1-1-2015

Development Of A Human Heart Rate And Skin Temeprature Monitoring System

Giribabu Sinnapolu
Wayne State University,

DEVELOPMENT OF A HUMAN HEART RATE AND SKIN TEMPERATURE MONITORING SYSTEM

by

GIRI BABU SINNAPOLU

THESIS

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

2015

MAJOR: ELECTRICAL ENGINEERING

Approved By:

Advisor Date

DEDICATION

Dedicated to

My Parents & My Brother

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my advisor Dr. Amar Basu for his advice, guidance and support. I would like to thank Dr. Robert Erlandson and Dr. Yong xu for being part of my thesis committee. I thank my parents, Sudhakar and Suvarna, my Brother, Jagadish for the support in all my endeavors. I thank Shiby for helping me out with Android Application, I thank Dr. Syed Mahmud for the advices throughout my MS study and Aziz makkiya our team manager at Ford motor company. I thank my friends Sumanth, Sai, Premkumar for all the support and encouragement during my difficult times.

PREFACE

In this thesis, I present my work Development of a human heart rate and skin temperature monitoring system.

Chapter 1 gives an introduction of HR, HRV, medical applications, problems and different kinds of heart rate devices available in market along with basic understanding of photoplythosmographic concept.

Chapter 2 explains about system design, which explains about the whole product design along with components integrated to the MCU.

Chapter 3 explains the test setup and method.

Chapter 4 demonstrates the experimental results carried out in this project.

Chapter 5 discusses the conclusion of the project.

Sincerely,

Giri babu sinnapolu

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Background/Motivation

The growth of population in the United States, including the elderly population, has led to significant challenges in US Healthcare. One such challenge is the increasing costs of managing chronic disease, including cardiovascular disease, diabetes, and mental health disorders [1]. Chronic disease is often linked to lifestyle factors, such as high blood pressure, obesity, poor diet, stressful environments, and alcohol consumption [2]. In the effort to reduce chronic disease, there is a great interest in encouraging patients to adopt a healthy lifestyle choices [1]. One philosophy is to provide patients with frequent, quantitative data about their health, and giving appropriate guidance based on such data [3].

Among the major chronic diseases, cardiovascular disease is the most common [4], affecting 611,105 Americans annually [5]. It is also the most expensive, costing the US healthcare system an estimated $315.4 billion in 2010 [6]. Quantitative methods for managing heart disease include heart rate monitoring, electrocardiography, nuclear stress testing, chest x-ray, and blood tests [7]. Most of the above tests require the patient to be at a medical clinic, which increases the cost of care. From a personal standpoint, patients often prefer home monitoring technologies and treatments for managing chronic disease [8]. Hospitals and healthcare organization are also interested in remote patient monitoring (RPM) technologies because they have been shown to significantly improve patient outcomes while reducing costs [1]. In order to encourage adoption of remote monitoring technologies, they must be accurate, simple to use, low cost, and provide the required data at regular intervals. Some notable examples of home health monitoring technologies used for cardiovascular disease include home blood pressure monitors, handheld ECG units, smartphone heart rate monitors, chest strap ECG monitors, and most recently, heart rate monitoring watches.

1.2 Wearable sensors

Among home health monitoring technologies, wearable sensors are useful when continuous physiological monitoring is required [9]. For example, continuous assessment of heart rate enables one to diagnose cardiac arrhythmias and assess heart rate variability (HRV), an indicator of cardiovascular and psychophysiological state [10]. In rehabilitation health care, wearable sensors allow caregivers to monitor physiological changes like fall detection etc., sense heartbeat to detect heart attack, wherein all these disorders can be monitored remotely [1] [2].

Other wearable sensors like skin temperature play an important role in sensing activity, energy expenditure, and potential medical emergencies. For example, an acute stroke could be assessed by increase in body temperature and fall detection, and skin temperature play an important role in diagnosis [11] [9]. When combined with wireless transmission, the implementation of these sensors will help in monitoring medical emergencies remotely [12]. There are several other emerging applications of wearable sensors. A few examples: clinicians use motion sensors to monitor neonatal lung development [13] by recording the number of breathing cycles; by golfers to monitor improper swings[14]; and by divers to detect harmful chemicals underwater which are a threat to human and aquatic life [15]. The most popular use is in the fitness market, where athletes and coaches track motion, energy expenditure, cardiovascular state, and other conditions using wearable sensors [10].

A decade earlier, wearable sensors were not feasible because technology could not meet the size and power constraints required for wearable devices. However, recent advances in microsensors, radio communication ICs, open software, low power microcontrollers, and smartphone technology has led to breakthrough advancements in developing wearable sensors [1]. This technology advancement has also helped researchers in various other ways to implement wearable sensors. Wearable sensors are a combination of software and hardware, these sensors are designed to help us detect various emerging medical conditions prior to serious symptoms. The sensor technology consists of important building blocks: 1) displays for visualization, 3) software and hardware required for sensing, 4) communication protocols like GSM, Zigbee, Bluetooth [16] for remote monitoring, 5) information technology for analyzing the data according to the requirement. After collecting and analyzing the raw data, the next important step is to communicate of the filtered data to the Android application.

As human life expectancy and chronic disease increases, healthcare costs have also increased [16]. Wearable sensors can potentially help mitigate issues such as cost, size, efficiency, accuracy, and power consumption [17] [18].

1.3 Heart Rate Monitoring Technologies

Heart rate is a universal physiological parameter, which responds to both physical and psychological load. The heart responds to increases in physical activity, and also to emotions such as fear, stress, and anxiety. Increases and decreases in heart rate are controlled by the central nervous system [18] [19], which includes the sympathetic and parasympathetic nervous system. The sympathetic nervous system controls the ‘flight or fight’ response, generally increases heart rate, while the parasympathetic nervous system brings the heart rate and blood pressure down during moments of relaxation [18] [19]. Continuous heart rate monitors (CHRM) can document these regular changes in heart rate, and therefore important play an important role in diagnosing physical and psychological load.

A popular device used for continuous heart rate monitoring is the Holter monitor (Figure 1.1), a portable electrocardiography (ECG) unit with electrodes attached to the chest area, and a recording device attached to the patient’s waist. The Holter monitor records continuous ECG waveforms, which can identify irregularities in heart function, including arrhythmias, palpitations, and others. Due to the size and bulk of the device, it is prescribed only on a short term basis, such as weekly or monthly, before they are returned to the doctor for data analysis. Holter monitors are prescribed to diagnose atrial fibrillation, multi focal atrial, palpitation, fainting etc.[21] [20].

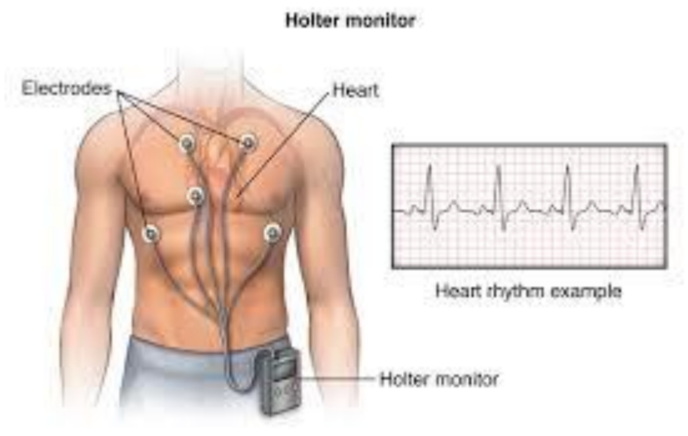


Figure 1.1: Holter monitoring [21]

1.4 Photoplethysmography (PPG)

Photoplethysmography (PPG) is an optical method of measuring heart rate, which can be used in wearable sensors [16]. When the heart contracts, it generates a pulsatile flow of blood in the circulatory system. Capillary beds experience and increase in blood volume, which can be detected by infrared optical sensors [20]. When PPG is measured in the transmission mode, it must be placed at extremities such as the fingertip or earlobe.

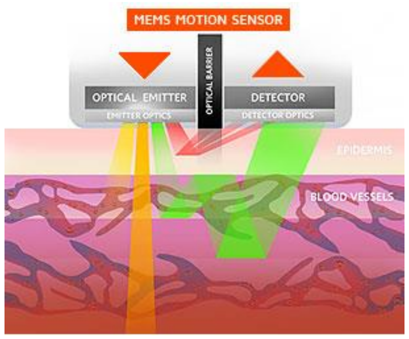


Figure 1.2: Concept of Reflectance Mode PPG

Reflectance mode PPG (Figure 1.2) is an alternative to transmission mode, which simplifies the sensor design, and enables measurement at more anatomical locations. A reflectance mode PPG sensor consists of an infrared (IR) light emitter and adjacent photodetector. When the sensor is placed close to a fingertip or an earlobe, it can detect local changes in blood flow. When a pulse of blood passes through the capillary bed just above the sensor, the increased capillary volume scatters the emitted light, reducing the intensity of the reflected light received at the photodetector. The periodic fluctuations in reflected light, typically about 1-5% of the total signal, correlates with heart rate [22]. The period of fluctuation, referred to as the r-r interval, is used to calculate heart rate. Transmission mode PPG, where the emitter and receiver are placed on opposite sides of a tissue, is a more common method, but are limited to extremities such as the fingertip or earlobe.

Signal processing is an important aspect of PPG, particularly in wearable sensors. PPG signals are often filtered to within a 0.5-5Hz passband using a combination of low pass, high pass, band pass, and moving average filters. The role of signal processing includes removing unwanted noise and signals resulting from electronic noise, ambient light, and motion artifacts. High pass filtering also removes the DC component of the optical signal, which is up to 100X larger than the

AC component. After filtering, the frequency of pulses are counted using a zero crossing threshold detector, and this is used to calculate the number of beats per minute (BPM). Signal processing will be discussed in chapter 3.

(Figure 1.3) shows a typical arterial pulsation waveform obtained from a PPG sensor. The waveform has a systolic phase, dicrotic phase, and diastolic phase. The ascending limb is the systolic phase and the descending limb is the diastolic phase, while the dicrotic notch indicates the closure of the aortic valve. The dicrotic notch changes according to the closure and opening of the valve [23]. The different parts of the waveform indicate the functioning of the right and left valve.

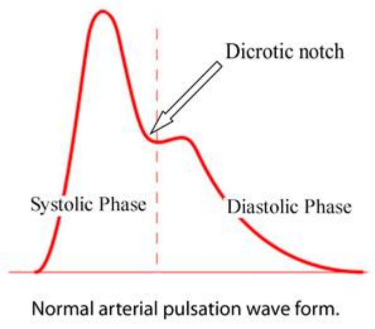


Figure 1.3: Normal arterial pulsation waveform [23]

1.5 Heart Rate Target Zones

In the consumer fitness market, one of the popular uses of heart rate monitoring is to help athletes maintain target heart rates during training. There are several predefined target zones, and a wearable heart rate monitor can help the athlete remain in one or more target zones to reach a specific training goal. The zones are defined as a percentage of maximum heart rate, which is age dependent. There are several algorithms available to calculate maximum heart rate (HRmax), but the simplest is **HRmax = 220 – age in years**.

	Heart Rate Target Zone 50-85%	Avg. Maximum Heart Rate100%
Age	**Beats/minute**	**Beats/minute**
20	100–170	200
25	98–166	195
30	95–162	190
35	93–157	185
40	90–153	180
45	88–149	175
50	85–145	170
55	83–140	165
60	80–136	160
65	78–132	155
70	75–128	150

Figure 1.4: Heart rate target zones

The fat burning or the fat loss zone, also called the recovery zone, accounts to a heart rate of 60% to 70%. The aerobic training zone, in the 70-80% of HRMax, helps an athlete to become stronger and fitter by developing the cardiovascular system endurance. The anaerobic zone, falling within 80-90% of HRMax, builds the lactic acid system which in turn helps burning the glycogen in lactic acid. The red line zone, between 90% to 100% of HRMax, is used for high intensity Interval training (HIIT) or other high level fitness training [24]. The recommended target zones are a general guideline, and individuals are recommended to consult with physician, and proceed with a stress test, to determine his or her exercise program.

1.6 Comparison of Wayne State Earlobe CHRM with existing devices

A medical-grade continuous heart rate monitor (CHRM) must meet several key criteria: 1) it must provide beat-to-beat accuracy even when the subject is in motion, 2) it must be small and comfortable to wear, 3) it must have low power consumption for all day operation on a rechargeable battery, and 4) it must be inexpensive to encourage wide adoption by healthcare organizations and the competitive consumer market.

Several commercially available heart rate monitors are shown in (Figure 1.6). Although these devices are adequate for consumer fitness, do not meet all these criteria need for medical monitoring. Most are based on Photoplethysmographic (PPG), which is prone to motion artifacts if used in fingertip and wristwatch devices. On the other hand, electrocardiography (ECG) monitors such as the Holter monitor, are tolerant to motion artifacts, but are typically in the form of chest straps, which are not comfortable for all-day use.

The Microfluidics and Bioinstrumentation lab is currently developing a CHRM which is both comfortable and free of motion artifacts, as shown in (Figure 1.5). It is based on a patented sensing technology which uses digital optical proximity sensors to detect heart rate [25] [26]. It results in a small (4×4 mm^2), low power (300 μW), and inexpensive (<$4 US) heart rate sensing technology. The miniature form factor enables placement on the earlobe where motion artifacts are minimized.

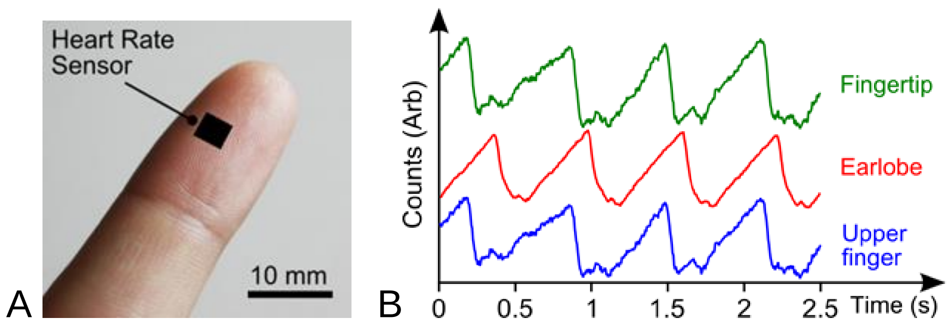


Figure 1.5: (A) Photograph of patented heart rate sensor showing the sensor size. (B) High quality heart rate waveforms obtained using this sensor in various anatomical locations [25] [26].

We are incorporating this sensor into a small earlobe, mounted system (1cmx1.5cm), which consists of the heart rate sensor, skin temperature sensor. Preliminary testing found that the signal received at the ear is more precise and accurate than anywhere else in the body due to fewer motion artifacts. By contrast, commercial products designed for the wrist or chest have substantial motion artifacts, which are difficult to remove even by filtering.

In addition to the on chip sensors, the earlobe sensor includes a Bluetooth Low Energy transmitter for sending data to an Android, iOS, or windows app, and an a built-in flash memory which stores 7-30 days of data when not connected. Designed for low power consumption, the device will enter sleep mode when not used. The two sensors for heart rate and skin temperature all consume low energy. It is expected the system will have full day battery life on a small rechargeable cell. Compared to existing devices, the improved accuracy, low power, and data caching capabilities make the current system well suited for continuous heart rate monitoring (CHRM).

Name	Basis Peak [27]	Garmin Forerunner [28]	Mio Fuse [29]	Fitbit Charge [30]	Garmin Vivosmart [31]	Jawbone UP24 [32]	Microsoft Band [33]	Jabra [34]	Tom Tom [35]	Bragi [36]	Wayne State OPS Earlobe Tracker [26]
Device Type	Wrist Band, Sports Watch	Sports Watch, GPS	Wrist Band, Sports Watch	Wrist Band	Wrist Band	Wrist Band	Wrist Band, GPS	Ear bud	Wrist Band, Sports Watch	Ear bud	Ear clip
Display Type	Monochrome LCD	Monochrome LCD	Dot-matrix LCD	OLED	OLED	LED indicator lights	OLED	LED Indicator	Monochrome LCD	LED Indicator	LED indicator lights
Compatibility	Android, iOS, Web	Windows, Mac	Android, iOS	Windows, Mac, Android, iOS	Windows, Mac, Android, iOS	Android, iOS	Android, iOS	Android, iOS	Windows, Mac, Android, iOS	Windows, Mac, Android, iOS	Android, iOS, Windows
Heart Rate Monitor	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Sleep Tracker	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No
Battery Life	About 4 days		About 7 days	About 10 days	About 7 days	5 to 7 days	About 2 to 4 days	10 days	10 days	10 days	10 days
Temperature sensor	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Flash Memory	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes, serial flash memory to store all sensor data
cost	$199	$139	$149	$129	$170	$99	$199	$199	$199	$299	<$50

Figure 1.6: Comparison of Wearable Heart Rate Monitors

1.7 Objective of this Thesis

The overall goal of this thesis is to develop the firmware for the CHRM sensor described above, which will run on the Texas Instruments CC2541, a low power microcontroller with an integrated bluetooth low energy radio. Specific goals include:

1. Acquire photoplethysmography (PPG) data from the Vishay VCNL4000 proximity sensor, using an I2C interface.
2. Acquire skin temperature from the Texas Instruments TMP112 sensor, using an I2C interface.
3. Acquire battery level data from the CC2541 analog to digital sensor.
4. Save PPG, and temperature data to a serial flash memory chip (Macronix MX25L6445E), using a serial peripheral interface (SPI)
5. Improve PPG data by optimizing the VCNL4000 sensor for heart rate monitoring.
6. Perform low power signal processing onboard the microcontroller to process PPG data, Perform median filtering to reduce noise, high pass filtering to remove the offset from the PPG signal, and calculate heart rate using a peak detector.
7. Send all data to an Android application via Bluetooth low energy.

All of the above tasks, and the individual sensors and ICs, must be optimized for low power consumption to improve battery life.

CHAPTER 2: SYSTEM DESIGN

2.1 System Block Diagram

The block diagram as shown in (Figure 2.1) represents various components which are used to build this application. The Texas Instruments CC2541, a low power microcontroller with integrated Bluetooth radio, is the master component of the system, while all other sensors such as skin temperature, and memory chip act as slave. The CC2541 collects data from the heart rate sensor, skin temperature sensor through I2C serial interface, saves the data to a local flash memory via a serial peripheral interface (SPI), and sends the data to an Android application via Bluetooth low energy. The CC2541 acts a peripheral as per the GAP roles, and sends sensor data to the GATT table. The GATT table will load the profiles with this data which will be sent over the air to the slave upon connection. When sending data, the CC2541 acts as a master, and the slave is the Android application or a PC based software (Btool).

The CC2540 has several low power modes which allows it to reduce power consumption when it completes taking measurements, and the Bluetooth low energy protocol also reduces power consumption during wireless transmission. Since wireless transmission consumes the most power, the system saves data locally on an on-board flash memory. By implementing these power saving measures, the system can provide long battery life.

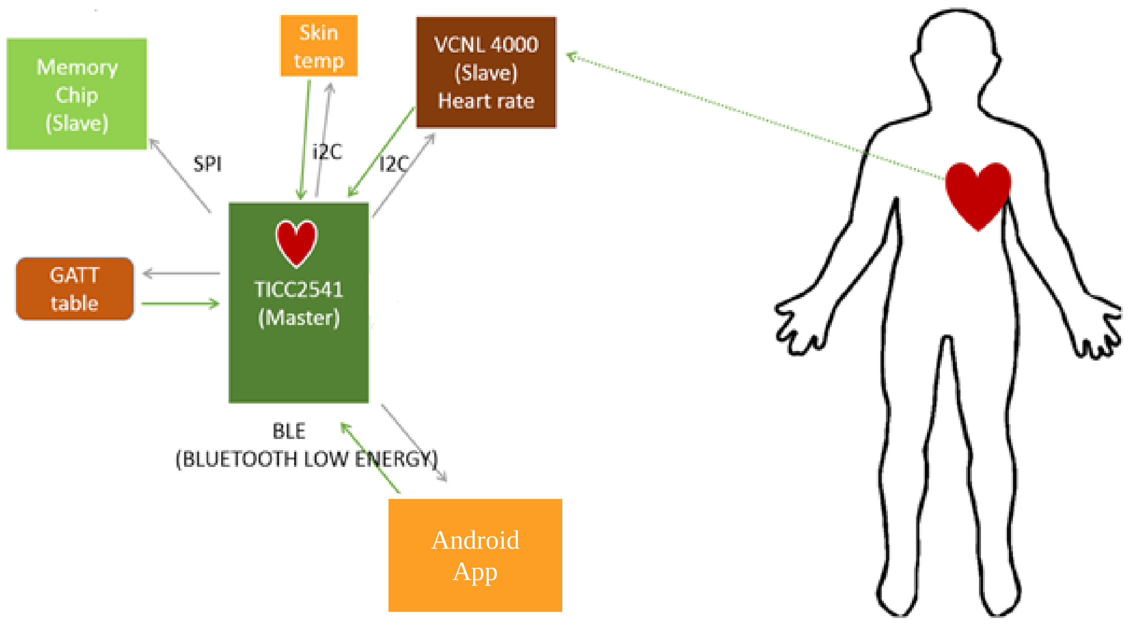


Figure 2.1: Block diagram of the sensor system

The following sections describe the individual components of the system.

