Development of a Health Watch for Monitoring Daily Effects of Stress and Disease

KYLE RYAN LASHBROOK

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Masters of Science in Electrical Engineering

University of Washington

2017

Supervisory Committee:

Robert Bruce Darling, Chair

Jim Peckol

Program Authorized to Offer Degree:

Electrical Engineering

University of Washington

ABSTRACT

Development of a Health Watch for Monitoring Daily Effects of Stress and Disease

Kyle Ryan Lashbrook

Masters of Science in Electrical Engineering

Chair of the Supervisory Committee:

Professor Robert Bruce Darling

Electrical Engineering

Heart rate variability (HRV), the changes in the beat-to-beat fluctuation in the heart rate, has been directly linked to levels of chronic stress in individuals. Monitoring HRV in individuals has been shown to reduce levels of stress and increase overall health. Chronic stress has a dire effect on human health, and it is linked to an increase in cardiovascular disease, as well as diabetes. However, it has been vastly overlooked by society as a problem pertaining to health, and there are currently no solutions to monitor the daily effects of stress. This thesis is aimed at solving this problem through the design and development of a real-time health watch to monitor an individual's heart rate variability, heart rate and respiration rate. The intended use of the device is directed toward medical professionals who find it necessary to monitor a patient's daily stress in hopes to mitigate current and future health problems.

DEDICATION

I'd like to dedicate this thesis to my friends and family, for their encouragement and support over the last five years. You instilled confidence in me along the way and provided help in times of need. I would especially like to thank my brother Kevin. Without your relentless support and inspiration, none of this would have been possible.

Also, I would like to dedicate this thesis to the memories of my late friend and brother Shaun Maurer. I would not be where I am, or who I am today without your friendship and love over the years. You will be forever with me through the rest of my journeys.

ACKNOWLEDGEMENT

Firstly, I would like to thank and acknowledge Professor Bruce Darling for advising me and supporting the ideation and creation of this thesis. His guidance through the process was invaluable to me. Professor Darling supported the idea of the health watch in undergraduate studies, and allowed me to continue its work through graduate school as a thesis. The knowledge and support he lends to his students is unmatched, and I am grateful to have had him as my advisor.

Also, I would like to acknowledge Professor Arabshahi and Professor Peckol. Both professors allowed me to work as their teaching assistant through graduate school. Without these opportunities that they provided it would have been profoundly more difficult to succeed.

Finally, I'd like to acknowledge the department of Electrical Engineering at the University of Washington. My acceptance into graduate school from undergrad made this all possible. The education I've received over the years is irreplaceable. The staff and faculty allows students to grow and explore their passions.

TABLE OF CONTENTS

Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1:

1 INTRODUCTION

1.1 Background

In current civilization stress is a part of daily life. 77 percent of Americans report they feel the physical symptoms caused by stress, and 48 percent say it has a negative impact on their personal and professional life [1]. Stress is linked to the increased risk of heart disease and has been proven to worsen many preexisting diseases such as diabetes, rheumatoid arthritis and peptic ulcer disease [2]. Obesity, which the US Centers for Disease Control and Prevention (CDC) have ranked as the number one health threat in America, with an estimated 400,000 deaths annually [3], has recently been linked to stress [4]. Even with all the statistics and medical studies only 29 percent of adults believe they are doing a very good job at managing or reducing their stress, and the American Psychological Association (APA) warns that this disconnect is cause for concern. Clearly, various studies have shown chronic stress is a driving force behind chronic illness in America, which is not only causing mortality rates to increase, but also the health care costs associated with it. It is critical that the entire health community recognize the role of stress in causing and exacerbating chronic health conditions, and support models of care that help people make positive choices [5].

Not all stress is bad, in fact amounts of good stress, also known by the term eustress, have been shown to enhance and improve cognitive brain function. This is because its duration is short lived, only refreshing the parasympathetic response and not deteriorating it [6]. Bad

stress is continual, it lasts for long durations and puts the adrenal gland in overdrive, secreting levels of cortisol and cortisone that can stay in the blood for hours leading to havoc on the body and mind. Within a person's daily activities, onsets of both good and bad stress may occur, but often they cannot be differentiated between, or their potential for health problems appreciated. Heart rate variability in humans is known to decrease in moments of stress, both good and bad. It's a natural response of the body, controlled by the autonomic nervous system. Because of the connection between variability of the heart and stress, if monitored, heart rate variability can indicate an individual's state of health.

Recent advancements in machine learning driven by the increase in available computational power has brought a new range of opportunities in wearable devices. Big Data is being used to find trend lines and predictors for different classes of information in diseases. The medical field stands to benefit greatly from the advancements in big data technologies for its predictive attributes. The healthcare industry produces between 1.2 to 2.4 exabytes of data every year [7]. The data being recorded by medical devices is an array of different health metrics during patient visits to the doctor's office or hospitals. While this data is being used to track a patient's health over time, it falls short of providing any indication of a person's daily health issues. A big question is how to get data from individuals during their daily lives outside of specialized doctor visits. Today we need to see doctors or physicians to do medical testing to have any indication of a disease present in the body, or to get diagnosed for such things as stress, depression or anxiety. If real time data can be recorded, individuals living with certain types of disease or acute stress may find immense insight into the daily effects of their ailment on the body. Monitoring real time health metrics of the human body has the potential to provide early warning signs of stress and

certain diseases to help aid medical professionals in proper diagnosis, and or alert individuals to health risks. If proper data collection is established, then health care efficiency would increase, and overall human health would benefit.

Many of today's heart rate monitors are developed for fitness tracking and are intended for use by the general public. This has created a large and partially over saturated market of products encompassing a large spectrum of high to low quality. Current fitness trackers utilize 3-axis accelerometers to provide activity count, which can be used to estimate the heart rate of the individual. This takes their focus away from the heart, and simply provides a better step counter. Most users find it appealing to count their daily steps and relate this to a health indicator, but many of these recordings provide false information to individuals who may be suffering from underlying health issues. Recent discoveries have found HRV to be a desired health indicator that may predict the current and future state of an individual's health. This goes beyond step counting, activity counting and calorie counting for the sake of attempting to stay within a certain weight range. HRV gives insight into the autonomous nervous system, which is an important control system of the body. For individuals suffering from disease, anxiety, stress or signs of possible disease onset, HRV tracking has the potential to provide a physician with additional information to help in predicting and reacting to a patient's state of health. If individuals log their HRV data, correlations can be made between new and past data readings to help assist in predicting the probability of a potential disease.

1.2 Need Statement

The biotech device industry is flooded with products aimed at individuals trying to quantify daily activity with a health status, putting the burden of knowledge and diagnosis on the user and not a trained medical expert. These devices do not consider individuals that suffer from chronic stress or disease, who are more concerned with being informed if they are taking the proper daily actions that help minimize their symptoms. A device that may be used by the medical profession to aid in patient doctor knowledge and relationship through tracking and recording important health metrics has the potential to advance the medical field as well as the health of patients.

