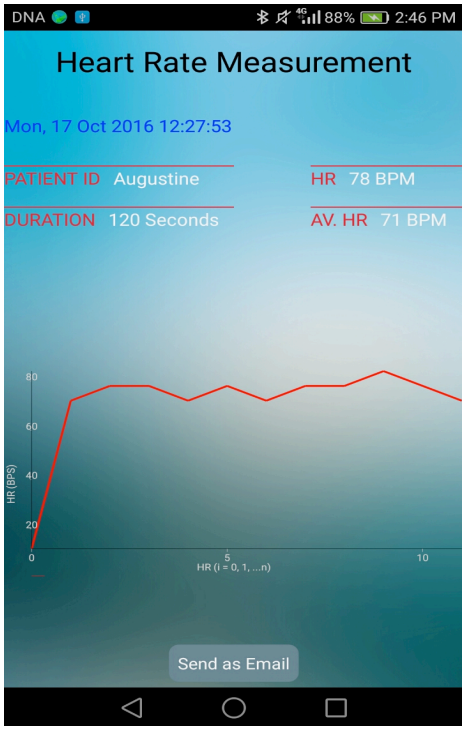
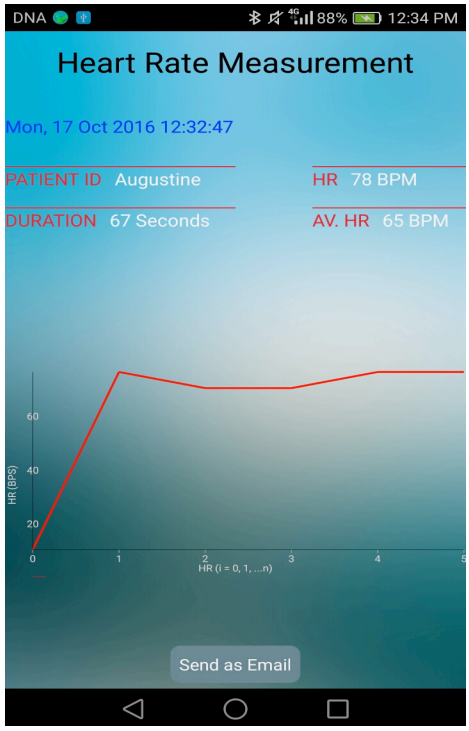


Figure 16. A pictorial overview of the heart rate monitor operation

Figure 16 describes the operation of the heart rate monitor developed in this project. When the sensor is turned on, the user opens the mobile application and then enters her user id (or some other form of identification) and the sensor id. The application remembers the most recent user and sensor ids, so if the user has just used it she would not have to enter these again. She then places her finger on the sensor, right on top of the oximeter chip so that the finger covers the LEDs and the photodiode. The application then detects the user's finger and starts to visualize the PPG samples captured from it. At the same time it calculates both the heart rate and average heart rate, and records both the PPG samples and the heart rate values. The recording lasts for about two minutes.

The recorded data could be retrieved and viewed. Here only the heart rate and average heart rate are shown together with a graph of all the heart rate values calculated during the two minutes of recording. The heart rate presented on this view is the value that has the highest frequency, in other words, the mode of the set of heart rate values calculated by the application during the two minutes of recording, which could also be seen visually on the graph. To test the functionality of this heart rate monitor, some measurements were taken and recorded. The results of the test are presented in the following screen shots.