2.2 Proximity Sensor for Heart Rate detection

The VCNL4000 is an optical proximity sensor with an integrated ambient light sensor.

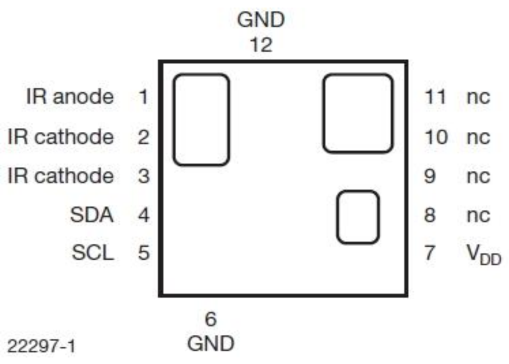
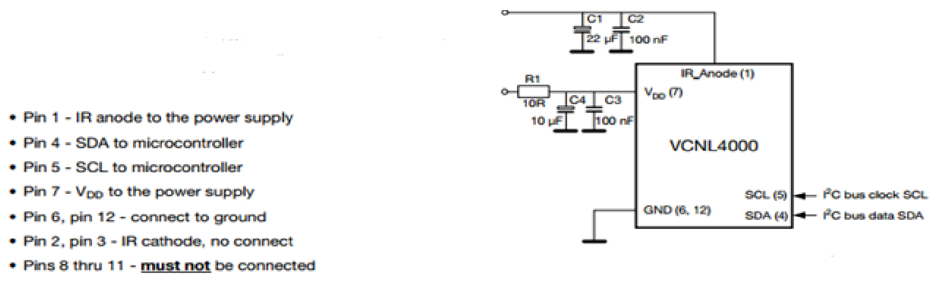


Figure 2.2: VCNL Pin Diagram [37]

As shown below, the pin configurations and connections are to be made to connect the microcontroller to the VCNL4000 board using the SmartRF05 (describe in the testing section).

2.2.1 Pin Connections



- Pin 1 - IR anode to the power supply
- Pin 4 - SDA to microcontroller
- Pin 5 - SCL to microcontroller
- Pin 7 - V_{DD} to the power supply
- Pin 6, pin 12 - connect to ground
- Pin 2, pin 3 - IR cathode, no connect
- Pins 8 thru 11 - **must not** be connected

Figure 2.3: VCNL4000 Application circuit and Pin connection [37]

The VCNL4000 board is connected to the SmartRF05 board via the debug connector. The connections are mentioned below:

	VCNL4000	SmartRF05
SDA	4	3(p18)
SCL	5	5(p18)
GND	6, 12	20(p18)
Vdd	7	3(p20)

Figure 2.4: Assigning VCNL4000 pin to SmartRF05 for connection

The connections are made for p18 and p20 accordingly. The (Figure 2.4) describes the connections.

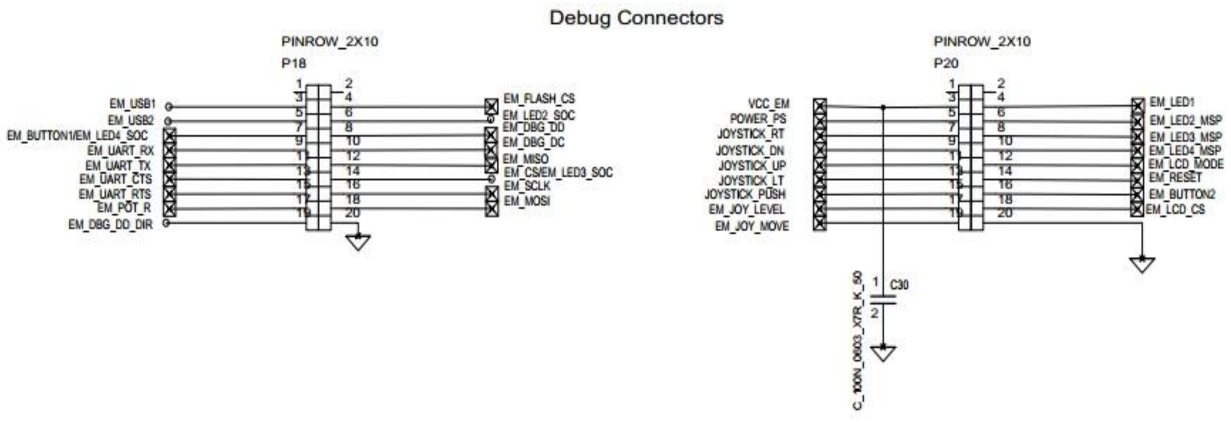


Figure 2.5: TI CC2541 Debug connectors

The VCNL4000 is used as a heart rate monitoring sensor. The VCNL4000 application document and datasheet gives a clear understanding of working procedure of the sensor. Make the hardware connections as shown in the (Figure 2.4 and 2.5). IAR workbench was used as a compiler to execute <simpleBLEperipheral> sample code [37]. The below figure explains the conceptual overview of designing an I2C communication.

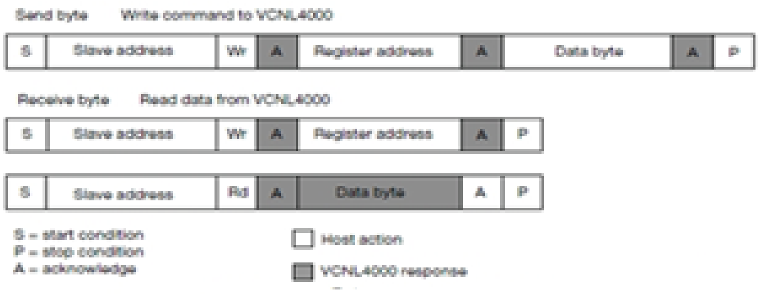
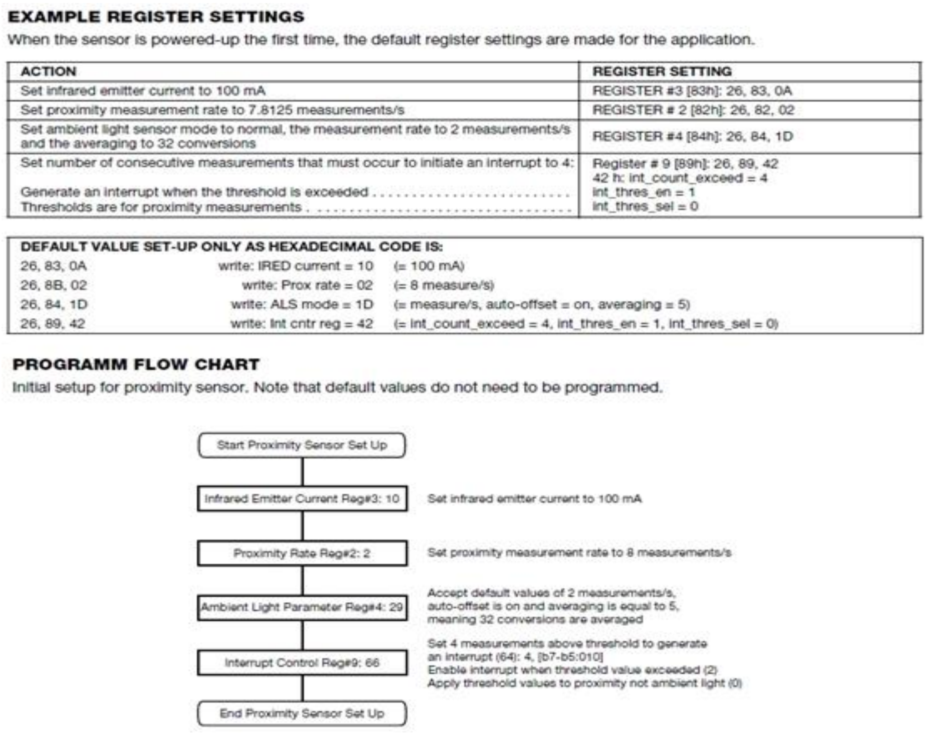


Figure 2.6: I2C Interface Concept diagram [37]

The VCNL4000 registers will be read using I2C protocol, the (Figure 2.7) explains the register settings and flow chart.



EXAMPLE REGISTER SETTINGS

When the sensor is powered-up the first time, the default register settings are made for the application.

ACTION	REGISTER SETTING
Set infrared emitter current to 100 mA	REGISTER #3 [83h]: 26, 83, 0A
Set proximity measurement rate to 7.8125 measurements/s	REGISTER # 2 [82h]: 26, 82, 02
Set ambient light sensor mode to normal, the measurement rate to 2 measurements/s and the averaging to 32 conversions	REGISTER #4 [84h]: 26, 84, 1D

Set number of consecutive measurements that must occur to initiate an interrupt to 4:
 	
Generate an interrupt when the threshold is exceeded . . . .
Thresholds are for proximity measurements . . . .
	Register # 9 [89h]: 26, 89, 42
	42 h: int_count_exceed = 4
	int_thres_en = 1
	int_thres_sel = 0

DEFAULT VALUE SET-UP ONLY AS HEXADECIMAL CODE IS:		
26, 83, 0A	write: IRED current = 10	(= 100 mA)
26, 8B, 02	write: Prox rate = 02	(= 8 measure/s)
26, 84, 1D	write: ALS mode = 1D	(= measure/s, auto-offset = on, averaging = 5)
26, 89, 42	write: Int cntr reg = 42	(= int_count_exceed = 4, int_thres_en = 1, int_thres_sel = 0)

PROGRAMM FLOW CHART

Initial setup for proximity sensor. Note that default values do not need to be programmed.

Start Proximity Sensor Set Up
Infrared Emitter Current Reg#3: 10
Set infrared emitter current to 100 mA
Proximity Rate Reg#2: 2
Set proximity measurement rate to 8 measurements/s
Ambient Light Parameter Reg#4: 29
Accept default values of 2 measurements/s, auto-offset is on and averaging is equal to 5, meaning 32 conversions are averaged
Interrupt Control Reg#9: 66
Set 4 measurements above threshold to generate an interrupt (64): 4, [b7-b5:010]
Enable interrupt when threshold value exceeded (2)
Apply threshold values to proximity not ambient light (0)
End Proximity Sensor Set Up

Figure 2.7: Register settings and flowchart [37]

2.2.2 Code Development

Based on the (Figure 2.7) flow chart, the following steps are used to communicate to the VCNL according to the I2C protocol:

1. Set the LED current and other options. Some options, such as the blinking frequency, has default option, and is therefore not mandatory to set.
2. Set the command register.
3. Set the proximity register.
4. Read and write the value using HalI2CRead(), HalI2CWrite() function.

5. Reset the command register.

Available sample codes were leveraged to design the sensor as a heart rate monitor. The <simpleBLEperipheral.c> file is a sample code which is developed according to the project requirement. In this project the file had all basic functions required for the application, although there were few added from <keyfob.c>.

The <simpleBLEperipheral> project doesn't include header files for I2C implementation, so first the available files must be downloaded. There are two files, <hal_i2c.c> and <hal_i2c.h>. Add these files in the source and target folders accordingly. The header file is located at: <*C:*\TexasInstruments\BLE-CC254x1.2.1\Components\hal\include>. The compiler file is located at: <C:\TexasInstruments\BLE-CC254x-1.2.1\Components\hal\target\CC2540EB>. The code functions when the below two variables are defined in <simpleBLEPeripheral.c>, <hal_i2c.h> and <ioCC2541.h>.

#define HAL_I2C TRUE

#define HAL_I2C_MASTER TRUE

These functions must be defined prior to this part of the below function:

#if (defineHAL_I2C) && (HAL_I2C==TRUE)

After defining and implementing the changes, the code is compiled. The <hal_i2c.h> edited code should be added to IAR workbench hal_i2c.h code. Referring to the <HAL_API.pdf> manual, carefully verify <hal_i2c.h> file when using the below function. In this project, three functions are used, HalI2CInit, HalI2CRead and HalI2CWrite.

For example, HalI2CInit (uint8 address, i2cClock_t clockRate);

It should be, HalI2CInit (I2CADDR, i2cClock_123KHZ);

Uint8 address (I2CADDR) comes from <ioCC2541.h>, and i2cClock_t clockRate (i2cClock_123KHZ) comes from <hal_i2c.h>.

To design an appropriate testing code, the <HAL Driver API.pdf> file should be verified for more functions. The <simpleBLEPeripheral.c> file contains <performPeriodicTask> section. The performed periodic task is an event generated function which is called every 30 milliseconds. The output received is the raw data from the sensor. Data is collected from the sensor at a rate of 33.3Hz (sampling period 30 ms), and saved into a buffer of size 100. Figure 2.8 shows 100 points of raw data.

The perform periodic task function is an event generated task which occurs every 30 milliseconds. Choosing a smaller time interval i.e. less than 30 milliseconds was shown to reduce the reliability of Bluetooth data transmission. Bluetooth generally requires a minimum time to initialize and execute other tasks in the OSAL layer before executing the application layer. The application layer consists of the Bluetooth stack. The Bluetooth low energy specification states throughputs of 0.27 Mbits/sec, which should be more than enough bandwidth to support faster data transmissions below 30 ms. However, in experiments with the Texas Instruments CC2540 chip, BLE communication proved unreliable when the acquisition time was set <30 ms. With troubleshooting, higher speed may be possible in the future. For the present application however, the periodic interval of 30 ms, corresponding to a data rate of 30.3 Hz, is sufficient for heart rate monitoring. Other wearable heart rate monitors, such as a wrist watch operate at 12HZ, was adequate for PPG sampling [39].

2.2.3 Signal Processing

The raw heart rate data received from the proximity sensor must be signal processed to remove noise and offset, and then calculate the r-r intervals (the time between heart contractions), and the beats per minute (BPM). Implementation of signal processing onboard the microcontroller allows the system to send the BPM to the smartphone app, rather than the raw data, which would require roughly a 10X higher data transmission rate. The power cost of onboard signal processing is small compared to the power required for wireless transmission.

The first signal processing step is a 3 point median filter which was implemented to remove the spurious noise. The median filter offers simple implementation and is more suitable for removing large spikes in the signal due to digital transmission errors. (Figure 2.8) compares typical raw data with the median filtered data.

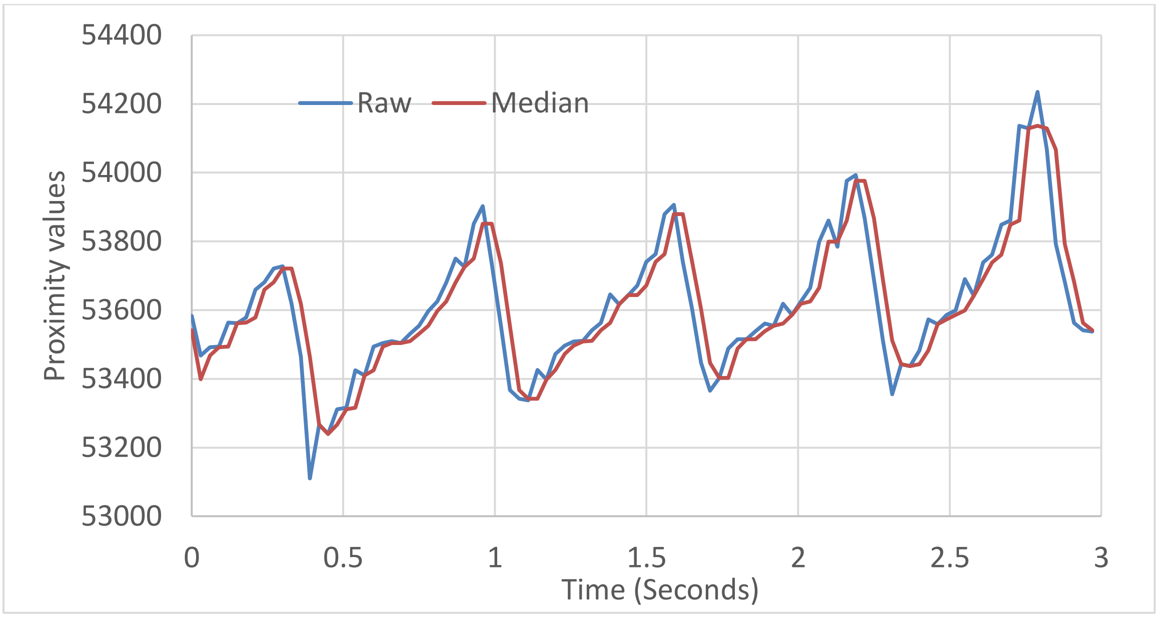


Figure 2.8: Raw heart rate and median filter data of 100 points plotted in 30 milliseconds

After the signal is median filtered, the 2nd step is to remove DC offset using a moving average high pass filter. In a typical PPG raw signal obtained from the heart rate sensor, the AC component is only 1-5% of the total signal [16]. We first low pass filter the signal using a moving average low pass filter. The high pass filtered signal is then obtained by subtracting the low pass filtered signal from original signal.

The moving average low pass filter is implemented recursively, and can be explained as follows. Consider a buffer which contains the last 5 data values from the heart rate sensor: d1, d2, d3, d4, d5. The earliest value is d1, followed by the others. When the pointer moves the present – next value i.e. d2-d3 is divided by 5 which is added with d1 to give an output of E1 for the first iteration and then when the pointer moves again E1+(d3-d4)/5 = E2 for the second iteration. The recursive formulae is used, which is the previous value of moving average + the current data point - the earliest data point. The moving average filter acts as a low pass filter, with the pass frequency set by the length of the filter. In the above illustrative example, the filter length is 5, but in our implementation it was set to 100 to select a lower transition frequency.

To obtain the high pass filtered signal, the low pass filtered signal is subtracted from the median filtered signal. Given the raw PPG waveform in (Figure 2.8), the moving average filtered signal and the high pass filtered signal are shown in (Figure 2.9) and (Figure 2.10), respectively. The moving average filtered signal tracks the DC value of the PPG waveform, and the high pass filtered is similar to the median filtered signal, but with zero offset.

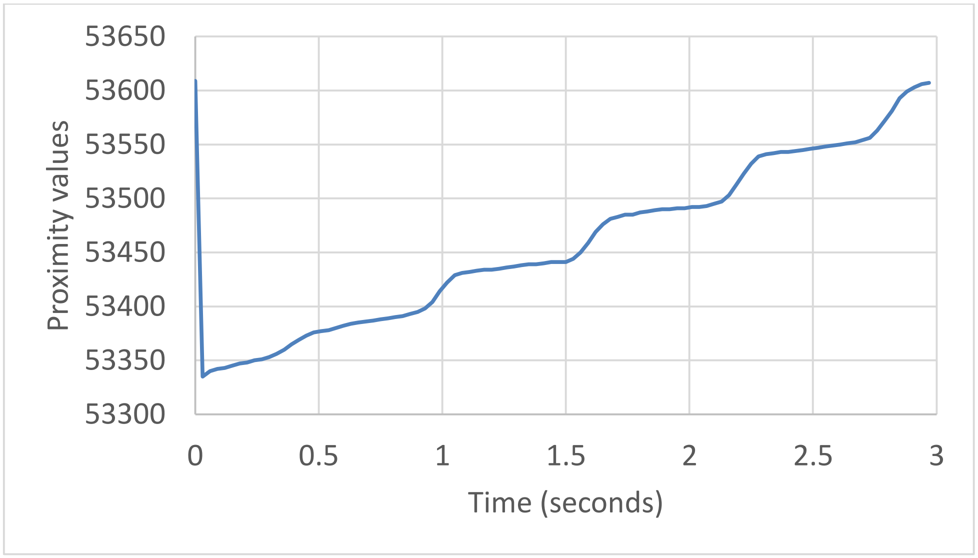


Figure 2.9: Moving average filtered data

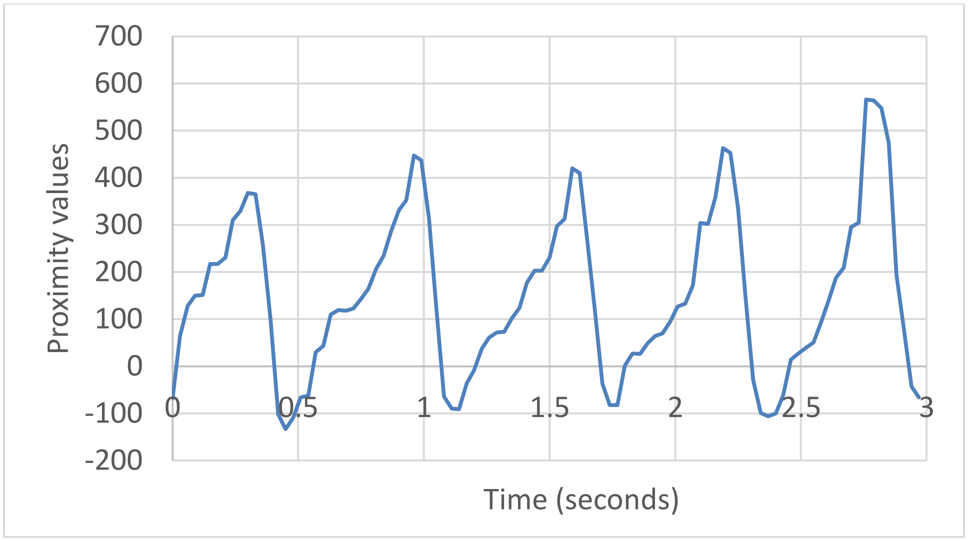


Figure 2.10: High pass filtered data.