1.3 Aims and Objectives

This thesis explores the end-to-end design and development of a product that is used to track an individual's daily health status by recording signals such as heart rate, heart rate variability and respiration rate. The device is a wrist watch that captures and records these metrics in real time. The goal of the health watch is to acquire real time HRV and respiration rate (RR) data in patients who struggle with stress and or have a diagnosed disease. The watch aims to be realizable in the manner that the data recorded should have enough confidence that it may be used as a medical tool by physicians and doctors. It should be able to collect and store enough data so that data scientists can produce long term trendlines to gain insight into a patient's health conditions.

The work starts with a literature review to provide context for the work. Then the design and development of the health watch is covered in its entirety. This includes but is not limited to

signal extraction by means of analog and digital design, followed by signal processing/algorithm design. Each stage in the design was carefully planned to obtain a robust signal from the patient, which provides confidence in the data.

The following aims and objectives will be presented within the thesis.

1) ***Provide a background on the research into the physiology of the heart and stress, and the importance of HRV to aid in building support for the creation of the health watch.*** This is aimed to provide the reader with sufficient medical background, and the technical background to understand the need of the product and the choices for design. **(Chapter 2)**

2) ***Review and describe on a technical level all hardware and software components that were used in the development of the health watch***. Technical reasoning behind the hardware and software design choices will be provided next, taking specifications from datasheets. This will be presented at an engineering level. **(Chapter 3)**

3) ***Proof of concept and design of health watch, block diagram, software control flow, current consumption and full schematic/PCB board and layout design***. A representation of the engineering design will be provided next. Mathematical analysis will be provided where necessary, such as for power considerations. **(Chapter 4)**

4) ***Signal processing and firmware overview including filtering and algorithmic design, as well as post processing.*** Analysis of filter and algorithmic design will be presented next, with comparative results supporting design choices. Post-processing using tools will be introduced, such as MATLAB for further data processing. Background for mathematical understanding and engineering reasoning for the use of each module/algorithm will be presented. **(Chapter 5)**

5) ***The thesis will finish with a brief discussion of its conclusions*** A review of the importance of the proposed health watch and potential future work will be presented. **(Chapter 6)**

Chapter 2:

2 REVIEW OF LITERATURE

This chapter provides an overview of the physiology of the heart, the physiology of stress and a detailed description of the physiological origins of heart rate variability and respiratory rate, and how the two are used as health indicators. An introduction of the measuring techniques and analysis of heart rate variability is covered, which encompasses the main characteristics of photoplethysmography (PPG) and proper analytical interpretations of heart rate variability.

2.1 Physiology of the Heart

The main function of the human heart is to pump blood throughout the body via the circulatory system, which supplies tissues with oxygen and nutrients while removing carbon dioxide and other waste. The heart is a two-sided structure with four chambers. An atrium which is located on both sides of the heart receives the blood flowing inward and which then flows into a ventricle, which pumps the blood out of the heart. Valves link both the atria and ventricles together, which prevents the backflow of blood. The left and right regions of the heart are parts of different circulation systems. The left side controls systemic circulation and receives oxygenated blood, while the right-side controls pulmonary circulation and receives deoxygenated blood. Figure 1 illustrates the structure of the heart.

The heart is its own self sustained pacemaker that sends neural impulses to the cardiac muscle tissue. The pacing centers are the sinoatrial node (SA), and the atrioventricular nodes (AV). The SA and AV nodes initiate depolarization of the cardiac muscle, which is both regular and spontaneous and runs at an intrinsic frequency between 100-120 beats per minute (BPM). Although, resulting heart rates are often much less than this because of the interaction between the surrounding cardiac tissues and complex chemical exchanges. These SA and AV nodes can send impulses to the heart without central nervous system stimulation, but may be influenced by nervous stimulation to alter heart rate.

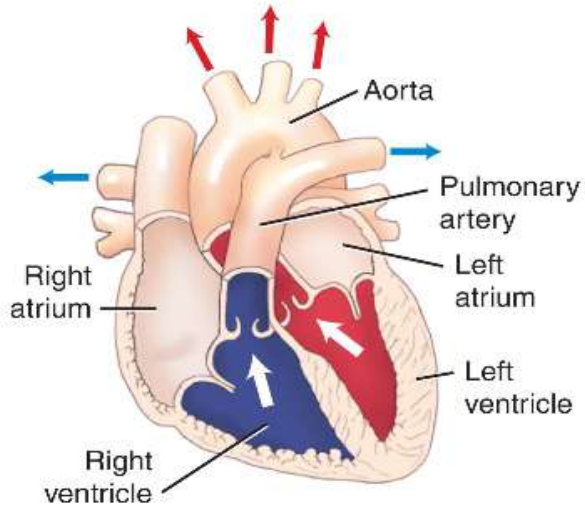

Figure 1 Broad overview of the human heart and its main components [8].

Each heart beat can be described by a single cardiac activity cycle, which can be divided into two basic phases, the diastole and the systole. The diastole and systole are the relaxation and contraction phases of the heart, respectively. The average human heart rate is roughly 75 BPM; and the average length of a cardiac cycle is less than one second, in this brief moment significant pressure changes take place in the heart.

The diastole phase is the low-pressure state of the heart, which allows for blood to flow through the atria and into the ventricles. At this point in time the AV valves are open, and the semilunar valves on the arteries leaving the heart are closed, allowing the flow of blood to fill the ventricles. During the diastole phase both pressure and volume begin to rise in the ventricles until the ventricles are almost full, leaving room for the atria to contract, to add the final volume to the ventricles. This final amount of blood in the ventricles is known as the end-diastolic volume. More end-diastolic volume results in more distention of the ventricle, and ultimately increased contraction strength. Once the ventricles are filled the systole phase begins. The ventricles now contract and the AV valves snap closed and the semilunar valves open. During this phase, blood is pushed from the ventricles into the arteries leaving the heart. Arterial blood pressure is the force exerted on the walls of blood vessels during the cardiac cycle (systole and diastole periods), this creates circulation through a closed loop circuit of blood vessels. The standard arterial blood pressure waveform is shown in Figure 2, this is also the waveshape resulting from a measurement technique, known as plethysmography. Plethysmography will be discussed further in section 2.3.8.