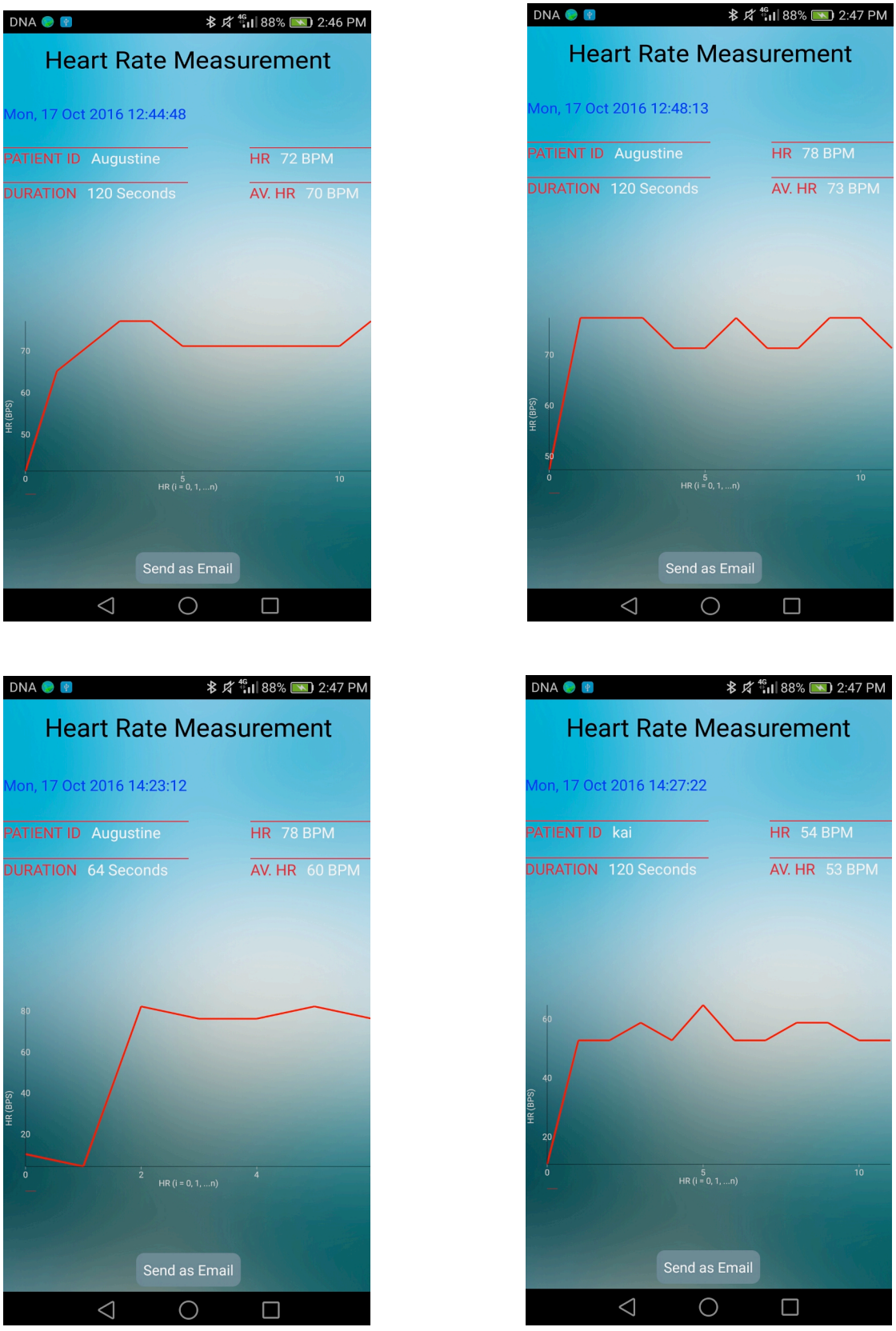


Figure 17. Screen shots of measurements taken from two volunteers

Figure 17 shows screen shots of the heart rate measurements taken from two persons, Augustine and Kai. The first five measurements were taken from Augustine and the measured heart rates were mostly 78 beats per minute (BPM). The last measurement in the figure was taken from Kai and the heart rate recorded was 54 BPM. The recorded heart rates together with the PPG samples could be sent as an attachment to a specified email. This functionality of the application would be useful if the user intends to perform further analysis on the samples.

6. Evaluation

This heart rate monitor was used alongside a commercially available pulse oximeter, Nonin G02, to heart rate results discussed in the previous chapter. The table below shows heart rate results from the two devices taken on the same day, 17 October, 2016.

NAME	TIME	PROJECT RESULTS(BPM)	G02 RESULTS (BPM)
Augustine	12:27:53 PM	78	78
Augustine	12:44:48 PM	72	74
Augustine	12:48:13 PM	78	76
Augustine	12:51:35 PM	72	73
Kai	02:27:22 PM	54	55
Augustine	02:33:52 PM	84	83

Table 1. Heart rate measurements.

In table 1 above, it is clear that the measurements obtained with the heart rate monitor developed in this project are quite impressive when compared with those recorded from a standard pulse oximeter for personal use. This is largely due to the high level signal-to-noise ratio of the oximeter sensor, MAX30100, the quality of the fitering process and the peak detection algorithm.

Even though the results look very impressive, there are still much room for improvement which could be achieved by using an oximeter sensor with a much higher signal-to-noise ratio, and much better filtering and peak detection algorithms. The sensor unit in this project could also be developed further into a wearable oximeter prototype. However, it would not bring any innovation since there are already wireless oximeter solutions in the market which measure both SP02 and heart rate.

The most challenging task for me in this project was removing the trends in the signal revieved from the sensor, and implementing the peak detection functionality, which is the key to calculating the heart rate. After extensive research, I was able to find some help on how to implement both the detrending and peak detection functionalities.

7. Conclusion

My goal in this project was to develop a wireless heart-rate monitor with MAX30100 for sensing PPG signal, nRF5822 for sensor-side control and BLE communication, and the Android system as the mobile platform for the sensor's client application for remote control, monitoring, data recording and display. The emphasis in this goal was mostly on the software rather than on the hardware. Therefore, the project excluded a complete prototype that could be tested beyond the development environment.