To calculate RR-intervals and BPM, a threshold crossing detector is implemented. When the signal crosses a predefined threshold (100), the time is recorded using the OSAL system timer. At each crossing, the r-r interval is measured by calculating the time between the current and previous peak. The formulae BPM=60000(in milliseconds)/r-r interval is used to obtain BPM. Further testing of the heart rate sensor is shown in the subsequent chapter. Specifically, it explores the effect of changing the moving average filter length, the LED current, acquisition rate, and threshold settings.

2.3 Temperature sensor

The sensor TMP112 is used to measure the skin temperature. This temperatures sensor is a low power two-wire sensor. This high precision sensor does not require external components. It reads digital temperature to a resolution of 0.0625 degree centigrade. This device operates with a temperature range less than -40 degree centigrade. The temperature of the skin is measured by touching the sensor with a tip of the finger. The 12-bit resolution sensor requires a maximum of 3.6 volts and minimum of 1.4 volts with a location accuracy of 0.5(+/-). An attractive feature in this sensor is the programming alert, which sends an interrupt when the temperature crosses a threshold value; this can potentially be used in the future for triggered detection. The design here explains on how to build the sensor.

The temperature sensor is first mounted onto the breadboard for prototype design, as per the pin configurations the connections are made as shown below.

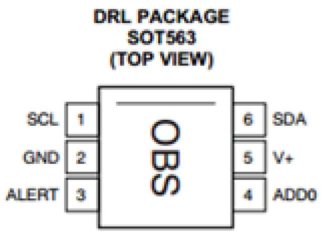
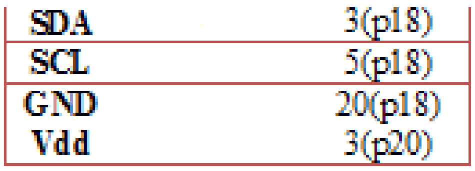


Figure 2.11: Pin diagram [40]

The TI CC2541 pins are shown below:



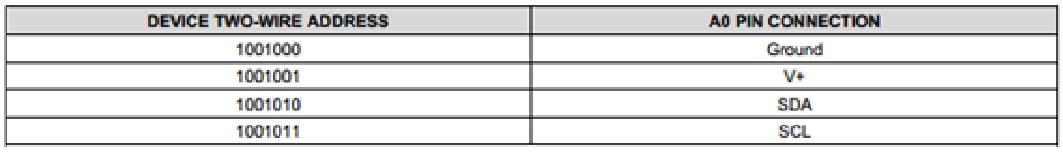
SDA	3(p18)
SCL	5(p18)
GND	20(p18)
Vdd	3(p20)

Figure 2.12: CC2541 I2C connections for the pin [40]

I2C protocol is used to communicate between the sensor and the CC2540 microcontroller. The microcontroller pins SDA, SCL are connected to the temperature sensor SDA, SCL pins. SDA and SCL are open drain bus lines used for I2C communication. The connections are made accordingly to read the register values.

I2C communication is enabled by setting up the HALI2C command in the CC2540 system firmware. To initiate communication between the CC2540 and sensor, one must first handshake the sensor and the microcontroller by setting up the connection after initializing the I2C clock.

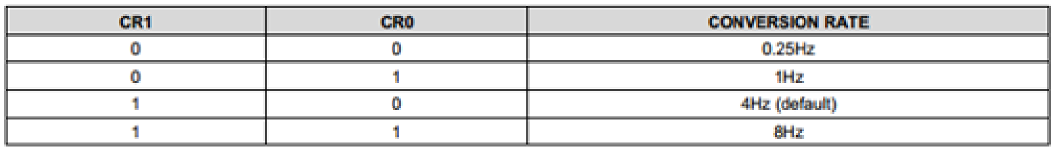
Read/write commands are then sent to the “who am I?” register, a command register value which is read from the sensor to verify if the sensor is communicating. The sensor supports transmission protocol between 1 kHz to 400 kHz [40]. Initializing the write values to the register gives the temperature values. The I2C slave address of the TMP112 can be configured depending on the A0 pin connection (Figure 2.13).



DEVICE TWO-WIRE ADDRESS	A0 PIN CONNECTION
1001000	Ground
1001001	V+
1001010	SDA
1001011	SCL

Figure 2.13: Pin and slave address [40]

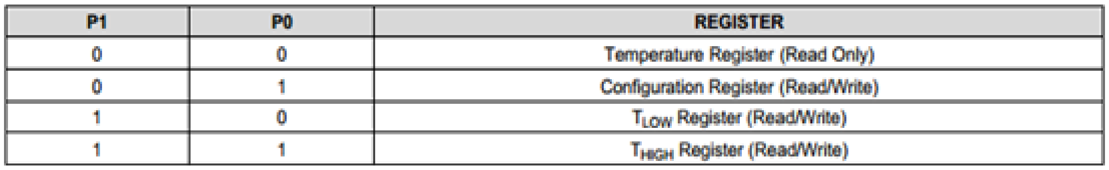
The sensor is initialized by writing the 0x14 command with a clock of 144 kHz. The default mode of the TMP112 device is continuous conversion mode. During this mode, the ADC performs continuous temperature conversions and stores the output in a CC2541 buffer to read. The conversion rate settings as shown in (Figure 2.14) are used to set the conversion at particular rate required in the application. By default the value is set to 4 Hz, and the conversion time is 26ms.



CR1	CR0	CONVERSION RATE
0	0	0.25Hz
0	1	1Hz
1	0	4Hz (default)
1	1	8Hz

Figure 2.14: Conversion rate settings [40]

After setting 4 Hz acquisition rate, the p0 and p1 will be set to 0 to read value from temperature register.



P1	P0	REGISTER
0	0	Temperature Register (Read Only)
0	1	Configuration Register (Read/Write)
1	0	T_{LOW} Register (Read/Write)
1	1	T_{HIGH} Register (Read/Write)

Figure 2.15:Pointer address[40]

The write register timing diagram is as shown below.

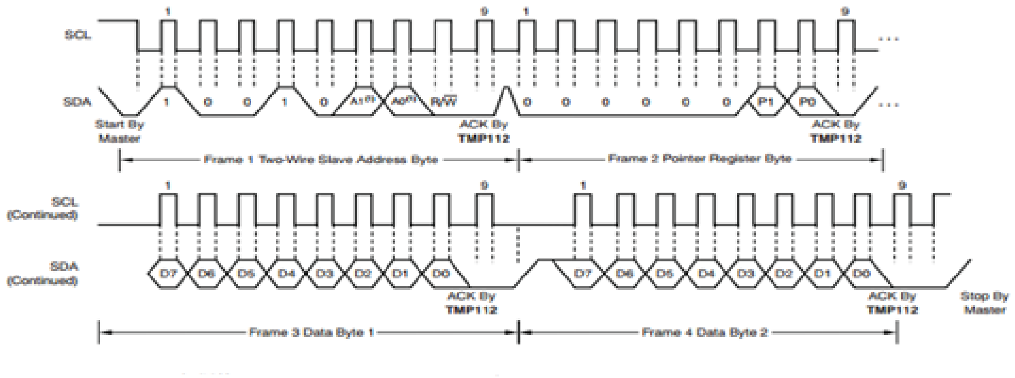


Figure 2.16: Two-wire timing diagram for write word format [40]

The timing diagram for read register is as shown below:

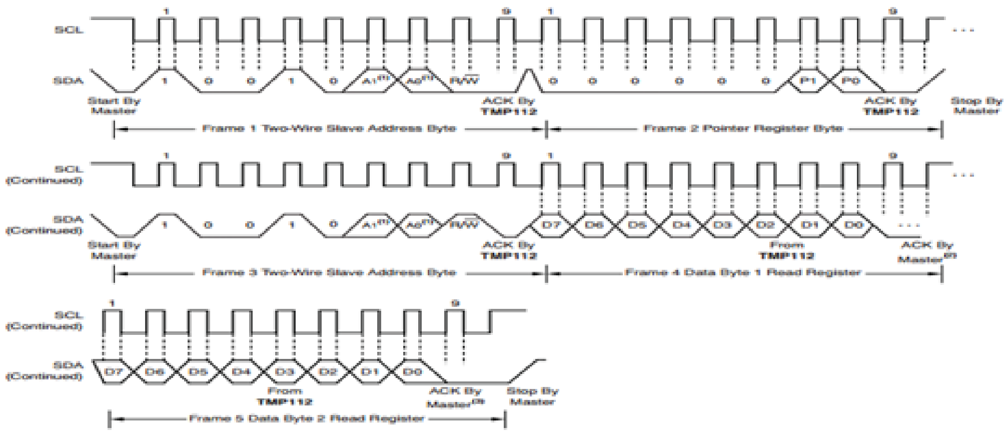
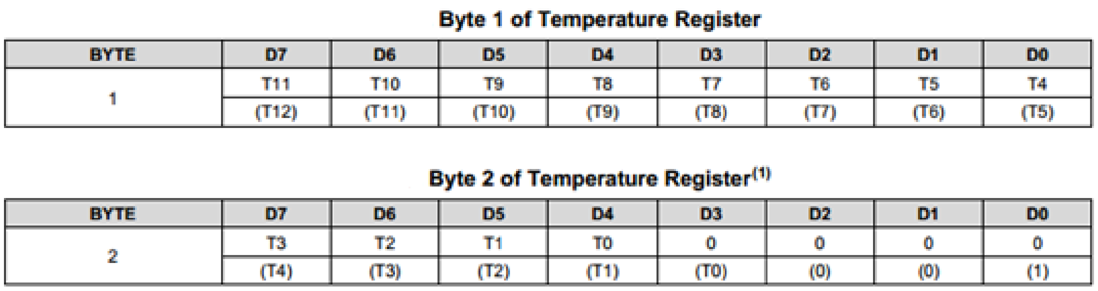

SCL
SDA
Start By Master
ACK By TMP112
Stop By Master
Frame 1 Two-Wire Slave Address Byte
Frame 2 Pointer Register Byte
SCL (Continued)
SDA (Continued)
From TMP112
ACK By Master
Frame 3 Two-Wire Slave Address Byte
Frame 4 Data Byte 1 Read Register
Frame 5 Data Byte 2 Read Register

Figure 2.17: Two-wire timing diagram for read word format [40]

Here A0, A1 represents the read and write values and P0, P1 represents the sensor enable values. Read the temperature register value into the buffer of the microcontroller. The register of the TMP112 device is configured as a 12-bit read-only register that stores the most recent conversion. Two bytes have to be read from the sensor, 1 byte from the least significant byte (LSB) and another from the most significant byte (MSB).



Byte 1 of Temperature Register

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	T11	T10	T9	T8	T7	T6	T5	T4
	(T12)	(T11)	(T10)	(T9)	(T8)	(T7)	(T6)	(T5)

Byte 2 of Temperature Register[1]

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
2	T3	T2	T1	T0	0	0	0	0
	(T4)	(T3)	(T2)	(T1)	(T0)	(0)	(0)	(1)

Figure 2.18: High and low temperature registers values [40]

The temperature values after receiving from these registers must be converted into digital format and then to temperature values. The temperature is divided by the resolution and LSB. Example: (50°C) / (0.0625°C / LSB) = 800 = 320h = 0011 0010 0000, multiply the decimal number by the resolution to obtain the temperature value:

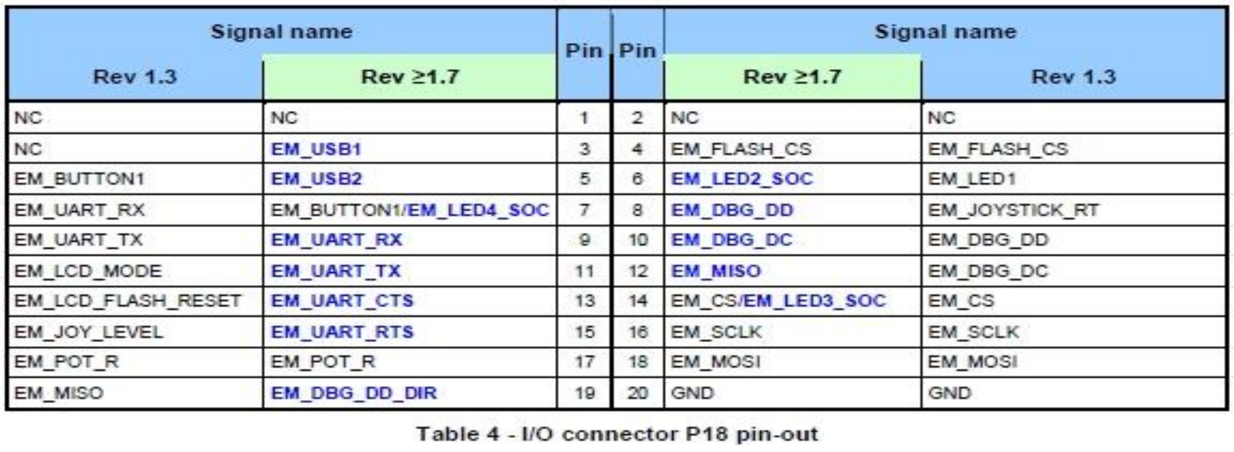
Example: 0011 0010 0000 = 320h = 800 × (0.0625°C / LSB) = 50°C

The temperature output values are demonstrated in chapter 4.

2.4 Local Flash Memory

The MX25L6445E is a serial flash memory, used to store sensor data when the device is not connected. During initial prototyping, before implementing the MX25L, the internal flash memory chip available in the SmartRF05 board was used to store the data. It was found that the data will not erase due to read/write restrictions. Hence, serial flash memory MX25L6445E was a preferred solution [41]. The device consists of 3 bus signals they are clock input (SCLK), a serial data input (SI), and a serial data output (SO). Access to the device is enabled by CS# input.

MX25L6445E latches address on both rising and falling edge. The data throughput doubles when MX25L6445E is used because it provides a high performance read mode. Moreover, this leads to direct code execution saving RAM size and cost. The continuous program mode execution is based on byte basis, or page (256 bytes) basis, or word basis. The CS# low and high is used to control the operation of the device, when the chip select is high the device enters standby mode which draws less than 100uA current [41]. (Figure 2.19) shows the input/output pin-out. The CC2541 board hardware schematics were verified for SPI pin connections. The data from the VCNL4000 is communicated through I2C but the value is stored into the SPI buffer.



Signal name		Pin	Pin	Signal name	
Rev 1.3	Rev ≥1.7			Rev ≥1.7	Rev 1.3
NC	NC	1	2	NC	NC
NC	EM_USB1	3	4	EM_FLASH_CS	EM_FLASH_CS
EM_BUTTON1	EM_USB2	5	6	EM_LED2_SOC	EM_LED1
EM_UART_RX	EM_BUTTON1/EM_LED4_SOC	7	8	EM_DBG_DD	EM_JOYSTICK_RT
EM_UART_TX	EM_UART_RX	9	10	EM_DBG_DC	EM_DBG_DD
EM_LCD_MODE	EM_UART_TX	11	12	EM_MISO	EM_DBG_DC
EM_LCD_FLASH_RESET	EM_UART_CTS	13	14	EM_CS/EM_LED3_SOC	EM_CS
EM_JOY_LEVEL	EM_UART_RTS	15	16	EM_SCLK	EM_SCLK
EM_POT_R	EM_POT_R	17	18	EM_MOSI	EM_MOSI
EM_MISO	EM_DBG_DD_DIR	19	20	GND	GND

Table 4 - I/O connector P18 pin-out

Figure 2.19: I/O connector p18 pin-out of TI CC2541

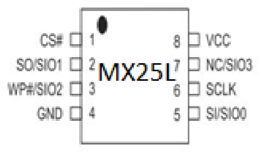


Figure 2.20: MX25L Pin Connections [41]

The MX25l Pin connections are connected accordingly to the figures above. The memory chip has 4 pins SCLK (serial clock), CS (chip select), MISO (SPI data master input slave output), and MOSI (SPI data master output slave input). In the above description the pins are connected to the serial flash memory. The way SPI works is similar to I2C but the chip select pin must be enabled and disabled in CC2541 to communicate to the sensor.

The firmware requires the SPI <flash.c> library. Here, the function <spiWriteByte> is working as "Save", and function <spiReadByte> is working as "Load". Even though <simpleBLEperipheral> project has UART functionality in the HAL, the SPI function were required so <keyfobdemo> project had SPI functions wherein these library files <cma3000d.c> and <cma3000c.h> were copied and pasted into <simpleBLEperipheral> code. After copying these files to the source directory <*C:*\TexasInstruments\BLECC254x1.2.1\Projects\ble \simpleBLEPeripheral\Source>, in <cma3000d.h> files, we can observe these two functions for SPI. In this library files it can be observed that the SPI read and write functions are available along with chip select and clock. Hence, these functions helped to build the code directly in <simpleBLEperipheral> without any major changes.

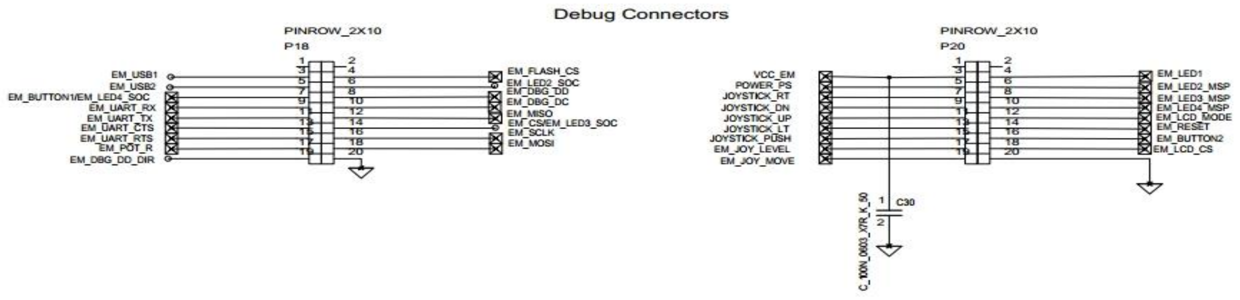
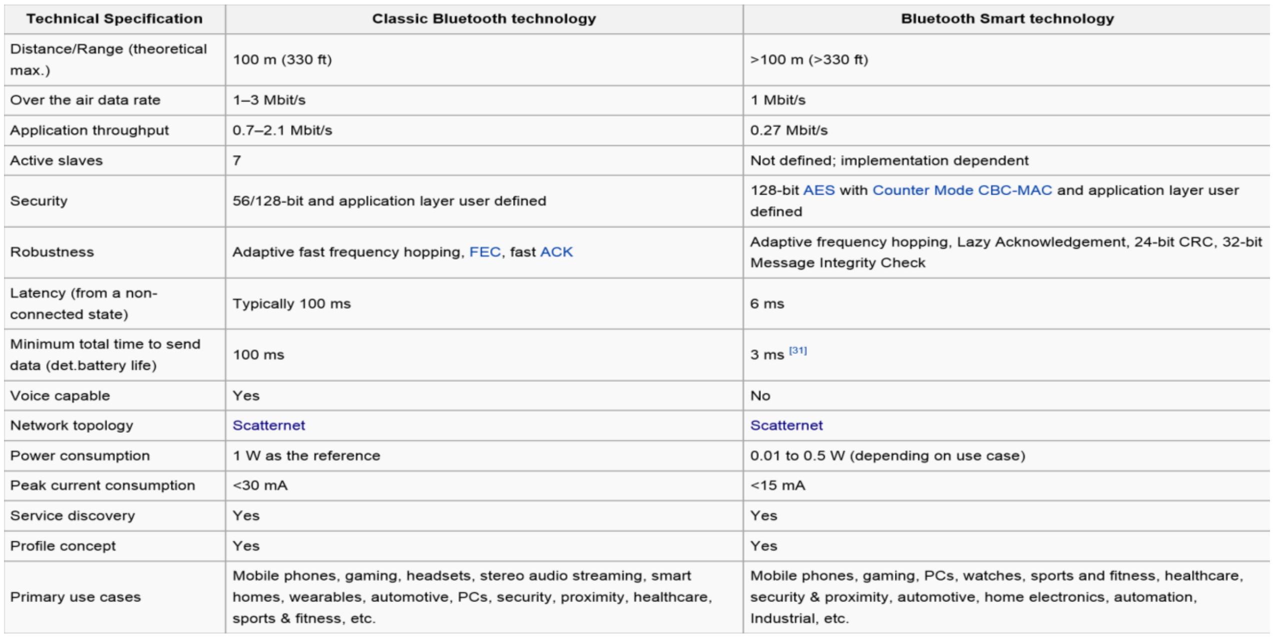


Figure 2.21: Debug connector for SPI connection

In (Figure 2.21), the debug connectors are connected to the Macronix memory chip. In typical operation, the proximity sensor data is first read using I2C and then the I2C buffer is read by the SPI buffer. First, the SPI chip select is enabled, then the read mode is initialized and then page programming method is selected. The preferred size is selected to store data in flash, and then the data from I2C buffer is read and loaded into the size selected. Later, the stored data is read in UART terminal, as shown in the following chapter.