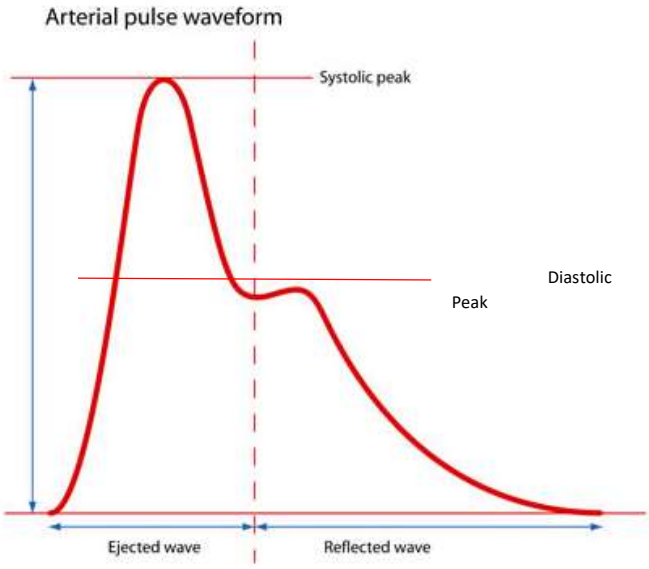

Figure 2 Arterial waveform showing systolic and diastolic peaks [9].

2.1.2 Brain Heart Interaction

Until the 60's and 70's physiologists believed that the brain controlled all physiological response in the body due to internal and external stimuli. This assumed that all cells, organs and tissues are kept in steady-state conditions solely by the brain. Due to the advancements in technology, and our ability to monitor the body's signals to greater depths, this belief has been greatly debunked. In 1974, French researchers stimulated the vagus nerve (which carries many signals from heart to brain) in cats and found that the brain's electrical response was reduced to about half its normal rate, suggesting that the heart and nervous system were not simply following the brains directions [10]. This lead to the discovery of the intrinsic cardiac nervous system (ICNS), known also as the heart-brain, and a new field of science "neurocardiology".

The ICNS is made up of much of the same complex circuitry that is found in the network of the brain. It is described to have ganglia, neurotransmitters, proteins and support cells. This

allows the ICNS to act independently from the brain to learn, remember and make decisions. The ICNS still communicates to the brain via the sympathetic and parasympathetic branches of the autonomic nervous system (ANS), which are embedded within the ICNS. The ANS and signals from sensory neurons in the heart make up the communication pathways of the ICNS. The sensory neuron signals carry the information about the hearts current state, such as pressure, heart rate, heart rhythm and hormones. The communication between the ANS and ICNS has been well known to be an efferent (descending) communication stream, meaning the majority of information is sent from the brain to the heart, which creates its regulation. But recent research has suggested that the majority of fibers that make up the vagus nerve are an afferent (ascending) communication stream. The vagus nerve, which is controlled by the parasympathetic nervous system has more neural pathways connected to the heart than any other organ, this indicates that the heart sends more information to the brain, than the brain to the heart [11].

The cardiac function discussed in 2.1 can partially be described as controlled by the ICNS. Where the ICNS processes the signals that are being linked to it and acts accordingly to control the heart for different instances. Under normal physiological conditions, the ICNS has control over the cardiac cycle, independently of the central nervous system. The spinal column and vagus nerve create the communication pathways for the ICNS to the brain, where it travels to the medulla, hypothalamus, thalamus and amygdala and then to the cerebral cortex. [12]. This is described further in Figure 3. Since we now know that the brain and heart are more deeply connected, signals from the heart can give a greater insight into the brain and causes or signs of disease and stress.

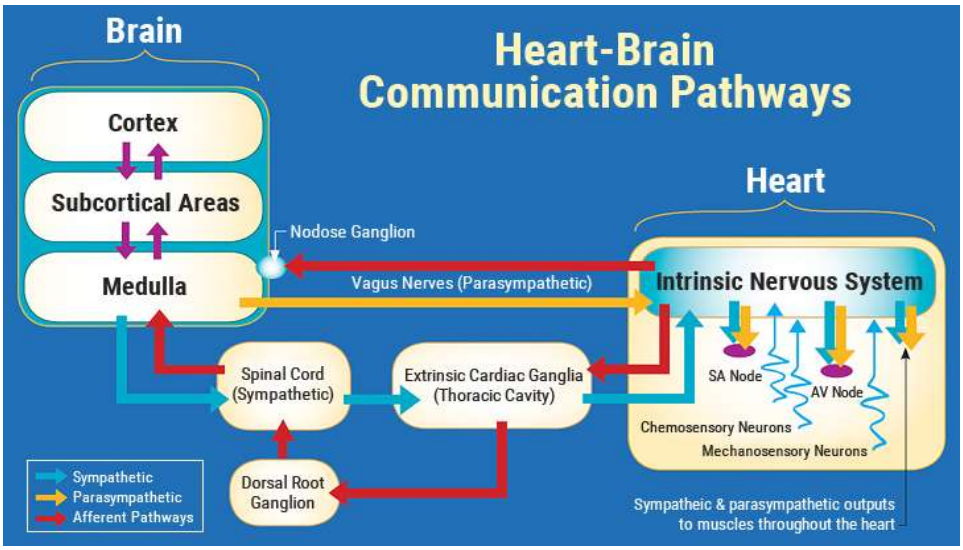

Figure 3 ICNS communication pathway to the brain [10].

2.1.3 Heart Disease

According to the WHO (World Health Organization) and the CDC, heart disease is the leading cause of death in the UK, USA, Canada and Australia. The number of US adults diagnosed with heart disease stands at 26.6 million (11.3% of the adult population) and accounts for 23.5% of all deaths in the USA today. Coronary heart disease alone kills over 370,000 people each year [13]. Heart disease is a broad term that encompasses any type of disorder that affects the heart. This section will briefly explain a few of the different types of heart disease.

The most common and serious types of heart disease are congenital heart disease and coronary artery disease. Congenital heart disease is due to birth defects that affects how the heart works. The defects that arise from congenital heart disease are septal defects, obstruction defects and cyanotic heart disease. Septal defects are when there is a hole between

the two chambers of the heart, which creates problems in function and blood flow. Obstruction defects are when the chambers of the heart are either partially, or fully blocked, leading to lack of blood flow. Cyanotic heart disease is due to the heart malfunctioning in a such a way that it cannot create enough oxygen to be pumped around the body [14]. Since these are birth defects, there is little prevention that can be done.

Coronary artery disease (CAD) is caused by a buildup of plaque on the inner walls of the coronary arteries. The arteries that supply blood to the heart become hardened and narrowed from the buildup of plaque creating a decrease in blood flow through the arteries. The plaque buildup is called atherosclerosis. When blood flow decreases through the coronary arteries the heart can't get the amount of blood or oxygen it needs to function properly. In the most severe cases, CAD leads to a heart attack from a blood clot cutting off part of the heart's blood supply. Long term effects of CAD weaken the heart and lead to heart failure and arrhythmias [15]. Heart failure does not necessarily mean the heart has failed, rather, it cannot adequately supply the body with the proper amount of blood anymore. Arrhythmias are simply changes in the normal rhythm of the heart, which can be increased or decreased, but both are deviations from the normalcy of the heart rate.