The result of this project is a working wireless heart-rate monitor, which includes a finger-based heart-rate sensor and a client application for the Android mobile platform. The sensor continously captures PPG data from a subject's finger and sends them via BLE to a connected Android mobile device running the client application, which then displays the data graphically, calculates and displays the periodic heart-rate, and records and saves the data for future reference.

The accuracy of the heart-rate measurements obtained from the monitor is quite impressive when compared to measurements from a standard finger oximeter given the circumstances and the level of this project. It could, however, be improved by using a higher quality sensor IC with a much better signal-to-noise ratio, and a better peak detection algorithm, since the PPG-based heart-rate is calculated from the number of peaks detected in a PPG signal of a given duration.

The source codes for both the firmware and the mobile application can be downloaded or cloned from these github repositories:
Firmware: https://github.com/augustio/heart_rate_sensor.git
Mobile application: https://github.com/augustio/HR_PPG_Monitor.git

References

1. Langereis Geert. Photoplethysmography (PPG) System Version 2 [online]. February 2010. https://www.cs.tau.ac.il/~nin/Courses/Workshop12a/PPG%20Sensor%20System.pdf. Accessed March 26 2016.

2. Omar Abdallah and Armin Bolz. Adaptive Filtering by Non-Invasive Vital Signals Monitoring and Diseases Diagnosis [online]. http://www.intechopen.com/books/adaptive-filtering-applications/adaptive-filtering-by-non-invasive-vital-signals-monitoring-and-diseases-diagnosis Accessed 26 March 2016.

3. Allen John, ”Photoplethysmography and its application in clinical physiological measurement” in Physiological Measurement, Vol. 28, Number 3 [serial online] 2007. http://iopscience.iop.org/article/10.1088/0967-3334/28/3/R01/meta. Accessed 26 March 2016.

4. Bluetooth SIG. Bluetooth low energy [online]. https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy. Accessed 26 March 2016.

5. Bluetooth Smart (Low Energy) Technology [online]. https://developer.bluetooth.org/TechnologyOverview/Pages/BLE.aspx Accessed 26 March 2016.

6. Bluetooth SIG. Specification of the Bluetooth System, Version 4.0, Vol. 1. June 30, 2010.

7. Open handset alliance. “Industry Leaders Announce Open Platform for Mobile Devices” November 5, 2007. http://www.openhandsetalliance.com/press_110507.html Accessed 26 March 2016.

8. Dunn Jeff, “There's no hope of anyone catching up to Android and iOS” in Business Insider, [online] August 22, 2016. http://www.businessinsider.com/smartphone-market-share-android-ios-windows blackberry-2016-8?r=US&IR=T&IR=T Accessed 16 October 2016.

9. open Handset Alliance. Android http://www.openhandsetalliance.com/android_overview.html Accessed 26 March 2016.

10. Smieh. Anatomy Physiology of an Android, CC BY-SA 3.0. https://commons.wikimedia.org/w/index.php?curid=20067152 Accessed 26 March 2016.

11. Android Architecture. http://www.tutorialspoint.com/android/android_architecture.htm Accessed 26 March 2016.

12. An Overview of the Android Architecture.
http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture
Accessed 26 March 2016.

13. Nordic Semiconductor. nRF51822
https://www.nordicsemi.com/eng/Products/Bluetooth-Smart-Bluetooth-low-energy/nRF51822
Accessed 26 March 2016.

14. Nordic Semiconductor. nRF51822 Product Specification V3.1.

15. Nordic Semiconductor. nRF51822 Evaluation Kit User Guide V1.2.

16. Maxim integrated. MAX30100 pulse Oximeter and Heart-Rate Sensor ICs Datasheet.
https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf
Accessed 26 March 2016.

17. LG Nexus 5.
http://www.lg.com/us/cell-phones/lg-D820-Sprint-Black-nexus-5
Accessed 26 March 2016.

18. Gadgets 360. LG Google Nexus 5
http://gadgets.ndtv.com/lg-google-nexus-5-1115
Accessed 26 March 2016.

19. Mouser Electronics. Maxim MAX30100 Pulse Oximeter & Heart-Rate Sensor ICs.
http://www.mouser.in/search/refine.aspx?Ntk=P_MarCom&Ntt=109862494
Accessed 26 March 2016.

20. Nordic Developer Zone.
https://devzone.nordicsemi.com/question/60/what-is-connection-parameters
Accessed 27 March 2016.

21. Nordic Semiconductor. nRF51 SDK Version 5.2.0.

21. Billauer Eli, “Peakdet: Peak detection using MATLAB” July 20 2012.
http://www.billauer.co.il/peakdet.html
Accessed 16 October 2016.

22. Townsend K, Cufi C, Akiba and Davidson R, Getting Started with Bluetooth Low Energy. Sebastopol, California: O’Reilly Media; 2014 [online edition].
https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/ch01.html
Accessed 5 November 2016.