2.5 Bluetooth Low Energy (BLE)

Bluetooth protocol is designed to communicate low bandwidth data over a short distance. There are currently existing two different kinds of Bluetooth, one is Bluetooth Low Energy (BLE) and another is classic Bluetooth[42]. Also called Bluetooth Smart, BLE consumes less power and is cheaper compared to classic Bluetooth, by maintaining the similar range of communication. Bluetooth is designed for applications which require short range communication such as medical care, fitness, and home applications [43]. They both operate at 2.4 GHz frequencies.



Technical Specification	Classic Bluetooth technology	Bluetooth Smart technology
Distance/Range (theoretical max.)	100 m (330 ft)	>100 m (>330 ft)
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.27 Mbit/s
Active slaves	7	Not defined; implementation dependent
Security	56/128-bit and application layer user defined	128-bit AES with Counter Mode CBC-MAC and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check
Latency (from a non-connected state)	Typically 100 ms	6 ms
Minimum total time to send data (det.battery life)	100 ms	3 ms [31]
Voice capable	Yes	No
Network topology	Scatternet	Scatternet
Power consumption	1 W as the reference	0.01 to 0.5 W (depending on use case)
Peak current consumption	<30 mA	<15 mA
Service discovery	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, gaming, headsets, stereo audio streaming, smart homes, wearables, automotive, PCs, security, proximity, healthcare, sports & fitness, etc.	Mobile phones, gaming, PCs, watches, sports and fitness, healthcare, security & proximity, automotive, home electronics, automation, Industrial, etc.

Figure 2.22: Comparison between Bluetooth low energy and Classic Bluetooth [42][43]

Even though they are wireless technologies using Bluetooth there are many advantages and disadvantages. The advantages are that it is easy and inexpensive to implement Bluetooth, and they can communicate within the range of 100m. But disadvantages include sending limited data at a time. Bluetooth low energy cannot be used for voice communication due to its data bandwidth requirements. Bluetooth low energy master and slave communication requires GATT and GAP roles implemented [43] which is secure compared to Classic Bluetooth.

2.5.1 GAP ROLES

Gap roles are four profile roles wherein the Bluetooth low energy can operate in [43]:

1) **Observer:** Observer scans for advertisements. Example: Temperature display
2) **Broadcaster**: Connection cannot be established it just broadcasts the data, Example: Temperature sensor.
3) **Peripheral:** Advertises and checks for connections after connection established, acts as a slave.
4) **Central**: Scans for advertisers, acts a master in connection

2.5.2 GATT ROLES

GATT roles send out commands to the client to discover profiles. The profiles are characteristic profiles which are designed to communicate data to and fro from the client. Every GATT role has a handle and notification which are designed using the profiles in the GATT table(user defined). The <simpleBLEperipheral> code consists of <simplegatt.c> file explains all the profiles which are designed. Major difference between GATT role, profile and table is that every GATT role should either have a handle or a notification which are designed in the GATT table(user defined).There is some information to be used by the server and the GATT profiles provide those commands [44]

Discover UUIDs for all primary services

- For the available UUID Find a service

- Find the primary and secondary services
- Discover characteristics for a service
- For a UUID find the matching characteristic
- Read descriptors

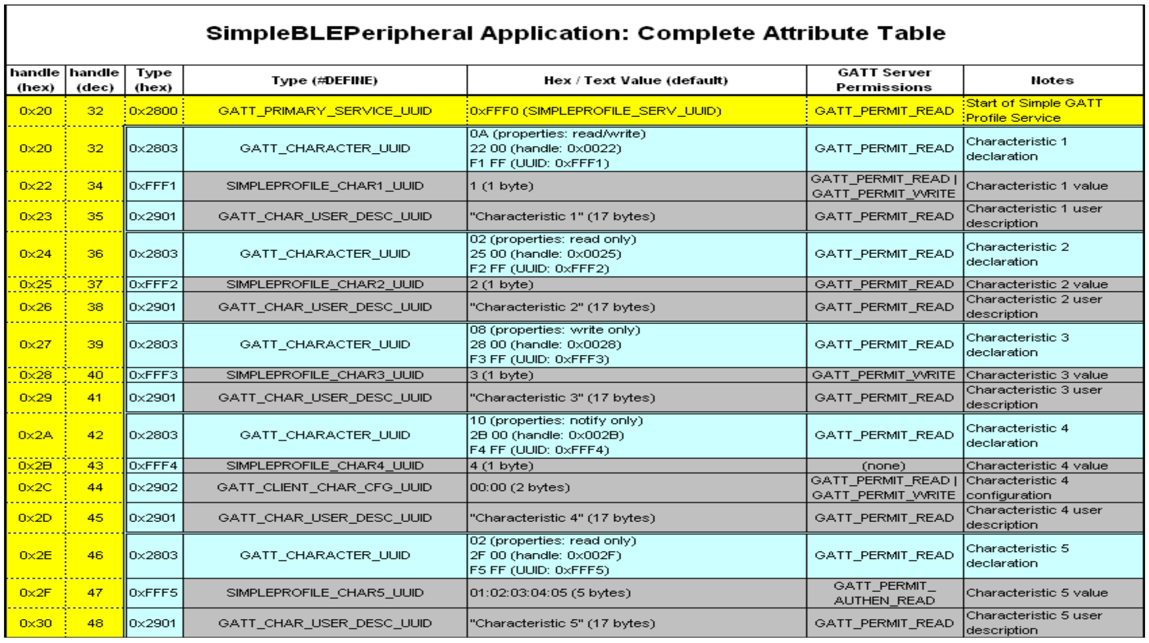
In addition, there are commands which provide data transfer from server to client and vice versa:

- Characteristic UUID value can be sued to determine the handle value.
- Write operations always identify response from a server for the particular characteristic UUID

The client can send a request to the server for the notification values. For example, whenever the sensor takes measurements it can request its client for notifications. When the CC2540 USB dongle is connected to the PC, the Btool will generate commands over the air to read and write the data, and the <simpleProfile_WriteAttrCB> will write a value to the characteristic profiles. The <simpleProfile_ReadAttrCB> will read a value from a characteristic profile. These are registered in the <simpleBLEPeripheral_SimpleProfileCBs> this is a default file which is already available. The <simplegatt.h> and <simplegatt.c> files should contain all the profiles expansion with a new one as defined for the previous ones. Once the header is defined with new profiles are added into file.

2.5.3 GATT Table for simpleBLEperipheral

The designing of characteristic profiles is a key part of Bluetooth low energy links. The Characteristic profile has different attributes for each service (e.g. temperature). The characteristic is a hex value which is defined in the table for a particular service and all these services with characteristics are called attributes. Every attribute has a UUID. Every attribute is given a unique notification or handle [44]. There are 128-bit UUID which can used for custom build. The GATT table in Figure 2.23 is a basic overview of all the attributes used in our application. The data from the sensors like heart rate data, skin temperature data, are loaded into the attributes. Some are designed to act only on notifications and these profiles only read the real-time data. There are also profiles which are password enabled. In the <simpleBLEperipheral> application, the sample file <simplegatt.h> and <simplegatt.c> are used to build the GATT table [44].


SimpleBLEPeripheral Application: Complete Attribute Table

handle (hex)	handle (dec)	Type (hex)	Type (#DEFINE)	Hex / Text Value (default)	GATT Server Permissions	Notes
0x20	32	0x2800	GATT_PRIMARY_SERVICE_UUID	0xFFF0 (SIMPLEPROFILE_SERV_UUID)	GATT_PERMIT_READ	Start of Simple GATT Profile Service
0x20	32	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 22 00 (handle: 0x0022) F1 FF (UUID: 0xFFF1)	GATT_PERMIT_READ	Characteristic 1 declaration
0x22	34	0xFFF1	SIMPLEPROFILE_CHAR1_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Characteristic 1 value
0x23	35	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 1" (17 bytes)	GATT_PERMIT_READ	Characteristic 1 user description
0x24	36	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 25 00 (handle: 0x0025) F2 FF (UUID: 0xFFF2)	GATT_PERMIT_READ	Characteristic 2 declaration
0x25	37	0xFFF2	SIMPLEPROFILE_CHAR2_UUID	2 (1 byte)	GATT_PERMIT_READ	Characteristic 2 value
0x26	38	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 2" (17 bytes)	GATT_PERMIT_READ	Characteristic 2 user description
0x27	39	0x2803	GATT_CHARACTER_UUID	08 (properties: write only) 28 00 (handle: 0x0028) F3 FF (UUID: 0xFFF3)	GATT_PERMIT_READ	Characteristic 3 declaration
0x28	40	0xFFF3	SIMPLEPROFILE_CHAR3_UUID	3 (1 byte)	GATT_PERMIT_WRITE	Characteristic 3 value
0x29	41	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 3" (17 bytes)	GATT_PERMIT_READ	Characteristic 3 user description
0x2A	42	0x2803	GATT_CHARACTER_UUID	10 (properties: notify only) 2B 00 (handle: 0x002B) F4 FF (UUID: 0xFFF4)	GATT_PERMIT_READ	Characteristic 4 declaration
0x2B	43	0xFFF4	SIMPLEPROFILE_CHAR4_UUID	4 (1 byte)	(none)	Characteristic 4 value
0x2C	44	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Characteristic 4 configuration
0x2D	45	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 4" (17 bytes)	GATT_PERMIT_READ	Characteristic 4 user description
0x2E	46	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 2F 00 (handle: 0x002F) F5 FF (UUID: 0xFFF5)	GATT_PERMIT_READ	Characteristic 5 declaration
0x2F	47	0xFFF5	SIMPLEPROFILE_CHAR5_UUID	01:02:03:04:05 (5 bytes)	GATT_PERMIT_ AUTHEN_READ	Characteristic 5 value
0x30	48	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 5" (17 bytes)	GATT_PERMIT_READ	Characteristic 5 user description

Figure 2.23: GATT Table [44]

2.6 OSAL

The OSAL is the operating system abstraction layer and it's a layer where the entire software architecture is built. The OSAL consists of interrupts for the tasks in the main code [45]. There are 12 tasks in the OSA, and each is assigned a priority number. The lower the priority number, the higher the priority it is assigned. In the <simpleBLEapplication>, the final task i.e the 12th task is the <simpleBLEapplication>, it is related to <perform_periodic_task> in the main code. This task consists of events to be executed. For example, if the event is pressing a button then when the button is pressed an event is created and the system will wake up from the sleep mode to perform the specific task [45]. There are few key API functions which must be used:

- osal_init_system: Initializes OSAL
- osal_start_system: Starts OSAL main loop
- osal_set_event: Sets OSAL event for a task
- osal_msg_allocate: allocates memory for OSAL
- osal_msg_send: Sends a message to a specific task,
- osal_msg_deallocate: Deallocates an OSAL.

For timers and clocks,

- osal_start_timer: Sets an OSAL event schedule time for a task
- osal_stop_timer: Cancels an existing event,
- osal_start_timerEx: start a timeout for a specific task
- osal_stop_timer:Ex :stop indicated timer
- osal_GetSystemClock: reads clock Memory
- osal_setClock: initializes real-time clock
- osal_getClock:retrieve the time
- osal_ConvertUTCTime: time in seconds

2.7 Non Volatile Memory

The Non volatile memory is used to store the Security key for the master and slave to connect. An identifier is defined for a data structure <ZComDef.h>, and uses the APIs to read and write to/from this structure; on start-up, the NV_RESTORE is a compile option that allows restoring dynamic data like security keys and routing information. Osal_nv_item_init: init is an item in non-volatile memory, osal_nv_read: reads item, osal_nv_write: writes item, osal_offsetof: calculates memory offset.

2.8 HAL

HAL is a hardware abstraction layer and it's the driver API for the board which is defined for all peripherals. The peripherals include the ADC, UART, SPI, I2C, Flash, Timers, Keys, Sleep, LED, and LCD. In our application we use everything apart from Keys. I2C and SPI are two most important protocols used in this project. The data rate in I2C is 3.4MHZ whereas the data rate in SPI is 10MHZ [45]. The SPI data rate is very high compared to I2C. There is a big difference between the HAL and OSAL, the hardware platform is HAL and the software platform is OSAL. The OSAL provides memory management, scheduling and messaging features, whereas the HAL provides programming access to hardware [45]. To understand the difference between the HAL timers and OSAL timers, the HAL initializes the hardware drivers required for the operating system to initialize all the peripherals. The OSAL controls the peripherals to perform specific task.

2.9 Clocks, oscillators, Timers and power modes

The CC2541 consists of 4 oscillators, of which 2 are high frequency oscillators and 2 are low frequency oscillators. The high frequency oscillators include a 32MHZ crystal oscillator and 16MHZ RC oscillator, which can be used for system clocks. The high frequency oscillators consume more power, and are used when the microcontroller is active. The 16KHZ and 32KHZ are available low frequency oscillators which are used as sleep timers and watch dog timers. The

Bluetooth RF transceiver requires a 32KHZ crystal oscillator and for some applications the start time is too long using this oscillator. The 32KHZ XOSC is designed to operate in 32.768KHZ for timing accuracy.

There are five different power modes, they are active mode, idle mode, PM1 mode, PM2 mode, PM3 mode.

- **Active mode**: the fully functional mode.
- **Idle mode**: Waiting to perform functions or tasks.
- **PM1**: the voltage regulator to the digital part is on. Any of the available oscillators can be used. The system enters active mode on reset, an external interrupt, or when the Sleep Timer expires.
- **PM2**: The voltage regulator to the digital core is turned off. Any of the available oscillators can be used. The system goes to active mode on reset, an external interrupt, or when the Sleep Timer expires.
- **PM3**: The voltage regulator to the digital core is turned off, none of the oscillators work. The system goes to active mode on reset, when an external interrupt is occurred.

Here, there are 4 timers: Timer 1, Timer 3, and Timer 4 can be used as Timer/counter/PWM capture etc. Timer 1 is 16-bit, Timer3 and 4 are 8-bit timers, Timer 2 is 40-bit. It has a 16-bit counter and is a configurable timer. If the POWER_SAVING is implemented in the software then an accurate of 32 kHz clock is needed for the sleep timer to maintain the connection timing required by BLE.

2.10 CC2540 memory

The local CC2540 memory is used to run the bluetooth processor, peripherals, store data temporarily before saving to the SPI memory, and for signal processing. There are 4 different types of memory spaces in CC2540 architecture. The memory spaces are: CODE: Read only memory space for program memory, the space taken by the address is 64kb. DATA: It is a read/write data memory for 1 CPU instruction cycle, the space taken is 56 bytes. XDATA: Read/write data memory 4 CPU instruction cycles, the space taken is 64kb. The CODE and XDATA share same CPU core [45]. SFR: Read/write memory single CPU cycle, the space taken is 128 bytes [45]. The DMA allocates memory with an upper 23kb of XDATA and the rest of the banks with 32kb is mapped in the Figure 2.24. Any of the available 32 KB flash banks can be mapped in here:

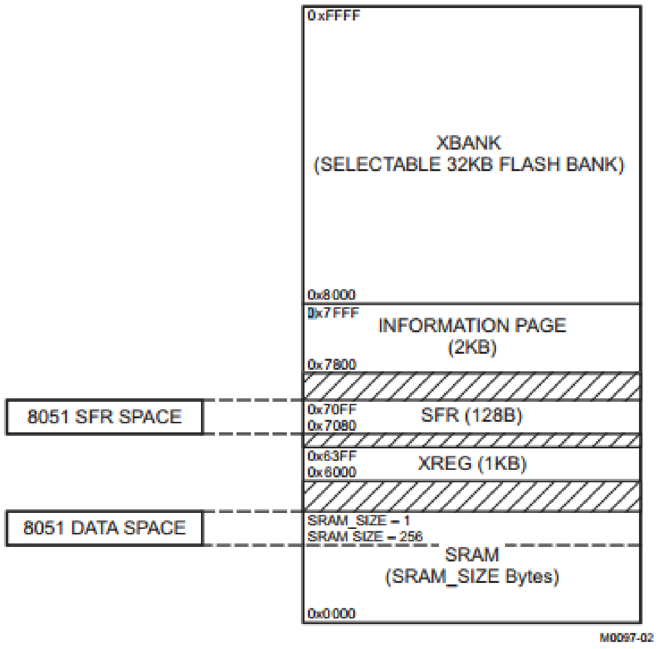


Figure 2.24: memory register

The flash memory has the following features:

- Page size: 1 KB or 2
- Flash-page erase time: 20ms
- Flash-chip (mass) erase time: 20ms
- Flash write time (4 bytes): 20µs
- Data retention (at room temperature): 100 years
- Program/erase endurance: 20,000 cycles

The XDATA memory space consists of additional XREG registers. These registers are mainly used for radio configuration and control. Below is the memory footprint for <simpleBLEperipheral> used in our system, printed using IAR workbench.

```
110 648 bytes of CODE memory
     35 bytes of DATA memory (+ 72 absolute)
  6 270 bytes of XDATA memory
    194 bytes of IDATA memory
      8 bits of BIT   memory
    581 bytes of CONST memory
```

Figure 2.25: Footprint for <SimpleBLEperipheral> application [39]

2.11 Power management

Care should be taken to reduce power consumption. The highest power consumption is during Bluetooth communication. This is somewhat reduced due to the use of Bluetooth Low Energy. The GAP role Peripheral is defined to show the amount of current consumed when the device goes to wake mode from sleep mode. The discharge times shown below assume a typical coin cell battery life with capacity 250mAh.

SimpleBLEPeripheral - *Advertising*
Time taken to wake up from sleep mode: 3.9 ms
Average time taken by an event to draw a current: 8.6 Ma for 3.9ms
Average time taken for a current to be drawn in 1s interval: 0.035 mA / ~260 days continuously

SimpleBLEPeripheral - *Notification* from master,
Time taken to wake up from sleep mode: 3.2ms
Average time taken by an event to draw a current: 8.8 mA in 3.2ms
Average time taken for a current to be drawn in 1s interval: 0.029 mA / ~316 days continuously

SimpleBLEPeripheral - *Notification* from master,
Time taken to wake up from sleep mode: 8.8ms
Average time taken by an event to draw a current: 7.89 mA in 8.8ms
Average time taken for a current to be drawn in 1s interval: 0.071 mA / ~135 days continuously

SimpleBLEPeripheral - *Read response* from master
Time taken to wake up from sleep mode: 2.8ms: 1.3ms processing after Rx/Tx
Average time taken by an event to draw a current: 8.6 mA in 2.8ms
Average time taken for a current to be drawn in 1s interval: 0.026 mA / ~352 days continuously

SimpleBLEPeripheral - *Write response* from master
Time taken to wake up from sleep mode: 3.1ms: 1.4ms processing after Rx/Tx
Average time taken by an event to draw a current: 8.8 mA in 3.1ms
Average time taken for a current to be drawn in 1s interval: 0.028 mA / ~327 days continuously

CHAPTER 3: TESTING SETUP AND METHODS

3.1 Test Setup

The Texas Instruments SmartRF05 prototype board used for firmware testing is shown in (Figure 3.1). This board allows mounting of the CC2541 sensor, and allows external connections to all the sensors on a separate breadboard. Note that in the final device, components will all be integrated on a small surface mount PCB. Using the SmartRF05 board, the CC2541 can be programmed using IAR workbench, connected via USB interface. The connections to the SmartRF05 board and sensors are designed and developed accordingly as discussed in the previous chapter. After implementing all the sensors on the breadboard the code is executed in the IAR workbench.