2.2 Physiology of Stress

Stress is a part of the genetic makeup of mammals, and it is what initiates the fight or flight reaction. Humans have the fight or flight response imprinted in their genes, but have also adapted to respond to different types of stress in different situations. For many the first onset of

stress comes when we reach college, often as the feeling that resides over us during an exam, or an oral report or presentation. We have come to know the symptoms of the stress response, but few know the underlining physiology driving it. This section will cover the definition of stress, the chemistry and physiology of stress, and the different level of repercussions from stress.

Psychophysiology as it is now regarded is the term to describe the body's physiological reaction to stressors, which is understood to be a mind-body interaction. In the human body, there are three known physiological systems that govern the stress response, the nervous system, the endocrine system and the immune system. Each one of these three systems can be triggered by stressors, or by perceived threatful situations. The nervous system is made up of two parts, the central nervous system (CNS) and the peripheral nervous system (PNS). The limbic system, which is in the mid-level of the brain, is part of the CNS and the emotional control center. The thalamus, hypothalamus, amygdala and pituitary gland all reside in the limbic system. These four glands are what keep the body's systems coordinated. The amygdala is the first gland in the body to register fear, making it the body's alarm system [16]. The hypothalamus potentially plays the most important role in the reaction to stress, and when fear is registered it carries out four functions: (1) it activates the autonomic nervous system; (2) it stimulates the secretion of adrenocorticotropic hormone (ACTH); (3) it produces antidiuretic hormone (ADH); and (4) it stimulates the thyroid gland to produce thyroxin [17], as depicted in Figure 4. Although for the context of this section the most important role of the hypothalamus is to activate the sympathetic and parasympathetic nerves, and have direct control over the endocrine system [18].

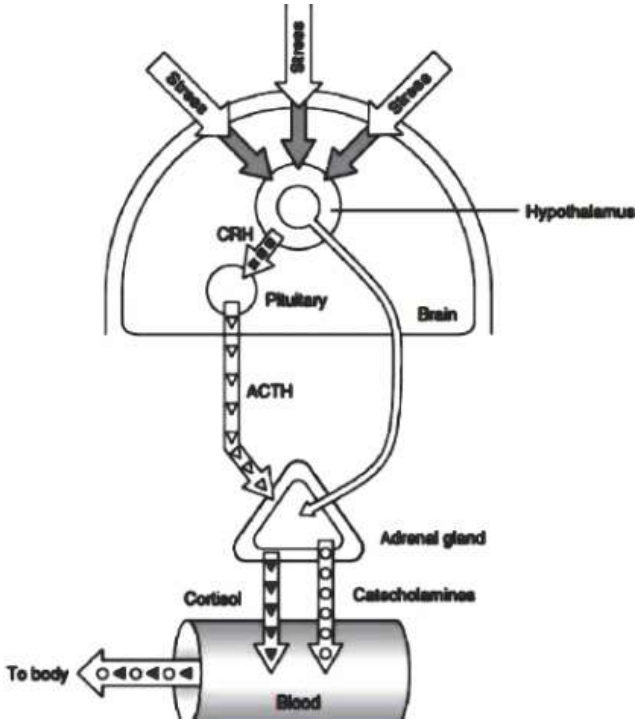

Figure 4 Mechanisms of stress in the body [19].

2.2.2 The Autonomic Nervous System (ANS)

The ANS affects all the internal organs, including the blood vessels, stomach, intestine, liver, kidneys, bladder, genitals, lungs, pupils, heart, and sweat, salivary, and digestive glands [20]. The ANS is made up of two sub systems, the sympathetic and parasympathetic nervous systems. The sympathetic and parasympathetic nervous systems can be described as the stimulators and inhibitors of the body. The sympathetic system stimulates the body during physically or mentally stressful situations, causing increase in cardiac output, HR, blood flow in muscles and a decrease in digestive system activity. Because of its response to stimuli, it is gives rise to the “fight or flight” response. The parasympathetic system causes the opposite effects to occur. It decreases HR and blood pressure and increases the digestive system activity. For these

reasons, it is known as the “rest and digest” response. Overall the two systems work in unison to make sure the body responds correctly to different situations.

The sympathetic system releases two substances, epinephrine (adrenaline) and norepinephrine (non-adrenaline). The release of these two substances warns the body of an upcoming metabolic change and physical movement. The sympathetic drive is the outcome of this, where energy expenditure takes place in the preparation of body movement. It is the supply of epinephrine and norepinephrine that increases the HR and cardiac output. The effects of epinephrine and norepinephrine last only a few seconds, and because of this the sympathetic system is said to have immediate effects.

The parasympathetic system is known for its conservation of the body’s energy. This is the time when cells can regenerate in the body, and is known as the anabolic state. The function of the parasympathetic system is modulated by the vagus nerve, which is influenced by the brain stem. When prompted the parasympathetic nervous system releases acetylcholine (ACh), which returns the body to homeostasis. This is what causes the decrease in HR and blood pressure. The sympathetic system is the dominate force during moments of stress, and the parasympathetic system is there to make sure the system returns to homeostasis after these moments. Figure 5 shows the origins of the two systems and some of their controls.

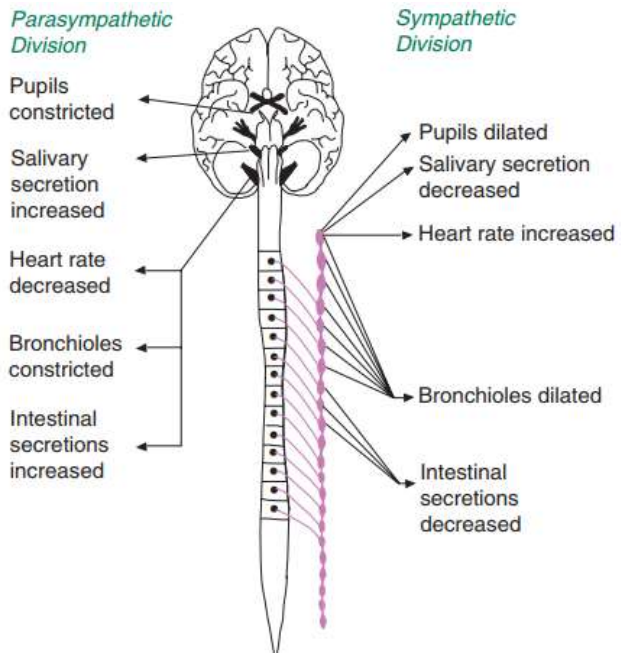

Figure 5 Origin and controls of sympathetic and parasympathetic nervous systems [17].

2.2.3 The Endocrine System

The endocrine system regulates metabolic function, sleep, mood and many other things through the production and release of hormones through a collection of glands. These glands are the pituitary gland, thyroid gland, parathyroid glands and adrenal glands [21]. The gland that is most directly involved with stress response is the adrenal gland. The adrenal gland is comprised of two parts, the adrenal cortex and the adrenal medulla. The function of the adrenal cortex is to create and release hormones known as corticosteroids. Corticosteroids are a group of biochemical agents that include cortisol and cortisone. Cortisol is released in response to fear or stress by the adrenal glands as part of the fight-or-flight response, and is known as the stress hormone. Cortisol helps generate glucose during physical activity, aiding the CNS and skeletal muscles as an energy source. Too much cortisol release can lower immune

function and bone density, increase weight gain, blood pressure, cholesterol and heart disease [22]. Chronic stress can keep cortisol levels high throughout the day keeping the body in a state ready for action. If this physical tension is not released, then the increased cortisol levels in the blood can wreak havoc on the mind and body.

The interior of the adrenal gland is the adrenal medulla. The adrenal medulla is similar to the sympathetic system, because it also secretes epinephrine and norepinephrine into the blood. The ratio of secretion is not balanced and is known to be 80 percent epinephrine and 20 percent norepinephrine. Under stress the adrenal medulla can release up to three hundred times the normal amount of epinephrine into the blood stream, and once again this can wreak havoc on the body if not dealt with properly [23].

The combination of the sympathetic nervous system and the adrenal medulla is evolution's way of increasing its probability of survival. During a fight or flight reaction there could be a failure of one system to efficiently supply the body with the needed amount of epinephrine, therefore the adrenal medulla acts as a backup to the sympathetic system. The hypothalamus initiates the sympathetic system for an immediate affect as mentioned earlier, while the posterior hypothalamus initiates the adrenal medulla through the sympathetic preganglionic neuron. Since epinephrine is released through the neural endings by the sympathetic system, its response is immediate and short lived. The release into the bloodstream has a slower response time, and for this reason the hormonal response from the adrenal medulla is known as the intermediate stress effect. The effects of epinephrine in the

blood stream can last up to two hours during high times of secretion, creating lasting effects of stress awareness [17].

Fight or flight is innate to most animals as a form of stress response, but human beings are not limited to just this response. We have the ability to self-regulate when encountered by challenges, disagreements and stressors. The vagus nerve acts as a braking mechanism for our ability to self-regulate under distress and allows a sense of calming to overcome the body by negating sympathetic response to the heart and adrenal glands. Because vagal activity has this type of control of the body, measurements of its activity may serve as a marker for self-regulation. This suggests that the overall health of the ANS system determines a person's emotional status and their ability to regulate their emotions and behaviors [24].

2.2.4 Stress and Heart Disease

Current research has focused on the link between heart disease and stress, and ways to decrease one by decreasing the other. Coronary heart disease is also much more common in individuals subjected to chronic stress and recent research has focused on how to identify and prevent this growing problem, particularly with respect to an individual's job stress [25]. The INTERHEART study investigated the relation of chronic stressors to incidence of myocardial infarction (MI) in a sample of approximately 25,000 people from 52 countries. Stress was defined as "feeling irritable, filled with anxiety, or as having sleeping difficulties as a result of conditions at work or at home." After adjusting for age, gender, geographic region, and smoking, those who reported "permanent stress" at work or at home had greater than 2.1 times the risk for developing MI [26]. Data available from MRFIT (Multiple Risk Factor

Intervention Trial) was used to track 12,336 men over the course of 9 years to determine if chronic stressors such as work stress and marital discourse had increased their mortality rate from coronary heart disease. The findings showed work and marital stressors have up to a 2% increase in mortality rate due to coronary heart disease in men [27].

In 2015 adults in America started showing more concern and awareness toward stressors in their daily lives. A substantial number of these adults report unwanted side effects such as headaches, feeling anxious or depressed, and some have reported behavioral changes such as yelling at loved ones and becoming unsocial. Thirty two percent of adults recognize and claim stress has an impact on their mental health and body/physical health. And forty two percent of adults claim to feel mental health symptoms due to stress, such as anxiety and depression [28]. Figure 6 shows the increase in stress from 2014 to 2015 in the United States. The importance of stress management and stress awareness is becoming ever increasing in developed nations where day to day chronic stressors are a part of life.

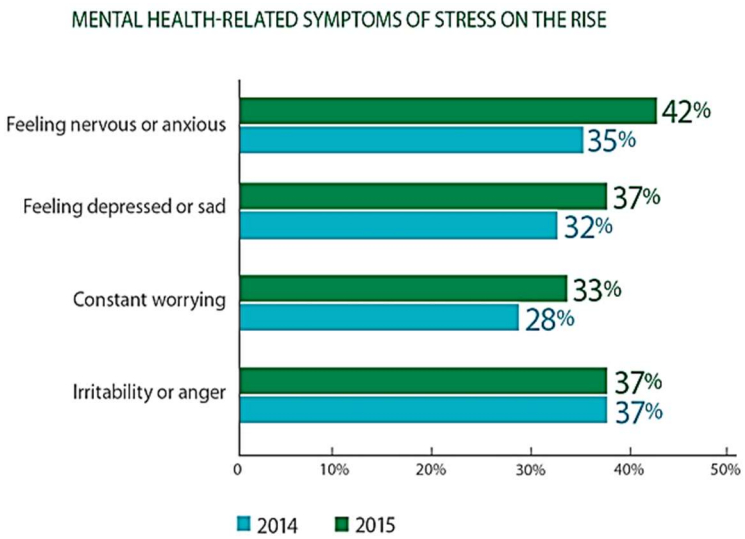

Figure 6 Stress increase in the United States between 2014 and 2015

2.3 Heart Rate Variability (HRV) and Beat to Beat Interval

2.3.1Heart Rate Variability (HRV)

The heart beat varies on an interbeat interval, which is simply the time between the QRS peaks of the ECG waveform, known as the R-R interval. The baseline heart rate which is often averaged over a 60 second period, is roughly 75 BPM for healthy adults. But as in the instance shown in Figure 7, the heart rate varies beat to beat roughly in the range of 50-80 BPM. This phenomenon of the heart rate rising and falling over given time periods is HRV. Assessing HRV has proven to be an important non-invasive method in uncovering information about cardiac autonomic modulation. Measuring the time between the R-R peaks reflects autonomic control of the parasympathetic and sympathetic influences on the heart.