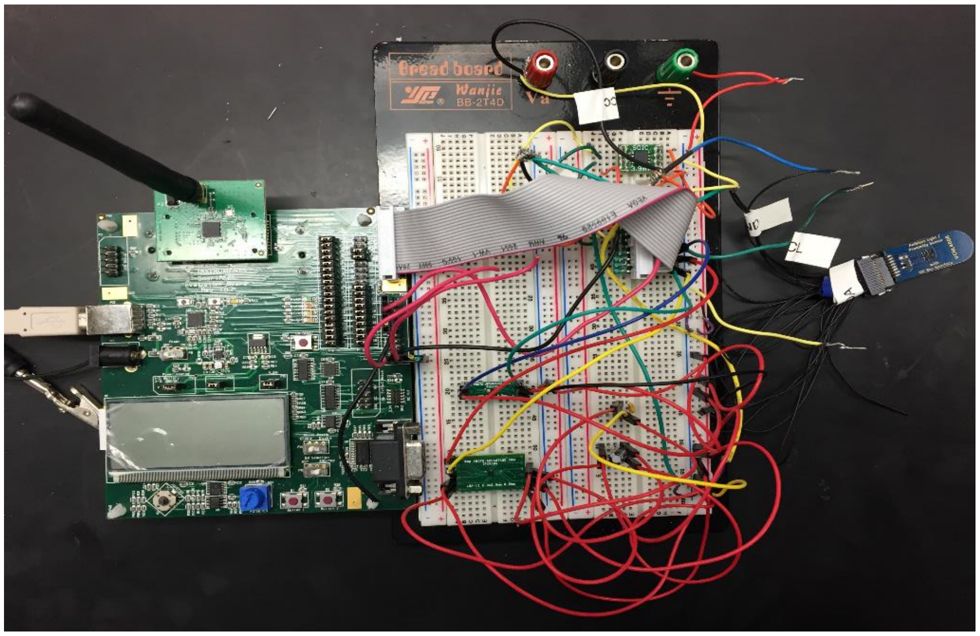
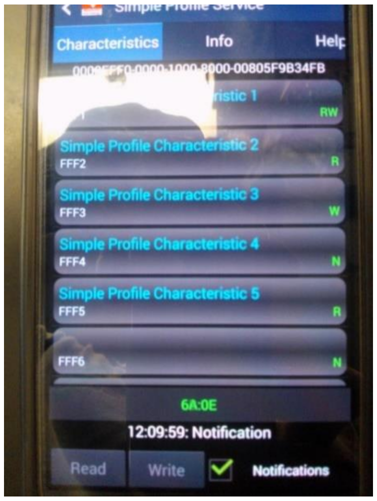
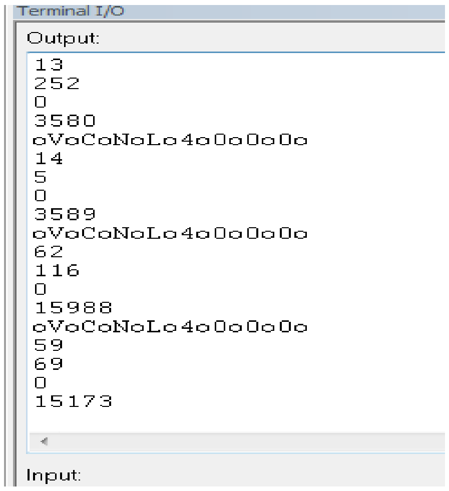


Figure 3.1: Connection setup of the Device

The below mentioned figures are the outputs from terminal, Btool and Android app, the results of the project will be demonstrated in the following chapter.

The system firmware is tested by writing the sensor values to IAR Terminal, Btool, and the Android BLE Device Monitor.


Simple Profile Service
Characteristics
Info
Help
0000FFF0-0000-1000-8000-00805F9B34FB
Simple Profile Characteristic 1
RW
Simple Profile Characteristic 2
FFF2
R
Simple Profile Characteristic 3
FFF3
W
Simple Profile Characteristic 4
FFF4
N
Simple Profile Characteristic 5
FFF5
R
FFF6
N
6A:0E
12:09:59: Notification
Read
Write
Notifications



```
 13
 252
 0
 3580
oVoCoNoLo4o0o0o0o
 14
 5
 0
 3589
oVoCoNoLo4o0o0o0o
 62
 116
 0
 15988
oVoCoNoLo4o0o0o0o
 59
 69
 0
 15173
```

Android output IAR TERMINAL OUTPUT

Figure 3.2: Output terminals using Android and IAR

3.2 Testing Tools and Methodologies

3.2.1 IAR Embedded Workbench I/O Terminal (UART/USB)

IAR Embedded Workbench (IAR Systems) is used to develop and test the firmware code. Through a USB connection to a PC or laptop, IAR also allows real time debugging. In addition to code stepping, breakpoints, and variable watches, IAR also provides a console for printf statements for debugging. Some guidelines are shown below.

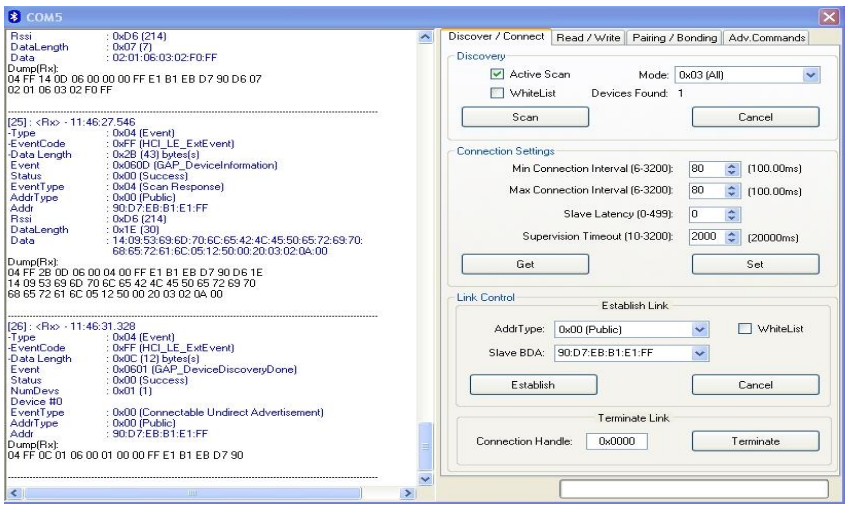
1. Use <printf> function properly. Such as printf("%d\n", data);
2. Click the Make_Restart_Debugger
3. Go to View>Terminal I/O
4. Make this code Go

3.2.2 BTOOL

Texas Instruments Btool is a PC based, Bluetooth communication tool also used for debugging. After the CC2541 from the SmartRF05 board the board reads data from various sensors, it sends the Bluetooth data to characteristic profile as designed in the <simpleBLEperipheral> code in system design.

Btool is used for reading and writing the characteristic value and printing the sensor values accordingly. Understanding the basic concept of reading and writing a characteristic value is described. To use Btool, the flash programmer SmartRF05 is installed and used to program the USB dongle and CC2541 module on development board. Here, the dongle and onboard module cannot be programmed at the same time, so they must be programmed separately. The hex file used to program these devices is available in the Texas instruments folder <*C:*/texasinstruments/blecc2541/accessories/hexfiles/cc2540_usbdongle_hosttestrelease.hex>. It is highly recommended to read through all the documents and have an in-depth understanding about reading and writing the data, which discusses configuring the respective firmwares (hex files) into the dongle and board. After the hex files is loaded, the dongle is powered up and the drivers can be detected.

Next, in the BTOOL terminal window, the port corresponding to the dongle is selected, and thee baud rate is set to 11500 and after the connection is established a Bluetooth configuration window opens as shown in (Figure 3.3).



```
COM5
 Rssi                  : 0xD6 (214)
 DataLength            : 0x07 (7)
 Data                  : 02:01:06:03:02:F0:FF
Dump(Rx):
04 FF 14 0D 06 00 00 00 FF E1 B1 EB D7 90 D6 07
02 01 06 03 02 F0 FF

------------------------------------------------------------------------
[25] : <Rx> - 11:46:27.546
-Type                  : 0x04 (Event)
-EventCode             : 0xFF (HCI_LE_ExtEvent)
-Data Length           : 0x2B (43) bytes(s)
 Event                 : 0x060D (GAP_DeviceInformation)
 Status                : 0x00 (Success)
 EventType             : 0x04 (Scan Response)
 AddrType              : 0x00 (Public)
 Addr                  : 90:D7:EB:B1:E1:FF
 Rssi                  : 0xD6 (214)
 DataLength            : 0x1E (30)
 Data                  : 14:09:53:69:6D:70:6C:65:42:4C:45:50:65:72:69:70:
                         68:65:72:61:6C:05:12:50:00:20:03:02:0A:00
Dump(Rx):
04 FF 2B 0D 06 00 04 00 FF E1 B1 EB D7 90 D6 1E
14 09 53 69 6D 70 6C 65 42 4C 45 50 65 72 69 70
68 65 72 61 6C 05 12 50 00 20 03 02 0A 00

------------------------------------------------------------------------
[26] : <Rx> - 11:46:31.328
-Type                  : 0x04 (Event)
-EventCode             : 0xFF (HCI_LE_ExtEvent)
-Data Length           : 0x0C (12) bytes(s)
 Event                 : 0x0601 (GAP_DeviceDiscoveryDone)
 Status                : 0x00 (Success)
 NumDevs               : 0x01 (1)
 Device #0
 EventType             : 0x00 (Connectable Undirect Advertisement)
 AddrType              : 0x00 (Public)
 Addr                  : 90:D7:EB:B1:E1:FF
Dump(Rx):
04 FF 0C 01 06 00 01 00 00 FF E1 B1 EB D7 90

------------------------------------------------------------------------
```

Discover / Connect | Read / Write | Pairing / Bonding | Adv.Commands

Discovery: Active Scan [✓] Mode: 0x03 (All); WhiteList [ ] Devices Found: 1; Scan; Cancel

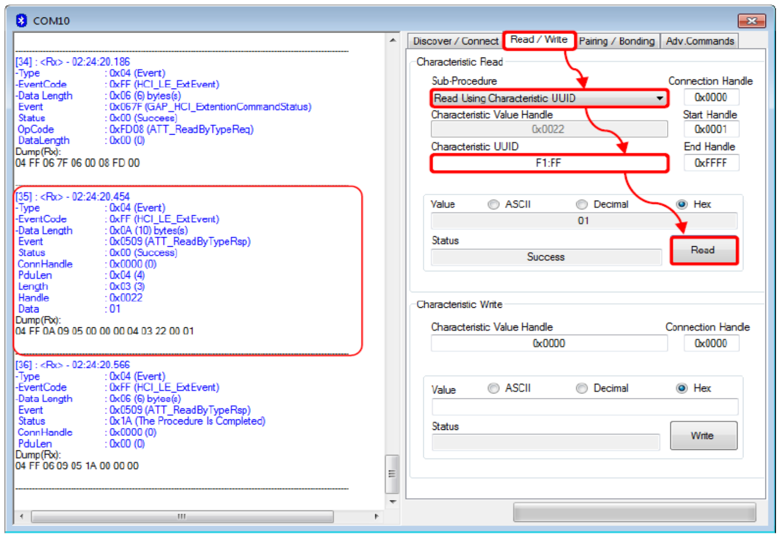
Connection Settings: Min Connection Interval (6-3200): 80 (100.00ms); Max Connection Interval (6-3200): 80 (100.00ms); Slave Latency (0-499): 0; Supervision Timeout (10-3200): 2000 (20000ms); Get; Set

Link Control — Establish Link: AddrType: 0x00 (Public); WhiteList [ ]; Slave BDA: 90:D7:EB:B1:E1:FF; Establish; Cancel

Terminate Link: Connection Handle: 0x0000; Terminate

Figure 3.3: Btool Overview

The scan button is enabled to scan for available Bluetooth slave devices. The slave BDA box in the window will display the connection ID for verification as designed in the <simpleBLEperipheral.c> file. Once it is verified, the GET, SET and ESTABLISH buttons are used to enter into read and write command window.



```
COM10
------------------------------------------------------------------------
[34] : <Rx> - 02:24:20.186
-Type                  : 0x04 (Event)
-EventCode             : 0xFF (HCI_LE_ExtEvent)
-Data Length           : 0x06 (6) bytes(s)
 Event                 : 0x067F (GAP_HCI_ExtentionCommandStatus)
 Status                : 0x00 (Success)
 OpCode                : 0xFD08 (ATT_ReadByTypeReq)
 DataLength            : 0x00 (0)
Dump(Rx):
04 FF 06 7F 06 00 08 FD 00

------------------------------------------------------------------------
[35] : <Rx> - 02:24:20.454
-Type                  : 0x04 (Event)
-EventCode             : 0xFF (HCI_LE_ExtEvent)
-Data Length           : 0x0A (10) bytes(s)
 Event                 : 0x0509 (ATT_ReadByTypeRsp)
 Status                : 0x00 (Success)
 ConnHandle            : 0x0000 (0)
 PduLen                : 0x04 (4)
 Length                : 0x03 (3)
 Handle                : 0x0022
 Data                  : 01
Dump(Rx):
04 FF 0A 09 05 00 00 00 04 03 22 00 01

------------------------------------------------------------------------
[36] : <Rx> - 02:24:20.566
-Type                  : 0x04 (Event)
-EventCode             : 0xFF (HCI_LE_ExtEvent)
-Data Length           : 0x06 (6) bytes(s)
 Event                 : 0x0509 (ATT_ReadByTypeRsp)
 Status                : 0x1A (The Procedure Is Completed)
 ConnHandle            : 0x0000 (0)
 PduLen                : 0x00 (0)
Dump(Rx):
04 FF 06 09 05 1A 00 00 00

------------------------------------------------------------------------
```

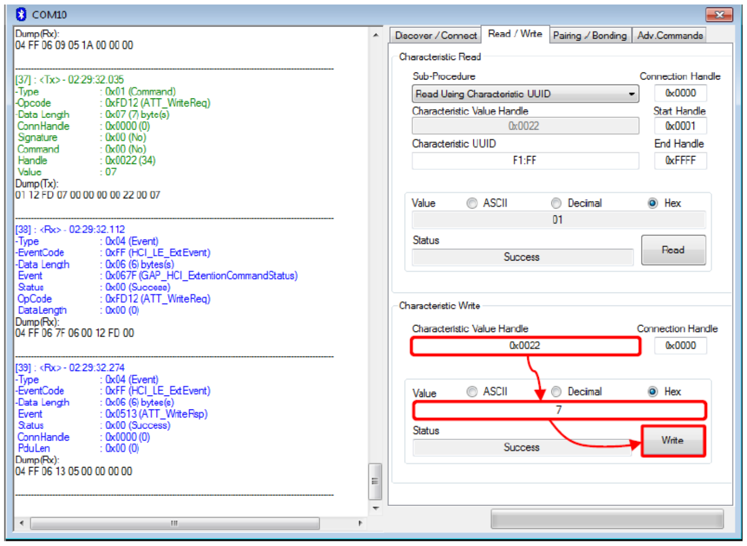
Discover / Connect | Read / Write | Pairing / Bonding | Adv.Commands

Characteristic Read: Sub-Procedure: Read Using Characteristic UUID; Connection Handle: 0x0000; Characteristic Value Handle: 0x0022; Start Handle: 0x0001; Characteristic UUID: F1:FF; End Handle: 0xFFFF; Value: ASCII ( ) Decimal ( ) [✓ Hex]: 01; Status: Success; Read

Characteristic Write: Characteristic Value Handle: 0x0000; Connection Handle: 0x0000; Value: ASCII ( ) Decimal ( ) [✓ Hex]; Status; Write

Figure 3.4: BTOOL, read a characteristic by UUID [43]

(Figure 3.4) shows the read and write characteristics window. The read/write tab in the sub-procedure tab is selected “read using characteristic UUID”. The “characteristic UUID” value is set as “F1:FF” because here we are trying to read from the first characteristic profile i.e. profile 1 0xF1FF. The same procedure is followed to read the profile 2 value by entering “F2: FF” as defined in the <simplegatt.c> and <simplegatt.h> file. The value shown here will be 01 when the read button is clicked and a “success” will be displayed in the status box. When the read button is clicked there are few packets that are sent over the air by the slave to the master. Here, read is basically to read data from profiles which are designed for sensor output.



```
Dump(Rx):
04 FF 06 09 05 1A 00 00 00

------------------------------------------------------------
[37] : <Tx> - 02:29:32.035
-Type                : 0x01 (Command)
-Opcode              : 0xFD12 (ATT_WriteReq)
-Data Length         : 0x07 (7) byte(s)
 ConnHande           : 0x0000 (0)
 Signature           : 0x00 (No)
 Command             : 0x00 (No)
 Handle              : 0x0022 (34)
 Value               : 07
Dump(Tx):
01 12 FD 07 00 00 00 00 22 00 07

------------------------------------------------------------
[38] : <Rx> - 02:29:32.112
-Type                : 0x04 (Event)
-EventCode           : 0xFF (HCI_LE_ExtEvent)
-Data Length         : 0x06 (6) bytes(s)
 Event               : 0x067F (GAP_HCI_ExtentionCommandStatus)
 Status              : 0x00 (Success)
 OpCode              : 0xFD12 (ATT_WriteReq)
 DataLength          : 0x00 (0)
Dump(Rx):
04 FF 06 7F 06 00 12 FD 00

------------------------------------------------------------
[39] : <Rx> - 02:29:32.274
-Type                : 0x04 (Event)
-EventCode           : 0xFF (HCI_LE_ExtEvent)
-Data Length         : 0x06 (6) bytes(s)
 Event               : 0x0513 (ATT_WriteRsp)
 Status              : 0x00 (Success)
 ConnHande           : 0x0000 (0)
 PduLen              : 0x00 (0)
Dump(Rx):
04 FF 06 13 05 00 00 00 00

------------------------------------------------------------
```

Characteristic Read

Sub-Procedure: Read Using Characteristic UUID
Connection Handle: 0x0000
Characteristic Value Handle: 0x0022
Start Handle: 0x0001
Characteristic UUID: F1:FF
End Handle: 0xFFFF
Value: ( ) ASCII ( ) Decimal [✓ Hex]
01
Status: Success

Characteristic Write

Characteristic Value Handle: 0x0022
Connection Handle: 0x0000
Value: ( ) ASCII ( ) Decimal [✓ Hex]
7
Status: Success

Figure 3.5: BTOOL, write a characteristic by UUID [43]

After reading the sensor data the next procedure is to write sensor data as shown in (Figure 3.5). Writing is designed to send notifications or values to the defined profiles in the slave. Here, profile 4 and 6 are notifications enabled. The characteristic value handle will be either 0x0022 or the content in which the value tab displays. The 01, 02 are the notifications which are defined again in <simplegatt.h> file and they are entered in the value box of write section. After clicking the write button “success” shows up to read the real time value in the terminal [44].

When the Bluetooth connection is established after referring to the GATT table, the discover UUID is used to setup a connection between the receiver device to read appropriate sensor values using characteristic profiles. The characteristic profiles play an important role in the design. There are 6 characteristic profiles designed and one for each sensor. (Figure 3.6) shows the profiles implemented.

After implementing the code and before executing the output in the IAR terminal, the next step is to execute the output using Bluetooth in Btool and the Android app. The <simplegatt.c> and <simplegatt.h> files are added and edited accordingly. In <simpleGATTProfile.h> we have the simplechar1-6 profiles added. The characteristic profiles are designed in <simpleGATTProfile.c>code as shown below in (Figure 3.6). The output of Btool and Android working is demonstrated the subsequent chapter. The final GATT table is shown below.