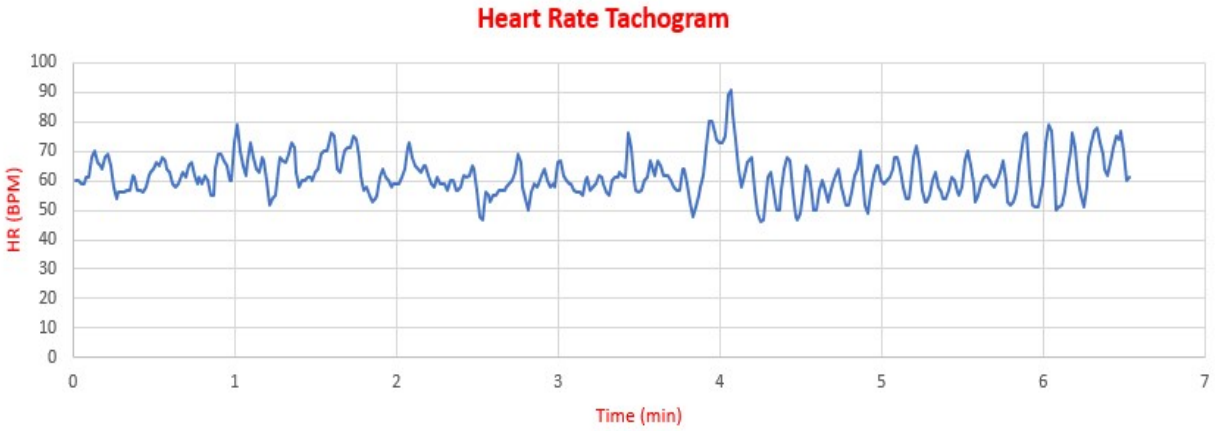

Figure 7 HR tachogram, recorded by the health watch.

Heart rate variability has been the focal point of a lot of research over the past decade, mainly due to its insights in determining the body's natural response to stimuli. The irregularity

of the beat-to-beat intervals of a human heart was first noted in the early 1600's [29], but its physiological importance was not appreciated until 1965 when Hon and Lee [30] noted that fetal distress was preceded by changes in the pattern of beat-to-beat intervals before any significant change in the baseline heart rate. It was once believed that human physiology strives to be in a steady state, and that variations away from the steady state indicated an unhealthy individual, but we now know biological processes vary in complex and nonlinear ways. For example, the heart is highly variable rather than being monotonously regular, this is expressed as HRV. Hart-brain interactions, as well as autonomic nervous system dynamics reflects HRV. Also, HRV has been shown to predict autonomic neuropathy in diabetic patients before the onset of symptoms [31], and when HRV is at lower levels it is linked to higher risks of fatal post-myocardial infarction [32].

Optimal levels of HRV within an individual reflects a healthy self-regulatory nervous system, while too much instability can indicate nervous system chaos. However, to little variation has indications toward aging, disease, depression, stress and anxiety. There may be a correlation with disease and mortality due to HRV, because it reflects a reduction in an individual's ability to adapt to physiological changes. A healthy body is constantly changing and adapting to external and internal stimuli. Therefore, an important indication of health is how well the regulatory system is reacting to stimuli and adjusting accordingly. As mentioned in the previous section, regulation is how well balanced the parasympathetic and sympathetic systems are. For example, during exercise the regulatory system must raise the heart rate, respiration, and metabolism to be able to perform at the level being asked of it. An individual with poor HRV will struggle to perform during these activities.

2.3.2 Reflex Control and Factors Affecting Heart Rate

The ANS is stimulated through different receptors in the central nervous system and cardiovascular system. These receptors play an important role in modulating the heart, and modulate the heart rate through tachycardia and bradycardic effects. These effects are broken into groups of reflexes that control the variability in the heart. The following is an overview of two reflexes, the Bainbridge and the baroreceptor. The Bainbridge reflex and the baroreceptor reflex act antagonistically to control heart rate. The baroreceptor reflex acts to decrease heart rate when the BP rises. When blood volume is increased, the Bainbridge reflex is dominant; when blood volume is decreased, the baroreceptor reflex is dominant [33].

The baroreflex is a homeostatic mechanism that helps to keep blood pressure at a constant level. It acts as a negative feedback loop that in which raised levels of blood pressure causes the heart rate of an individual to decrease, while the opposite happens when blood pressure levels fall. Baroreflex is a quick response mechanism, often occurring in less than a cardiac cycle [34]. The Bainbridge reflex is prompted by stretching of the right atrial wall. The filling pressure sends vagal afferent signals to the cardiovascular center of the medulla, which inhibits parasympathetic activity increasing ones' heart rate stimulation [35].

Regulation from the sympathetic and parasympathetic systems are not the only factors that regulate and modulate changes in heart rate. Heart rate changes can be caused by many factors such as Respiratory Sinus Arrhythmia (RSA), exercise, age and circadian rhythm, just to name a few.

2.3.3 Respiratory Sinus Arrhythmia (RSA)

Respiratory Sinus Arrhythmia is a physiological interaction between respiration and circulation, which shortens the R-R interval on inspiration and prolongs it during expiration. It can be shown from Figure 8 that RSA is the synchronization of HRV. As the figure shows, the peaks of the respiration signal and the HRV curve approximately line up in time, showing their synchronization and the effects of one on the other.

Physical action within the lungs partly influences RSA, and previous studies have shown that RSA improves the efficiency of pulmonary gas exchange between the lungs and blood, and fluctuations in the occurrence of heartbeats in phase with lung volume can enhance oxygen transfer [36]. The magnitude of RSA is claimed to provide an index of the vagal activity of the heart, since RSA is known to be mediated through changes in efferent vagal activity [37]. There is still active research into RSA and its effects on the heart and body.

Since both HRV and respiratory rate (RR) are associated with the sympathetic and parasympathetic nervous system, they are tightly correlated to one another. It has been shown that relatively small changes in natural breathing rates, in the region between 7.5-15 breaths per minute, can cause quite significant changes in measured HRV [38]. Therefore, when recording HRV, the RR should also be recorded.

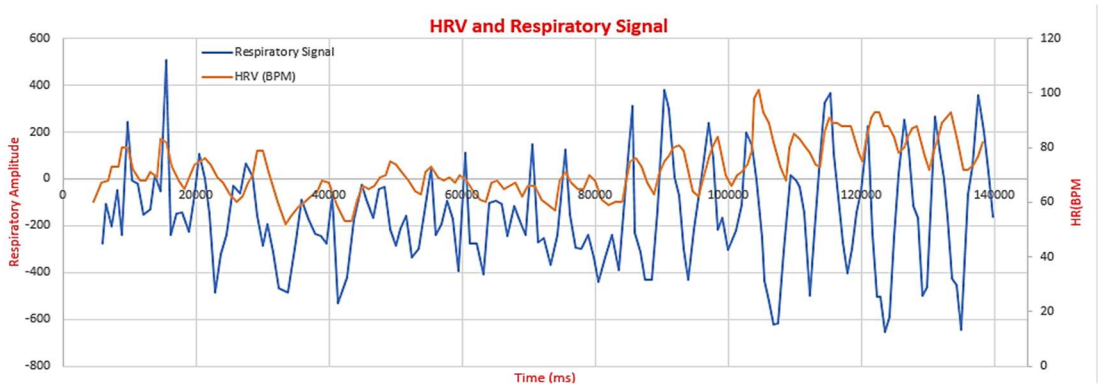

Figure 8 Respiration signal and HRV, shows the ralationship due to RSA.

2.3.4 Exercise Age and Circadian Rhythm

During rest HRV gives an indication into the functionality of ones ANS, though caution should be taken when measuring HRV during exercise, age differences and different times of day. It is well known that exercise increases ones HR in need of supplying oxygen to muscles to create energy for contraction. During elevated heart rates HRV will become diminishingly significant, because the R-R intervals become increasingly small and difficult to differentiate. Age also effects HRV, and it has been shown that youth and older individuals have a lower value of HRV. There is still much study going on concerning these factors and few conclusions have yet emerged.

The circadian rhythm has been shown to also play a role in HRV. A circadian rhythm is any biological process that displays oscillation of about 24 hours driven by a circadian clock [39]. This is most prominent in humans during their sleep periods, as shown in Figure 9, where the heart rate is much lower during the hours of sleep. The multiple HRV parameters shown are

different during sleep times as well, indicating that the variation is affected by the health of a patients ANS.

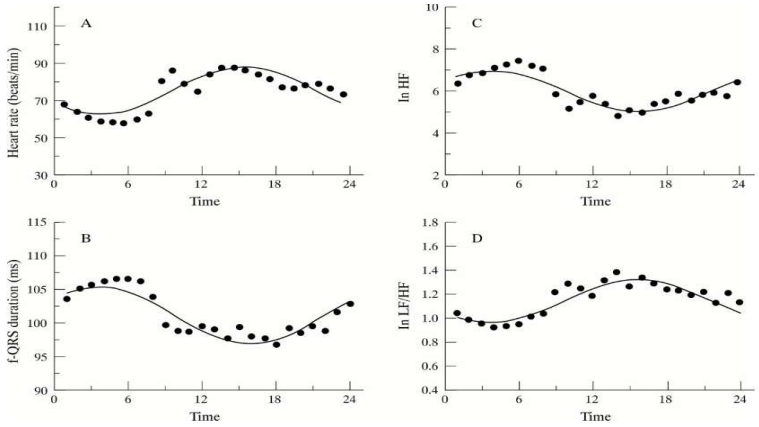

Figure 9 Circadian rhythm of HRV over 24 hr period [40].

2.4 HRV and Analysis

In common practice the most reliable measurement technique for acquiring heart rate and therefore HRV is the electrocardiogram (EKG or ECG). The EKG tracks the heart's electrical activity and produces a distinctive waveform with well-defined R-R intervals for HRV analysis. EKG recording is developed in two main forms: (1) as a mobile machine with multiple electrode leads that are placed on the chest; and (2) as a chest strap with built in electrodes, designed for fitness tracking. While the EKG is the clinical standard for use within the doctor's office or hospital, it is not convenient for mobile daily use. A technique known as photoplethysmography (PPG) has become a new standard in HR monitoring thanks to recent advancements in optical technology.

2.4.1 Analysis Methods

Analysis of HRV has many approaches, but the most commonly used are frequency domain analysis and time domain analysis. Both methods require the determination of the R-R intervals within the acquired signal, then further signal processing is done to determine the standard metrics of HRV. The physiological actions in the body that were discussed earlier, such as respiration and ANS control produce short and long term rhythms in HRV [41]. Visually a HR tachogram plot can be used to determine the HRV of an individual, as shown in Figure 10. For real time analysis of HRV, both time domain and frequency domain analysis are used. The health watch makes use of time domain analysis, so only it will be further discussed.

Figure 10 Heart rate tachogram acquired from health watch.

2.4.2 Time Domain Analysis

The following definition for measurement of HRV is taken from the *Task Force of The European Society of Cardiology and The North American Society of Pacing and Electrophysiology* [42].

"*From a series of instantaneous heart rates or cycle intervals, particularly those recorded over longer periods, traditionally 24 h, more complex statistical time-domain measures can be calculated. These may be divided into two classes, (a) those derived from direct measurements of the NN intervals or instantaneous heart rate, and (b) those derived from the differences between NN intervals. These variables may be derived from analysis of the total electrocardiographic recording or may be calculated using smaller segments of the recording period. The latter method allows comparison of HRV to be made during varying activities, e.g. rest, sleep, etc.*

The simplest variable to calculate is the standard deviation of the NN interval (SDNN), i.e. the square root of variance. Since variance is mathematically equal to total power of spectral analysis, SDNN reflects all the cyclic components responsible for variability in the period of recording. In many studies, SDNN is calculated over a 24-h period and thus encompasses both short-term high frequency variations, as well as the lowest frequency components seen in a 24-h period. As the period of monitoring decreases, SDNN estimates shorter and shorter cycle lengths. Thus, in practice, it is inappropriate to compare SDNN measures obtained from recordings of different durations. However, durations of the recordings used to determine SDNN values (and similarly other HRV measures) should be standardized, as short-term 5-min recordings and nominal 24 h long-term recordings seem to be appropriate options.

Other commonly used statistical variables calculated from segments of the total monitoring period include SDANN, the standard deviation of the average NN interval calculated over short periods, usually 5 min, which is an estimate of the changes in heart rate due to cycles longer than 5 min, and the SDNN index, the mean of the 5-min standard deviation of the NN interval calculated over 24 h, which measures the variability due to cycles shorter than 5 min. The most commonly used measures derived from interval differences include RMSSD, the square root of the mean squared differences of successive NN intervals, NN50, the number of interval differences of successive NN intervals greater than 50ms, and pNN50 the proportion derived by dividing NN50 by the total number of NN intervals. All these measurements of short-term variation estimate high frequency variations in heart rate and thus are highly correlated. No currently recognized HRV measure provides better prognostic information than the time–domain HRV measures assessing overall HRV. The following table describes the time domain statistical techniques."

Table 1 Time Domain statistical techniques for quantifying HRV [42].