Type (hex)	Type (#DEFINE)	Hex/Text Value(default)	R/W	Notes
0x28 00	GATT_PRIMARY_SERVICE_UUID	0x180A	READ	Start device information service
0x28 03	GATT_CHARACTER_UUID	02 (read permission) 24 00 (handle 0x0024) 51 2A (UUID 0x2A51)	READ	My name String characteristic
0x2A 51	DEVINFO_MY_NAME_UUID	“GIRI BABU :D”	READ	Value
0x28 00	GATT_PRIMARY_SERVICE_UUID	0XFFF0 (SIMPLEPROFILE_SERV_UUID)	READ	Start Simple GATT profile service
0X28 03	GATT_CHARACTER_UUID	0A (notify only) 22 00 (handle 0x0037) F1 FF (UUID 0xFFF1)	READ	Characteristic 1 declaration
0xFF F1	SIMPLEPROFILE_CHAR1_UUID	1 (1bytes)	GATT_PERMIT_READ GATT_PERMIT_WRITE	Value 1
		02(read only)		

0X28 03	GATT_CHARACTER_UUID	25 00 (handle 0x0025) F2 FF (UUID 0xFFF2)	GATT_PERMIT_READ	Characteristic 2 declaration
0xFF F2	SIMPLEPROFILE_CHAR 2_UUID	2 (1bytes)	GATT_PERMIT_READ	Value 2
0X28 03	GATT_CHARACTER_UUID	08 (write only) 28 00 (handle 0x0028) F3 FF (UUID 0xFFF3)	READ	Characteristic 3 declaration
0xFF F3	SIMPLEPROFILE_CHAR 3_UUID	3 (1 bytes)	GATT_PERMIT_READ	Value 3
0X28 03	GATT_CHARACTER_UUID	10(notify only) 2B 00 (handle 0x002B) F4 FF (UUID 0xFFF4)	READ	Characteristic 4 declaration
0xFF F4	SIMPLEPROFILE_CHAR 4_UUID	4(1 bytes)	none	Value 4
0X28 03	GATT_CHARACTER_UUID	02 (write only) 2F 00 (handle 0x002F) F5 FF (UUID 0xFFF5)	GATT_PERMIT_READ	Characteristic 5 declaration
0xFF F5	SIMPLEPROFILE_CHAR 5_UUID	01:02:03:04:05 (5 bytes)	GATT_PERMIT_READ	Value 5
0X28 03	GATT_CHARACTER_UUID	10 (notify only) 37 00 (handle 0x0037) F6 FF (UUID 0xFFF6)	READ	Characteristic 6 declaration
0xFF F6	SIMPLEPROFILE_CHAR 6_UUID	01:02 (2bytes)	(none)	Value 6
0x29 02	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2bytes)	READ | WRITE	Configuration
0x29 01	GATT_CHAR_USER_DESC_UUID	"Characteristic 6" (17bytes)	READ	User description

Figure 3.6: GATT profile table

All the profiles characteristics are designed accordingly so that every sensor data we have the characteristic profile designed to give respective sensor output.

1) SimpleProfileCharacteristic 1: Battery output

2) SimpleProfileCharacteristic 2: Temperature

3) SimpleProfileCharacteristic 3

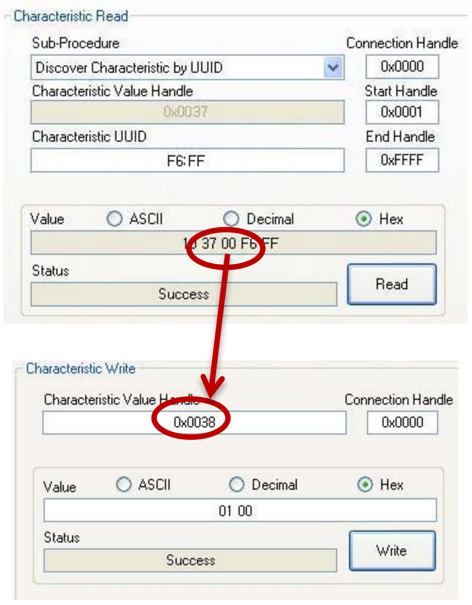
4) SimpleProfileCharacteristic 4: BPM

5) SimpleProfileCharacteristic 5

6) SimpleProfileCharacteristic 6: Raw heart rate data

To test all characteristic profiles, Btool is connected to the CC2541. The discover characteristic UUID is selected in sub-procedure and in UUID row, profile 1 is entered as F1:FF and the read button allows one to read the vale [43]. The data from the battery output of the evaluation board is displayed. The profile 4(UUID=F4:FF) is the temperature data which is received on request so a notification must be set. The profile 6 (UUID=F6:FF) is used to read the heart rate data from the sensor, also by writing the notification.

The notification concept is explained below. When the attribute displays the notification, for instance, <SIMPLEPROFILE_CHAR4_UUID> and <SIMPLEPROFILE_CHAR6_UUID>, the handle address should be added +1. In other words, if we get the handle value as 00 37, then the corresponding notification address is 00 38. When 01 00 is added to write we get the value of the sensor in the window as shown in (Figure 3.7). The values are in hex so they must be converted to decimal. The value is read continuously.



Characteristic Read

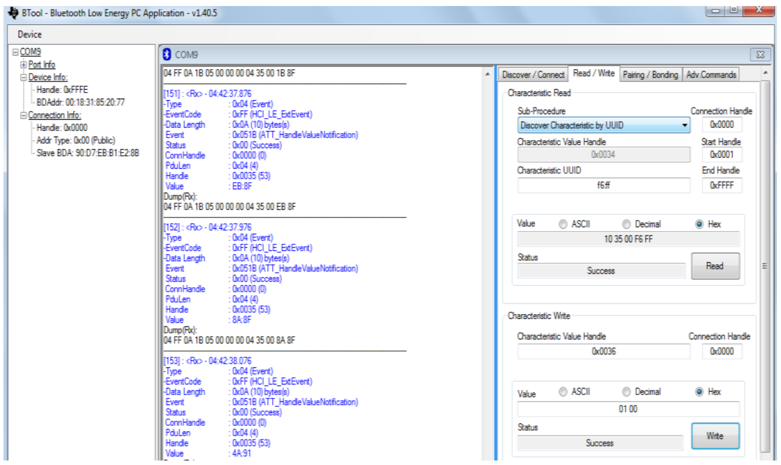
Sub-Procedure: Discover Characteristic by UUID
Connection Handle: 0x0000
Characteristic Value Handle: 0x0037
Start Handle: 0x0001
Characteristic UUID: F6:FF
End Handle: 0xFFFF
Value: ASCII Decimal [✓ Hex]
10 37 00 F6 FF
Status: Success

Characteristic Write

Characteristic Value Handle: 0x0038
Connection Handle: 0x0000
Value: ASCII Decimal [✓ Hex]
01 00
Status: Success

Figure 3.7: Btool write table

The profile 6 (UUID=F6: FF) is used to read the heart rate data from the sensor. The data read is selected which gives the handle value. When the read status button is pressed, we get 00 35 and so value handle entered should be 0x0036 and the notification is 01 00. After writing the notification the output of heart rate is read, consider the output terminal below. For the rest of the profiles from 1-5 for other sensors the output is in general which is read and write, the output will be demonstrated in next chapter along with their respective profile.



Handle: 0xFFFE
BDAddr: 00:18:31:85:20:77
Handle: 0x0000
Addr Type: 0x00 (Public)
Slave BDA: 90:D7:EB:B1:E2:8B

```
04 FF 0A 1B 05 00 00 00 04 35 00 1B 8F
------------------------------------------------------------
[151] : <Rx> - 04:42:37.876
-Type           : 0x04 (Event)
-EventCode      : 0xFF (HCI_LE_ExtEvent)
-Data Length    : 0x0A (10) bytes(s)
 Event          : 0x051B (ATT_HandleValueNotification)
 Status         : 0x00 (Success)
 ConnHandle     : 0x0000 (0)
 PduLen         : 0x04 (4)
 Handle         : 0x0035 (53)
 Value          : EB:8F
Dump(Rx):
04 FF 0A 1B 05 00 00 00 04 35 00 EB 8F
------------------------------------------------------------
[152] : <Rx> - 04:42:37.976
-Type           : 0x04 (Event)
-EventCode      : 0xFF (HCI_LE_ExtEvent)
-Data Length    : 0x0A (10) bytes(s)
 Event          : 0x051B (ATT_HandleValueNotification)
 Status         : 0x00 (Success)
 ConnHandle     : 0x0000 (0)
 PduLen         : 0x04 (4)
 Handle         : 0x0035 (53)
 Value          : 8A:8F
Dump(Rx):
04 FF 0A 1B 05 00 00 00 04 35 00 8A 8F
------------------------------------------------------------
[153] : <Rx> - 04:42:38.076
-Type           : 0x04 (Event)
-EventCode      : 0xFF (HCI_LE_ExtEvent)
-Data Length    : 0x0A (10) bytes(s)
 Event          : 0x051B (ATT_HandleValueNotification)
 Status         : 0x00 (Success)
 ConnHandle     : 0x0000 (0)
 PduLen         : 0x04 (4)
 Handle         : 0x0035 (53)
 Value          : 4A:91
```

Characteristic Read

Sub-Procedure: Discover Characteristic by UUID
Connection Handle: 0x0000
Characteristic Value Handle: 0x0034
Start Handle: 0x0001
Characteristic UUID: f6:ff
End Handle: 0xFFFF
Value: ASCII Decimal [✓ Hex]
10 35 00 F6 FF
Status: Success

Characteristic Write

Characteristic Value Handle: 0x0036
Connection Handle: 0x0000
Value: ASCII Decimal [✓ Hex]
01 00
Status: Success

Figure 3.8: Btool Output terminal using characteristic profile 6

3.2.3 BLE Device Monitor

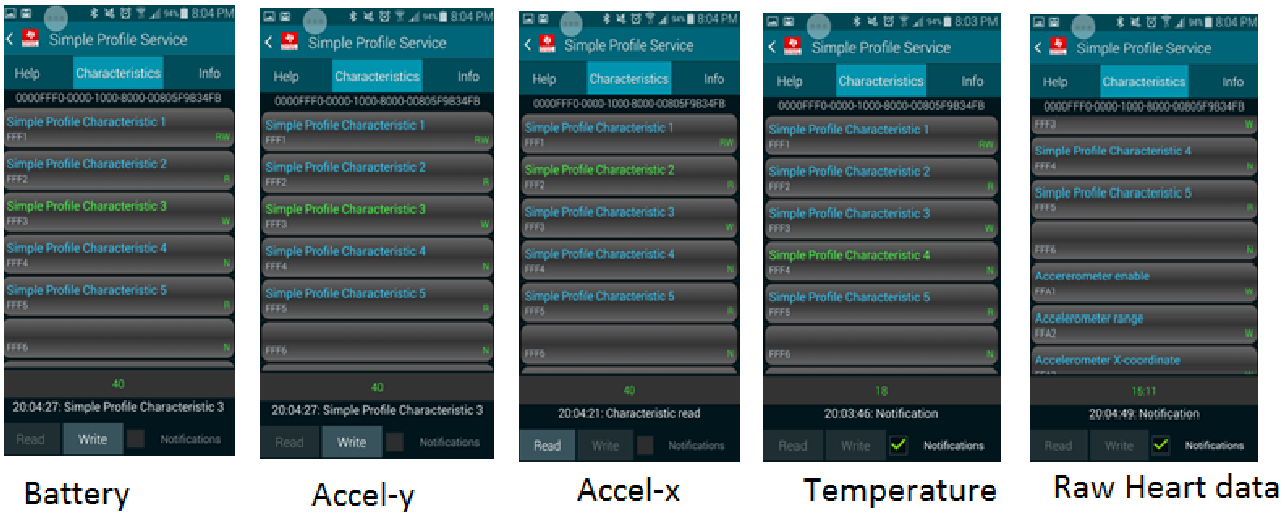
The sensor data is also read in an Android device using the BLE Device Monitor (Texas Instruments), a free Android software which can be downloaded to an Android phone. The protocol is similar to that used by Btool. The app connects to the CC2540 module, and then scans the characteristics in the profile. The user can then select certain characteristics to monitor. In our case, we use the <simpleBLEPeripheral> profile, with the GATT characteristics described above, and read the sensor values using the app.

The BLE device monitor is a precursor to a custom application which will be developed in future work.

CHAPTER 4: EXPERIMENTAL RESULTS

4.1 Verifying Bluetooth Streaming using BLE Device Monitor

The first set of experimental results is to verify the ability to send sensor data to a smartphone via the BLE <simple_GATT> profile. The Android output as shown in below figures shows various profiles along with sensor data. After opening the BLE device monitor app, the profiles tab is opened, and the <simpleBLEPeripheral> is selected. The values from characteristics 1, 2, 3, 4, and 6 are shown in (Figure 4.1). Each corresponds to a specific sensor as describe earlier.



```
0000FFF0-0000-1000-8000-00805F9B34FB
Simple Profile Characteristic 1   FFF1   RW
Simple Profile Characteristic 2   FFF2   R
Simple Profile Characteristic 3   FFF3   W
Simple Profile Characteristic 4   FFF4   N
Simple Profile Characteristic 5   FFF5   R
FFF6   N
40
20:04:27: Simple Profile Characteristic 3
```

```
0000FFF0-0000-1000-8000-00805F9B34FB
Simple Profile Characteristic 1   FFF1   RW
Simple Profile Characteristic 2   FFF2   R
Simple Profile Characteristic 3   FFF3   W
Simple Profile Characteristic 4   FFF4   N
Simple Profile Characteristic 5   FFF5   R
FFF6   N
40
20:04:27: Simple Profile Characteristic 3
```

```
0000FFF0-0000-1000-8000-00805F9B34FB
Simple Profile Characteristic 1   FFF1   RW
Simple Profile Characteristic 2   FFF2   R
Simple Profile Characteristic 3   FFF3   W
Simple Profile Characteristic 4   FFF4   N
Simple Profile Characteristic 5   FFF5   R
FFF6   N
40
20:04:21: Characteristic read
```

```
0000FFF0-0000-1000-8000-00805F9B34FB
Simple Profile Characteristic 1   FFF1   RW
Simple Profile Characteristic 2   FFF2   R
Simple Profile Characteristic 3   FFF3   W
Simple Profile Characteristic 4   FFF4   N
Simple Profile Characteristic 5   FFF5   R
FFF6   N
18
20:03:46: Notification
```

```
0000FFF0-0000-1000-8000-00805F9B34FB
FFF3   W
Simple Profile Characteristic 4   FFF4   N
Simple Profile Characteristic 5   FFF5   R
FFF6   N
Accererometer enable   FFA1   W
Accelerometer range   FFA2   W
Accelerometer X-coordinate
15:11
20:04:49: Notification
```

Figure 4.1: The figures represent battery, temperature, raw heart data

4.2 Temperature Output

To test the ability of the temperature sensor to monitor skin temperature, we plotted the variation of temperature with time after placing the finger on the sensor. The temperature rises to 30 Celsius. The acquisition rate of the sensor is intentionally kept low (1 Hz) to save power. Skin temperature monitoring does not require high temporal resolution.

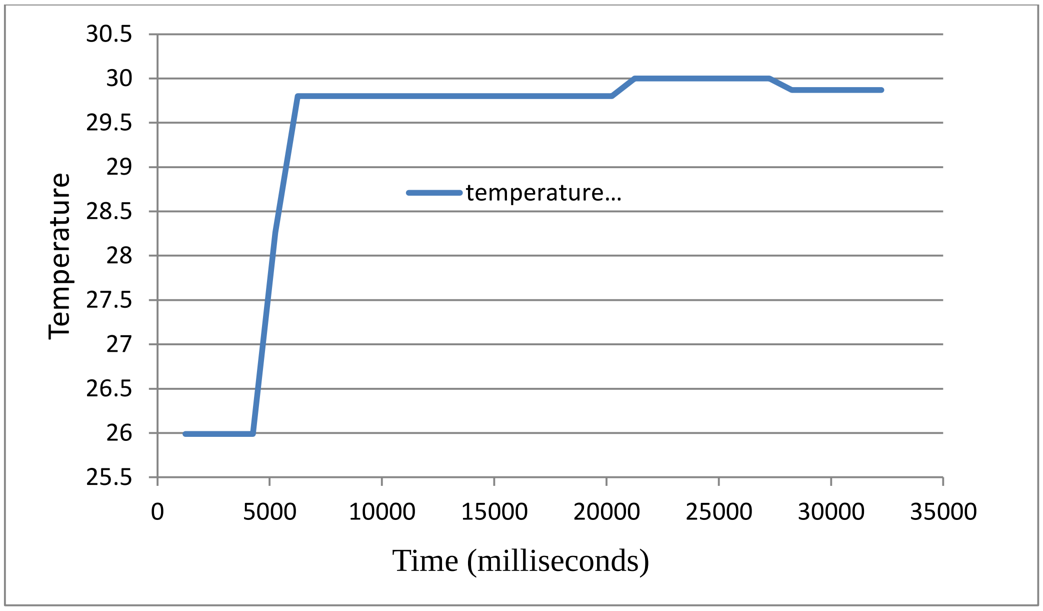


Figure 4.2: Skin temperature vs Time

4.3 Battery Output

To simulate a draining battery, the power supply voltage is changed from 3.3 to 4.5. (Figure 4.3) shows battery % vs supply voltage, demonstrating the ability to monitor battery levels.

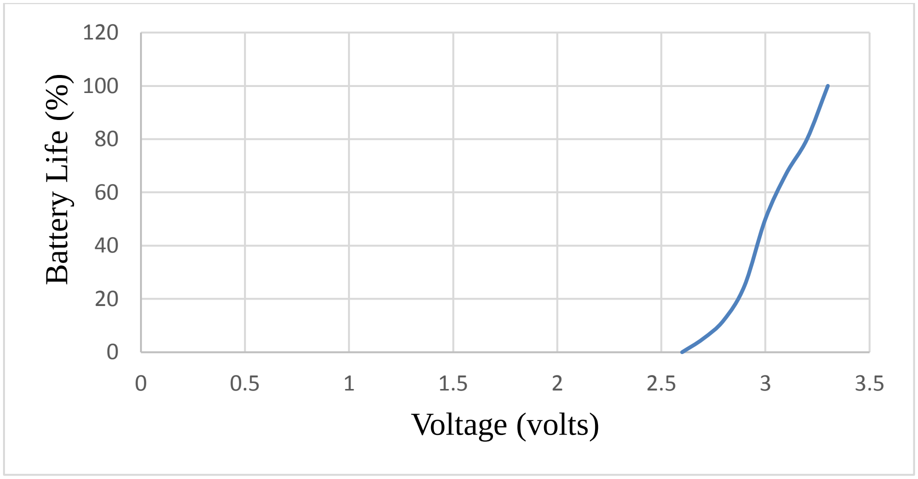


Figure 4.3: Battery Discharging (Percentage)

4.4 Heart Rate Output

4.4.1 Effect of Data Acquisition Rate

The data acquisition rate is tuned to balance power consumption and data integrity. A high data rate allows one to gather more data and provides flexibility in choosing signal processing filters which can improve signal quality. However, a high acquisition rate increases power consumption in the VCNL4000 sensor due to more optical measurements, as well as the CC2540 microcontroller for additional signal processing load. Thus is advantageous to select a minimum acquisition rate to detect and calculate the heart rate. The maximum human heart rate is 220 beats per minute, or 3.6 Hz [24]. The basic Nyquist criterion states that the sampling frequency must be at least twice the bandwidth of the signal, or ~7Hz. However, PPG signals can have higher frequency components in the systolic phase [23]. Thus to reduce the possibility of aliasing [45], a higher sampling rate may be used.

To find the minimum suitable sampling rate, this experiment varied the acquisition rate from 10ms, 30ms, 50ms, and 100ms. The LED current is fixed at 110mA. The data from the below experiments show how the heart rate PPG signals changes with data rate. The 10ms acquisition has the highest time resolution but also shows noise. The noise falls at lower acquisition rate, and the signal is still visible even at 100ms acquisition; however, this comes at the expense of less time precision. The 30ms acquisition speed demonstrates a good balance of reasonable noise, signal amplitude, and time resolution**.**

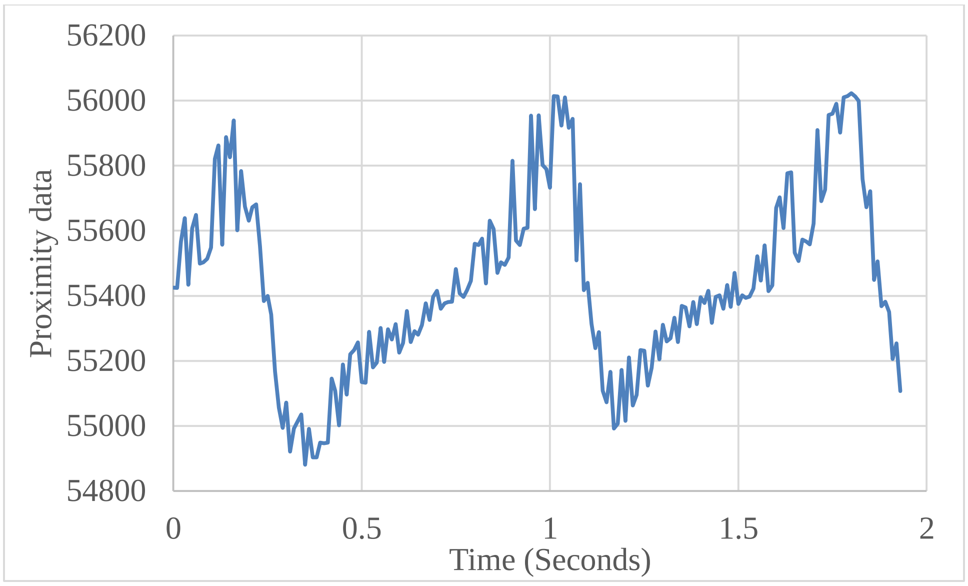


Figure 4.4: PPG signal with acquisition rate of 10 ms

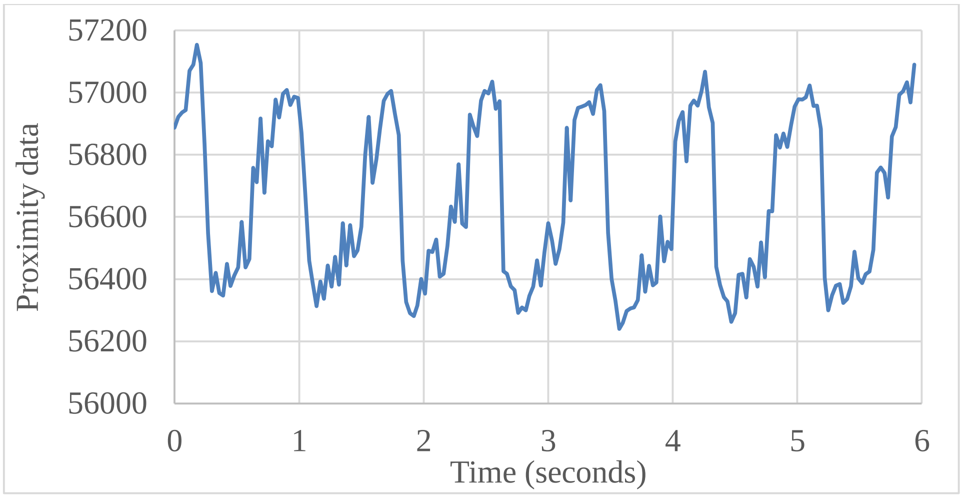


Figure 4.5: PPG signal with acquisition rate of 30 ms

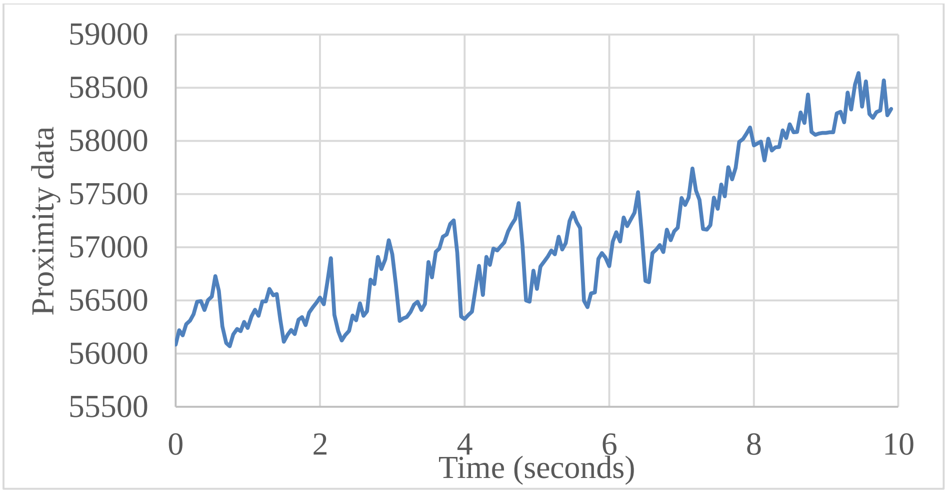


Figure 4.6: PPG signal with acquisition rate of 50 ms

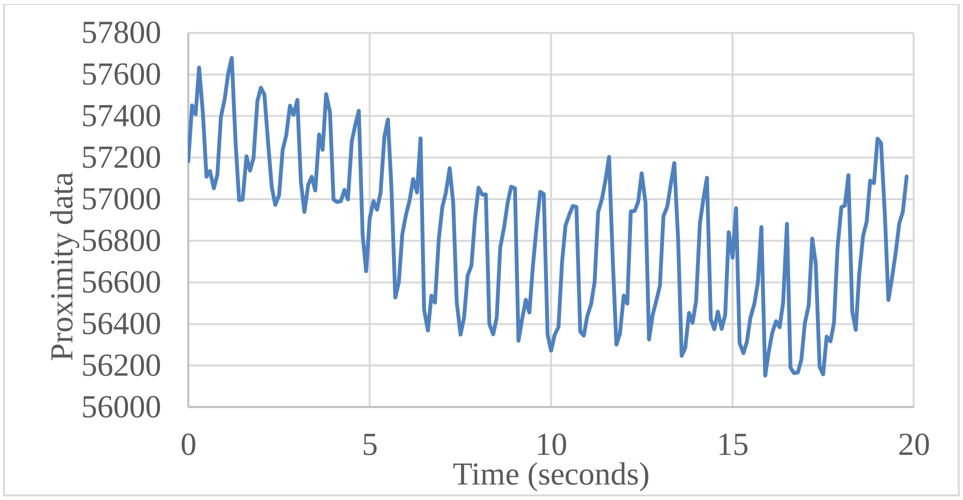


Figure 4.7: PPG signal with acquisition rate of 100 ms

4.4.2 Effect of LED current

Increasing the LED current increases the reflected optical signal, and therefore the signal to noise ratio. This experiment describes changing the LED current to 4 different values 70mA, 90mA, 110mA, 130mA. The acquisition rate is kept constant at 30msec.

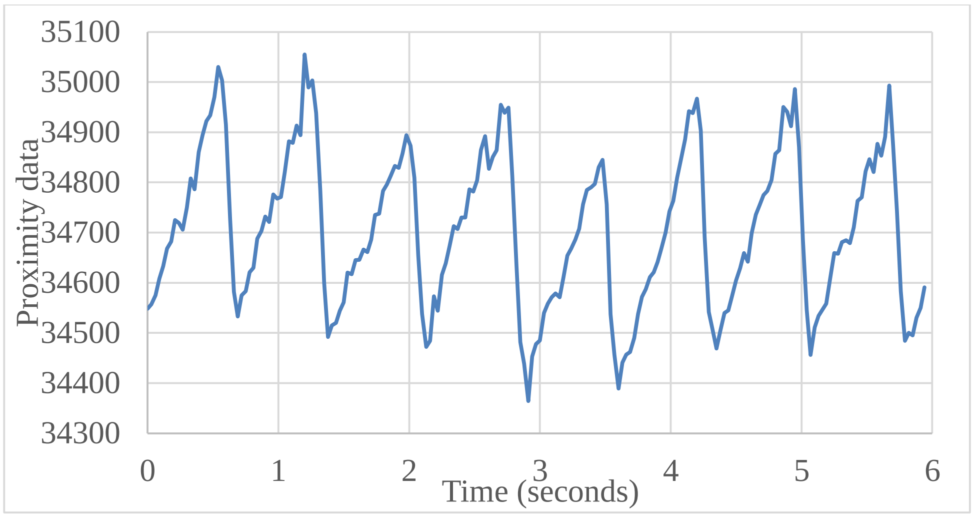


Figure 4.8: PPG signal with LED current of 70ma

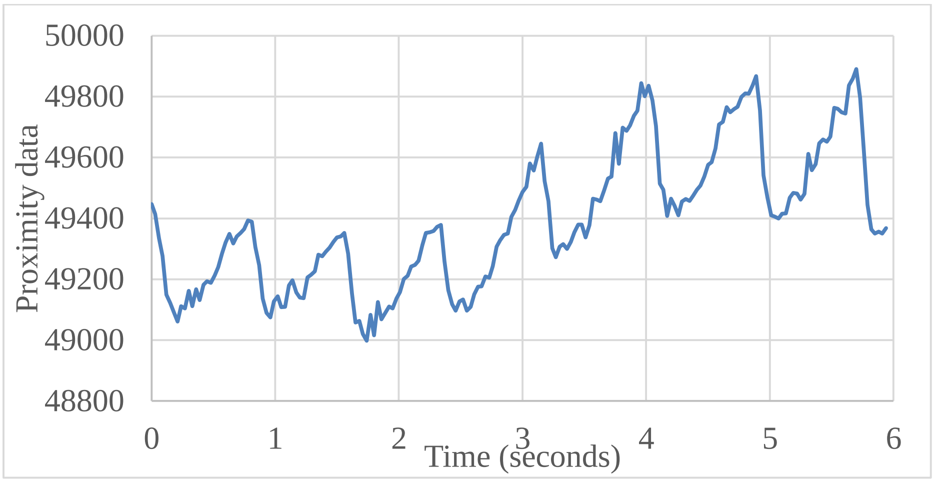


Figure 4.9: PPG signal with LED current of 90ma

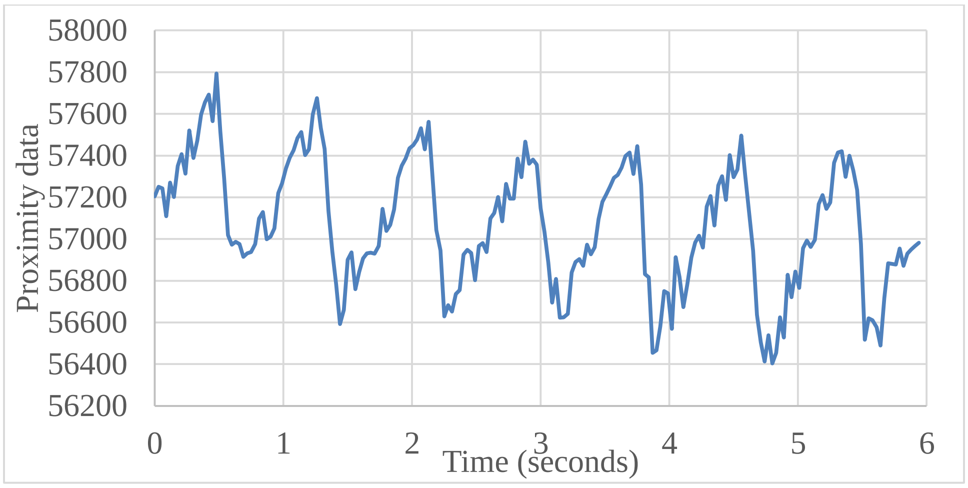


Figure 4.10: PPG signal with LED current of 110ma

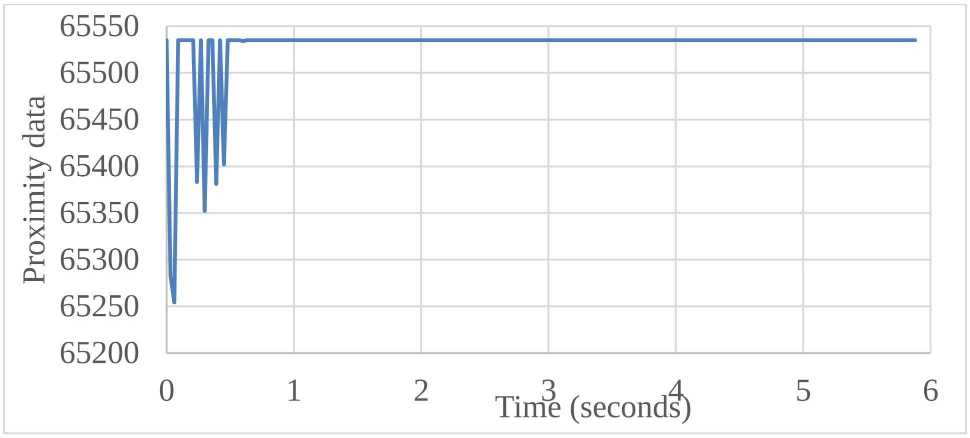


Figure 4.11: Constant Acquisition rate of 30msec and Led current of 130ma

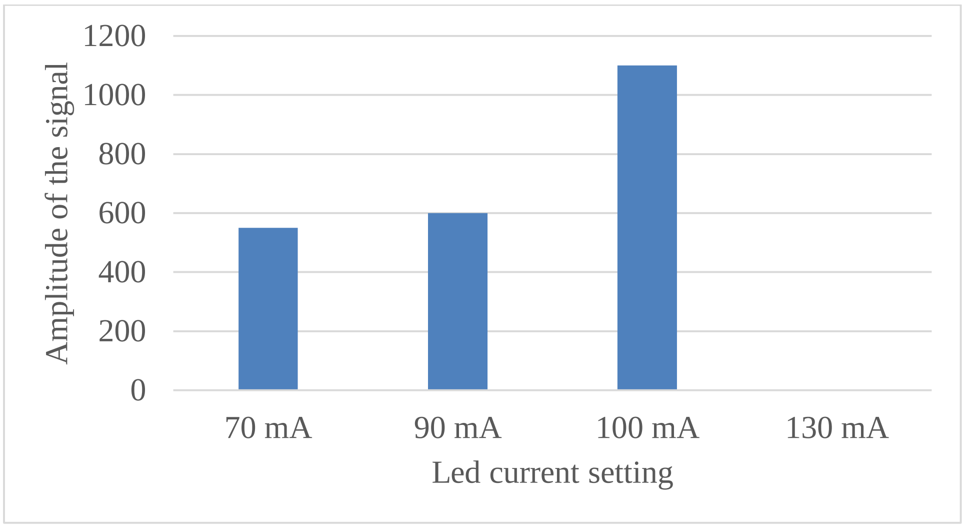


Figure 4.12: Amplitude of the signal when Led current is changed

Comparing the graphs shows that the amplitude of the signal increases when the LED current is increased. Hence, signal to noise ratio improves as shown in Figure 4.12. However, if the current is set too high (ie 130 mA), the signal becomes saturated, and the signal therefore has no amplitude. Higher LED currents also consume more power, which represents a trade-off.

4.4.3 Effect of Moving Average Filter Length

This experiment describes the effect of the different lengths of the moving average filters used. The filter length of 100 is used in this project while designing, and is used in all other experiments. Here, a test condition with a length of 200 is executed. Compared with length of 100 the 200 affects the ability to remove DC offset. A long filter length slows down the ability of the filter to track the DC value and thus more time is needed to remove offset.

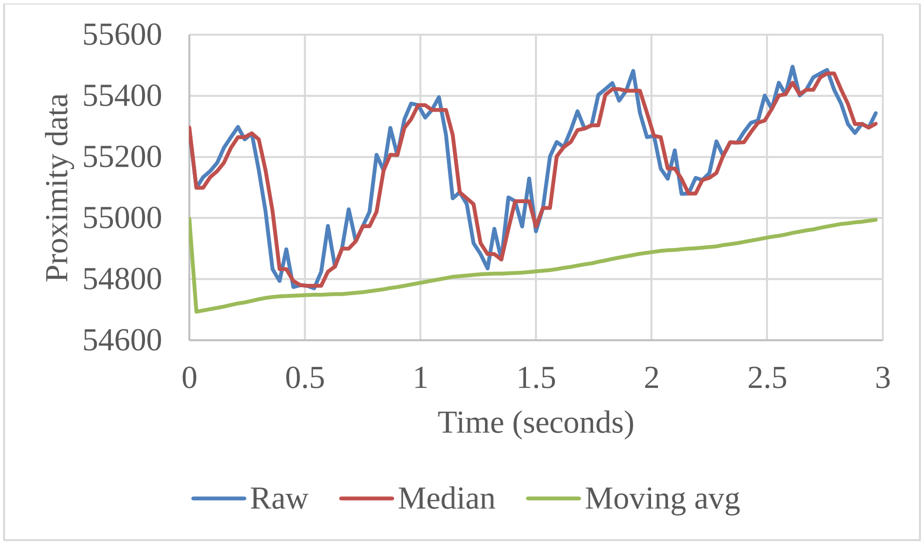


Figure 4.13: Filter length of 200 (raw, median, moving)

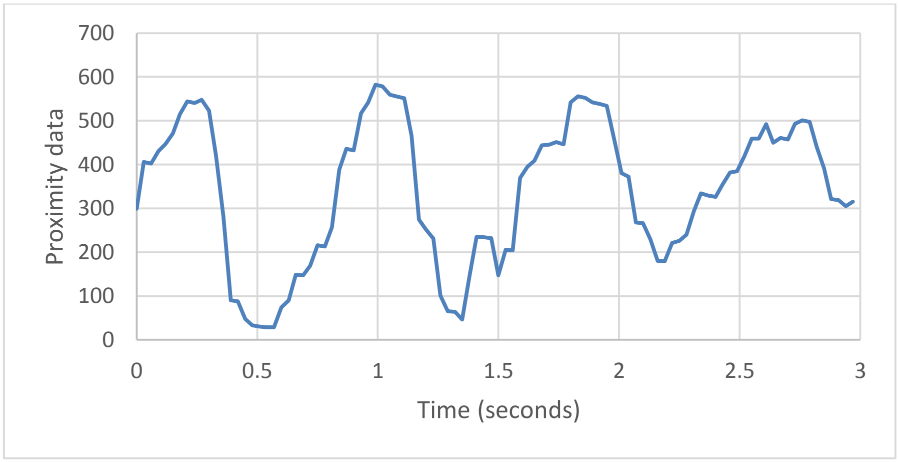


Figure 4.14: Filter length of 200 (High pass filter)

4.4.4 Effect of Filtering and Heart rate variability Monitoring

This test condition demonstrates the effect filtering of median filtering and high pass filtering the raw signal. The 3 point median filter removes spurious noise, and it's effect is shown below.

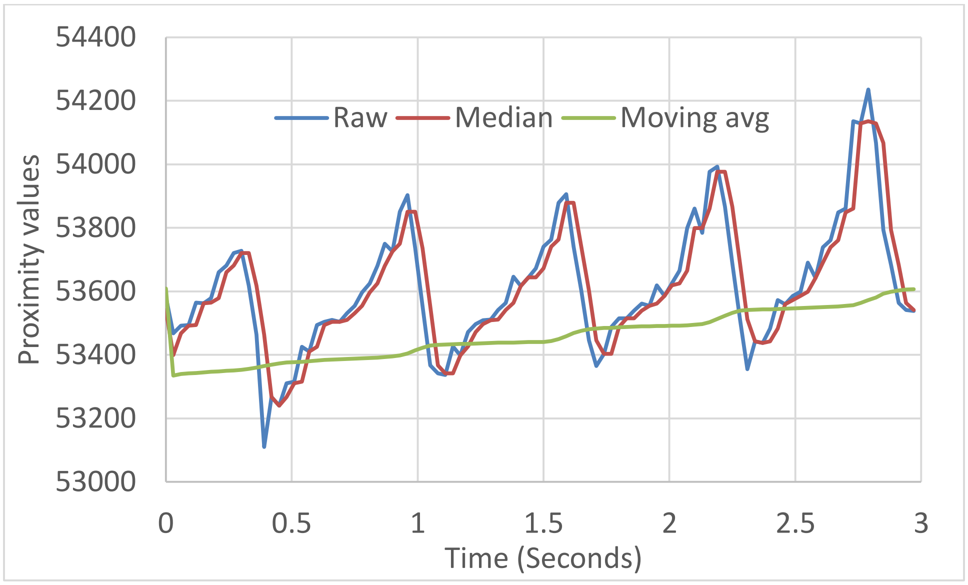


Figure 4.15: Raw, median, moving average filter data led current 110ma and acquisition at 30msec

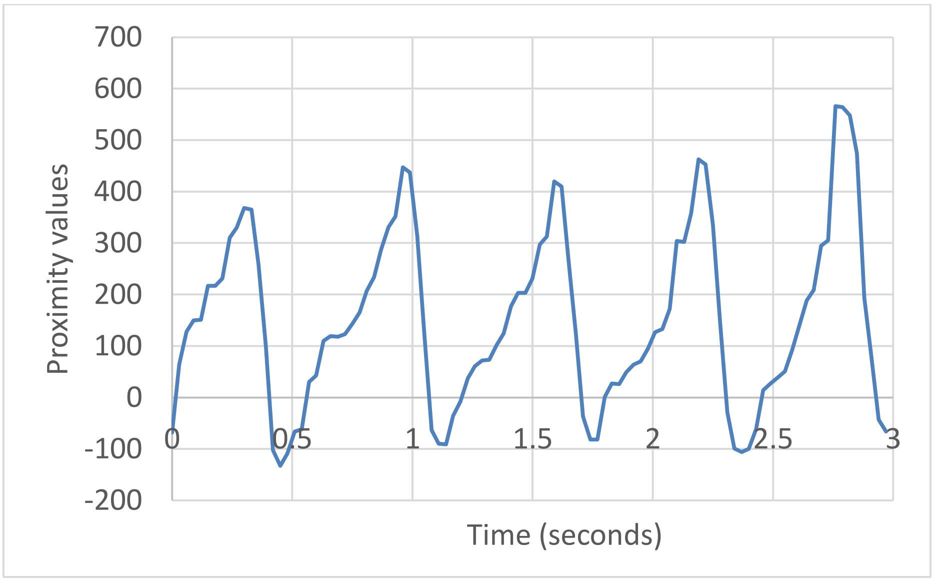


Figure 4.16: High pass filter data with led current of 110ma and acquisition at 30msec

The high pass moving average filter signal is obtained by subtracting the moving average signal from the median filtered signal. The signal here is offset to 0th level. The data shown in Figure 4.16 is not completely centered at zero because the high pass filter was not operated for a sufficiently long time. However, it shows the ability to remove DC offset.

The rr-interval and BPM changes whenever a person stood up and sat down. This experiment was recorded in a video. The collected data recorded in the videos was used to plot the figures. The data is displayed in the LCD of SmartRF05 board.

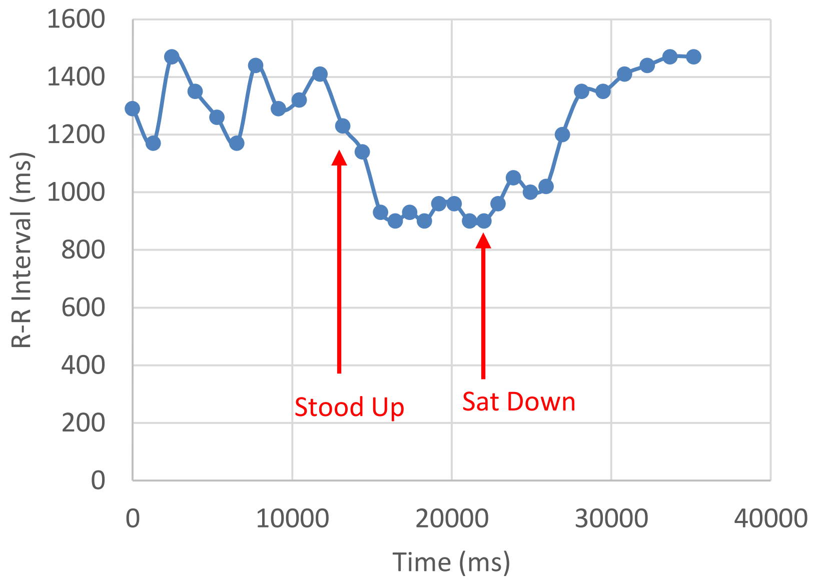


Figure 4.17: Data change in rr-interval

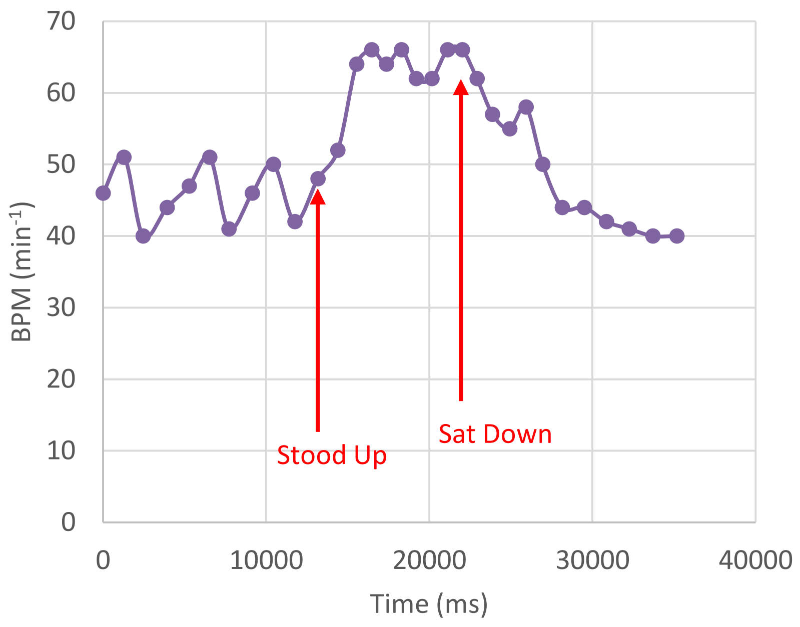
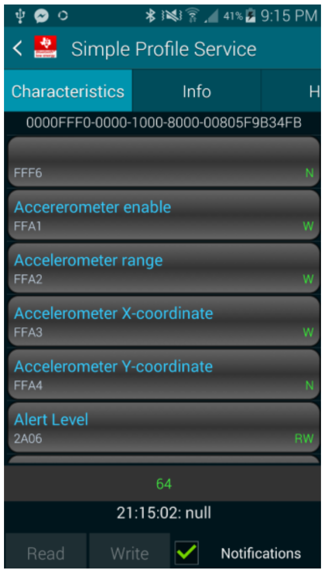


Figure 4.21: Data change in BPM

4.4.5 Beats Per Minute Measurements

The beats per minute (BPM) is calculated by a threshold detector as described earlier. Every peak here is an rr-interval which is calculated between the peaks using the OSAL system clock. This time between the peaks is used to calculate the BPM, using the formulae BPM=60/RR-INTERVAL. (Figure 4.) shows the BPM output sent to an Android application.



9:15 PM

Simple Profile Service

Characteristics | Info | H

0000FFF0-0000-1000-8000-00805F9B34FB

FFF6 N

Accelerometer enable
FFA1 W

Accelerometer range
FFA2 W

Accelerometer X-coordinate
FFA3 W

Accelerometer Y-coordinate
FFA4 N

Alert Level
2A06 RW

64

21:15:02: null

Read | Write | ✓ Notifications

Figure 4.22: BPM calculation shown in Android BLE device monitor

Beats per minute measurements were taken on 3 individuals. The sensor was mounted on the earlobe via an earclip. After placing the sensor the <simpleBLEperipheral> code was executed to read the BPM. The data demonstrates that the firmware can successful calculate the heart rate. Individual differences as well as temporal variation in heart rate can be seen in the experimental data below.

Person 1	Person 2	Person 3
BPM: 62	BPM: 69	BPM: 86
BPM: 63	BPM: 64	BPM: 71
BPM: 63	BPM: 68	BPM: 76
BPM: 65	BPM: 64	BPM: 77
BPM: 62	BPM: 68	BPM: 83
BPM: 62	BPM: 66	BPM: 87
BPM: 64	BPM: 64	BPM: 83
BPM: 64	BPM: 69	BPM: 80
BPM: 58	BPM: 64	BPM: 83
BPM: 60	BPM: 66	BPM: 79
BPM: 62	BPM: 66	BPM: 77

Figure 4.23: Beats per minute data

CHAPTER 5: CONCLUSION

The project objectives described earlier in the thesis has been successfully demonstrated. We found that the heart rate sensor, together with on-board signal processing, did indeed provide high precision in fingertip and earlobe measurements, as we had anticipated. The 3 point median filter was excellent in removing spurious noise without significant computational cost. The high pass moving average filter was able to remove the DC component, also with little cost when implemented recursively; however, it did add modest memory requirements. The current firmware allows integration of heart rate, skin temperature. Future work will consider several items including acceleration data, triggered heart rate detection, file transfers, and implementing the Bluetooth heart rate profile for compatibility with a wide range of smartphone apps.

REFERENCES

[1] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, “A review of wearable sensors and systems with application in rehabilitation,” *J. NeuroEngineering Rehabil.*, vol. 9, no. 1, p. 21, 2012.

[2] S. P. Gulley, E. K. Rasch, and L. Chan, “If We Build It, Who Will Come?: Working-Age Adults With Chronic Health Care Needs and the Medical Home,” *Med. Care*, vol. 49, no. 2, pp. 149–155, Feb. 2011.

[3] G. O. Rosenwasser and J. S. Tiedeman, “A stable slit lamp mounting device for 90 D lens use in non-contact ophthalmoscopy,” *Ophthalmic Surg.*, vol. 17, no. 8, p. 525, Aug. 1986.

[4] U. E. Bauer, P. A. Briss, R. A. Goodman, and B. A. Bowman, “Prevention of chronic disease in the 21st century: elimination of the leading preventable causes of premature death and disability in the USA,” *The Lancet*, vol. 384, no. 9937, pp. 45–52, Jul. 2014.

[5] “http://www.cdc.gov/nchs/fastats/deaths.htm.” .

[6] A. S. Go, D. Mozaffarian, V. L. Roger, E. J. Benjamin, J. D. Berry, M. J. Blaha, S. Dai, E. S. Ford, C. S. Fox, S. Franco, H. J. Fullerton, C. Gillespie, S. M. Hailpern, J. A. Heit, V. J. Howard, M. D. Huffman, S. E. Judd, B. M. Kissela, S. J. Kittner, D. T. Lackland, J. H. Lichtman, L. D. Lisabeth, R. H. Mackey, D. J. Magid, G. M. Marcus, A. Marelli, D. B. Matchar, D. K. McGuire, E. R. Mohler, C. S. Moy, M. E. Mussolino, R. W. Neumar, G. Nichol, D. K. Pandey, N. P. Paynter, M. J. Reeves, P. D. Sorlie, J. Stein, A. Towfighi, T. N. Turan, S. S. Virani, N. D. Wong, D. Woo, M. B. Turner, and on behalf of the American Heart Association Statistics Committee and Stroke Statistics Subcommittee, “Heart Disease and Stroke Statistics--2014 Update: A Report From the American Heart Association,” *Circulation*, vol. 129, no. 3, pp. e28–e292, Jan. 2014.

[7] “National Heart,lung and blood institute.” .

[8] K. G. Reeuwijk, S. J. W. Robroek, L. Hakkaart, and A. Burdorf, “How Work Impairments and Reduced Work Ability are Associated with Health Care Use in Workers with Musculoskeletal Disorders, Cardiovascular Disorders or Mental Disorders,” *J. Occup. Rehabil.*, vol. 24, no. 4, pp. 631–639, Dec. 2014.

[9] A. Ciccone, M. G. Celani, C. Rossi, V. Villa, and E. Righetti, “Continuous physiological monitoring for acute stroke,” in *Cochrane Database of Systematic Reviews*, The Cochrane Collaboration, Ed. Chichester, UK: John Wiley & Sons, Ltd, 2010.

[10] G. G. Berntson, J. Thomas Bigger, D. L. Eckberg, P. Grossman, P. G. Kaufmann, M. Malik, H. N. Nagaraja, S. W. Porges, J. P. Saul, P. H. Stone, and M. W. Der Molen, “Heart rate variability: Origins, methods, and interpretive caveats,” *Psychophysiology*, vol. 34, no. 6, pp. 623–648, Nov. 1997.

[11] E. Lim, H.-K. Lee, H.-S. Myoung, and K.-J. Lee, “Development of a noncontact heart rate monitoring system for sedentary behavior based on an accelerometer attached to a chair,” *Physiol. Meas.*, vol. 36, no. 3, pp. N61–N70, Mar. 2015.

[12] V. P. Andreev, T. Head, N. Johnson, S. K. Deo, S. Daunert, and P. J. Goldschmidt-Clermont, “Discrete Event Simulation Model of Sudden Cardiac Death Predicts High Impact of Preventive Interventions,” *Sci. Rep.*, vol. 3, May 2013.

[13] J. M. Khor, A. Tizzard, A. Demosthenous, and R. Bayford, “Wearable sensors for patient-specific boundary shape estimation to improve the forward model for electrical impedance tomography (EIT) of neonatal lung function,” *Physiol. Meas.*, vol. 35, no. 6, pp. 1149–1161, Jun. 2014.

[14] S. Stančin and S. Tomažič, “Early Improper Motion Detection in Golf Swings Using

Wearable Motion Sensors: The First Approach," *Sensors*, vol. 13, no. 6, pp. 7505–7521, Jun. 2013.

[15] K. Malzahn, J. R. Windmiller, G. Valdés-Ramírez, M. J. Schöning, and J. Wang, "Wearable electrochemical sensors for in situ analysis in marine environments," *The Analyst*, vol. 136, no. 14, p. 2912, 2011.

[16] Y. Hao and R. Foster, "Wireless body sensor networks for health-monitoring applications," *Physiol. Meas.*, vol. 29, no. 11, pp. R27–R56, Nov. 2008.

[17] P. S. Pandian, K. Mohanavelu, K. P. Safeer, T. M. Kotresh, D. T. Shakunthala, P. Gopal, and V. C. Padaki, "Smart Vest: wearable multi-parameter remote physiological monitoring system," *Med. Eng. Phys.*, vol. 30, no. 4, pp. 466–477, May 2008.

[18] "Heart disease Health center," *WEBMED*. [Online]. Available: "http://www.webmd.com/heart-disease/guide/what-causes-heart-palpitations." .

[19] "Heart disease," *WEBMED*. .

[20] Michael A chen, "National instititute of health," *service of the U.S. National Library of Medicine*. [Online]. Available: "http://www.nlm.nih.gov/medlineplus/ency/article/003877.htm." .

[21] university of rochester medical center, "Arrhytmias," *Health encyclopedia*. [Online]. Available: http://www.urmc.rochester.edu/encyclopedia/content.aspx?ContentTypeID=85&ContentID=P00195.

[22] Web design cochin, "Circuits today," *Heart rate monitoring using 8051*. [Online]. Available: "http://www.circuitstoday.com/heart-rate-monitor-using-8051."

[23] ISA KANYAKUMARI, "ISA KANYAKUMARI," *Anasthesia news*. [Online].

Available: “http://www.isakanyakumari.com/february-1-2013.php.” .

[24] Brain mac sports coach, “Heart rate training zones.” [Online]. Available: “http://www.brianmac.co.uk/hrm1.htm.” .

[25] D. Chandrasekar, B. Arnetz, P. Levy, and A. S. Basu, “Plug-and-play, single-chip photoplethysmography,” 2012, pp. 3243–3246.

[26] A. S. Basu, “Sensor and method for continuous health monitoring.”

[27] Ziff davis, “Basis peak,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2471564,00.asp.

[28] Ziff davis, “Garmin Forerunner 15,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2458665,00.asp.

[29] Ziff davis, “Mio Fuse,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2473681,00.asp.

[30] Ziff davis, “FitBit Charge,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2473164,00.asp.

[31] Ziff davis, “Garmin vivosmart,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2470661,00.asp.

[32] Ziff davis, “Jawbone UP24,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2455390,00.asp.

[33] Ziff davis, “Microsoft band,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2471962,00.asp.

[34] Ziff davis, “Jabra,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2472050,00.asp.

[35] Ziff davis, “Tom Tom,” *PC MAG*. [Online]. Available:

http://www.pcmag.com/article2/0,2817,2428355,00.asp.

[36] Ziff davis, “Bragi earbud,” *PC MAG*. [Online]. Available: http://www.pcmag.com/article2/0,2817,2414017,00.asp.

[37] VISHAY SEMICONDUCTORS, “Designing VCNL4000 into an application.” [Online]. Available: http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CCkQFjAC&url=http%3A%2F%2Fwww.adafruit.com%2Fdatasheets%2Fvcnl4000AN.pdf&ei=wLkdVaeSLMLisAXyioDYBg&usg=AFQjCNFR1U4guzduxzOCePsxzpR9Oo4z6A&sig2=n06yjFXwlaLHn32m04oVnA.

[38]VISHAY SEMICONDUCTORS. "VCNL4000 DATAHSEET". [Online]. Avilable: “http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CCkQFjAC&url=http%3A%2F%2Fwww.adafruit.com%2Fdatasheets%2Fvcnl4000AN.pdf&ei=wLkdVaeSLMLisAXyioDYBg&usg=An06yjFXwlaLHn32m04oVnA.” .

[39] William H. Righter, 540 SW N191st Aloha oreg John J. Filla 7680 SW 182 PL, Aloha, “Wrist worn heart rate monitors,” 4938228, 03-Jul-1990.

[40] TEXAS INSTRUMENTS, “TMP 112 TEMPERATURE SENSOR.” [Online]. Available: http://www.ti.com.cn/general/cn/docs/lit/getliterature.tsp?baseLiteratureNumber=SBOS473&fileType=pdf.

[41]TEXAS INSTRUMENTS, "TMP 112 TEMPERATURE SENSOR USER GUIDE." [Online]. Avilable:“http://www.ti.com.cn/general/cn/docs/lit/getliterature.tsp?baseLiteratureNumber3&fileType=pdf.” .

[42]"Bluetooth Low energy." [Online].

Avilable:“http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CC4QFjAA&url=http%3A%2F%2Fakizukidenshi.com%2Fdownload%2Fds%2Fmxic%2FMX25L6445E_3V_64Mb_v1.8.pdf&ei=fL4dVfqwC8fksAXqxYC4Bw&usg=AFQjCNGmFXlytGR7CcRfO80QXvVKLYwz_A&sig2=MoTKdr3N3TiNu6rxtusgQQ&bvm=bv.89947451,d.b2w.” .

[43]“Bluetooth.” [Online]. Available: http://en.wikipedia.org/wiki/Bluetooth.

[44]"Bluetooth Low Energy software developers guide". [Online]. Available: “http://www.ti.com.cn/general/cn/docs/lit/getliterature.tsp?baseLiteratureNumber=swru301&fileType=pdf.” .

[45] S. S. Young, *Computerized data acquisition and analysis for the life sciences: a hands-on guide*. Cambridge, UK ; New York, NY: Cambridge University Press, 2001.

ABSTRACT

DEVELOPMENT OF A HUMAN HEART RATE AND SKIN TEMPERATURE MONITORING SYSTEM

by

GIRI BABU SINNAPOLU

MAY 2015

Advisor: Dr. Amar Basu

Major: Electrical Engineering

Degree: Master of Science

Continuous heart rate monitors (CHRM) can give unique insights into a person's physiological and psychological state; however, existing CHRMs are not suitable for continuous monitoring because they do not provide beat-to-beat accuracy and comfort needed for all-day use. Our group is currently developing a miniature earlobe mounted CHRM system which is both comfortable and provides beat-to-beat accuracy, based on a miniature heart rate sensing technology developed previously in our lab. The objective of this thesis is to design low power system firmware to perform key tasks: i) acquire heart rate and skin temperature data from wearable sensors; ii) signal process the raw heart rate data to calculate r-r interval; iii) optimize the heart rate sensor parameters for power and noise; iv) store the sensor data in local flash memory, and v) stream the data to an Android device. Each of the above functions was successfully implemented with low power considerations.

AUTOBIOGRAPHICAL STATEMENT

Giribabu Sinnapolu received his Bachelor of Engineering in Electronics and Communication from KSIT, Bangalore, India in May 2010. He worked as Embedded Engineer for 3 years in automotive and industrial sectors, he worked on various protocols like SPI, UART, CAN, I2C etc. He worked on battery management and monitoring project for Mahindra, controlling moving robots using ZigBee for L&T, designed vehicle tracking system using GPS and GSM modules from U-blox for Indian army. His projects includes various microcontrollers from Freescale, Texas instruments, Nuvoton, Stmicro, and modules from U-BLOX, SENA. His work includes end-to-end product design starting from component selection, hardware schematics, prototype design, and software implementation etc. He has worked primarily on the embedded development and is well experienced with C and C++ programming using workbench like IAR, ECLIPSE, and KEIL etc. His experience at ECOCAR-2 includes front end development using QT creator for automotive infotainment project for GM at Wayne state university. He is now working full time at Ford Motor Company's Research and Innovation Division on connected vehicles project as a system design engineer. He is expected to graduate with a Master of Science in Electrical Engineering in May 2015.