Variable	Units	Description
		Statistical measures
SDNN	ms	Standard deviation of all NN intervals.
SDANN	ms	Standard deviation of the averages of NN intervals in all 5 min segments of the entire recording.
RMSSD	ms	The square root of the mean of the sum of the squares of differences between adjacent NN intervals.
SDNN index	ms	Mean of the standard deviations of all NN intervals for all 5 min segments of the entire recording.
SDSD	ms	Standard deviation of differences between adjacent NN intervals.
NN50 count		Number of pairs of adjacent NN intervals differing by more than 50 ms in the entire recording. Three variants are possible counting all such NN intervals pairs or only pairs in which the first or the second interval is longer.
pNN50	%	NN50 count divided by the total number of all NN intervals.

2.5 HRV Measurement Method (PPG)

Photoplethysmography is a noninvasive optical technique used for acquisition of the heart rate. PPG detects changes in blood volume in peripheral tissues by making use of different wavelength LEDS to detect differential optical absorption. Light traveling through biological tissues gets absorbed by bone, skin pigments and both venous and arterial blood. The venous and arterial blood absorbs more light compared to other tissues, which allows for changes in the intensity of the received backscattered light due to changes in blood volume. The signal output is the backscattered optical intensity, which creates a photocurrent in the photodetector that is in inverse proportion to the volume of blood flowing in the optically excited area. PPG signals are mainly acquired at the wrist, fingertip and earlobe.

2.5.1 LED Wavelength

The choice of the optimal excitation wavelength has been of debate in PPG measurements. For pulse oximetry, the standard is red and infrared wavelengths, typically 630 and 900 nm. These longer wavelengths were chosen because they penetrate deeper into the body, thus being good candidates for transmissive PPG, which is usually taken on the finger. It is also known that shorter wavelengths such as green (around 550 nm) penetrate less than longer wavelengths, but still penetrate deep enough to interact with arterial and venous blood flow. Because green wavelengths do not penetrate as deeply, they are less likely to be influenced by deeper tissue movements. Therefore, they have more recently been accepted as the wavelength of choice for reflective PPG [43]. Commercial LEDs are readily available in these wavelengths, and have been the most commonly used optical sources for PPG measurements. Figure 11 shows both reflective and transmissive techniques taken on the finger.

2.5.2 Reflective PPG

Reflectance mode is when the photodiode detects light that is backscattered or reflected from the tissue, bone, or blood vessels. It is used more often in wearable devices since sensor placement (LED and photodiode) does not have to be perfect. Reflection mode PPG is usually accompanied by unwanted affects known as motion artefacts. Motion artefacts are caused by nearly any physical movement between the tissue and the sensor. The movement, such as physical activity, leads to motion artefacts that corrupt the PPG signal. This leads to inaccuracies and interferences of the bio signals being acquired from the sensor. Great care must be taken to minimize the effects of motion artefacts, such as the use of pre-processing and digital algorithms.

There has been much literature written on the effects and reduction of artefacts in the PPG signal over the past decade, but no fool proof solution has yet been presented.

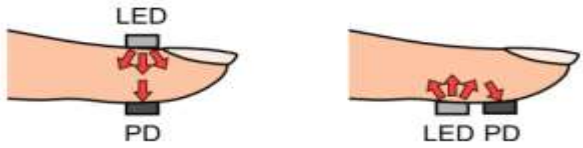

Figure 11 Reflective and transmissive PPG techniques [43].

2.5.3 AC and DC Components

The PPG waveform consists of two components, an alternating current (AC) component and a direct current (DC) component. The AC component changes with blood volume pulsations that occurs between the systolic and diastolic phases of the cardiac cycle, and the heart rate determines the fundamental frequency of the AC component [44]. The slowly varying DC component, in which the AC signal is superimposed on, changes due to respiration, sympathetic nervous system activity and thermoregulation. Because each component has different characteristics that cause its change in frequency, the PPG signal can be used for a wide range of different medical purposes. The signal is used by physicians for clinical physiological monitoring, blood oxygen saturation, blood pressure, cardiac output, heart rate and respiration rate just to name a few. Heart rate and respiration rate are the most widely used metrics of the PPG signal due to the vast amount of information that they provide about the health state of an individual. Since the heart rate is defined by the AC component of the PPG and the respiratory rate is defined by the DC component, it is possible to separate the two and record their values

in real time. Figure 12 shows an example of a photoplethysmograph waveform, consisting of DC and AC components.

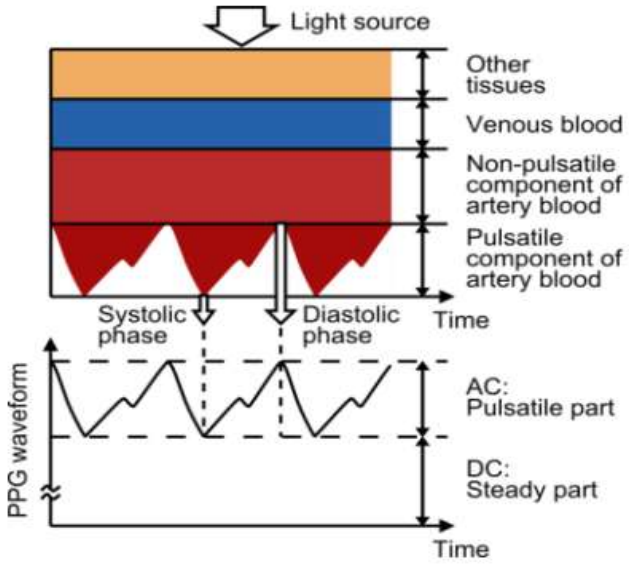

Figure 12 Insight into the production of the photoplethysmograph waveform [44].

Chapter 3:

3 HARDWARE and SOFTWARE

3.1 Overview

Chapters 3 provides an overview of the hardware and software used in the development of the health watch. This includes the integrated circuit (IC) choices, such as the microcontroller, the analog front end (AFE), and the accelerometer, followed by the software language and compiler choice.

3.2 Hardware Components

Hardware components make up a portion of an embedded system. Any engineering system that is controlled by computer software is an embedded system. In many devices, such as the health watch, the embedded systems main function is to control hardware components for a desired purpose. The following section discusses the hardware components in detail that are in the health watch.

3.2.1 Wrist Watch

The health watch is made up of a group of modules that communicate and interact with each other to create the desired outputs. The hardware modules that make up the health watch are as follows:

- TM4C123GH6PM microcontroller

- AFE4404 ultra-small, integrated AFE for wearable, optical, heart-rate monitoring and bio-sensing
- Battery
- MSP73831 miniature single cell, fully integrated Li-ion, Li-polymer charge management controller
- 4 GB SD-card
- SFH7051 BIOMON sensor
- MMA8451Q accelerometer

3.2.2 Microcontroller

Microcontrollers (MCUs) are small computers printed on an integrated circuit. Their functionality depends on the software program that the user loads on it. Like computers, microcontrollers have a memory system, a central processing unit (CPU), an I/O system, a clock/timing system, and a bus that connects constituent's systems. The MCU is the brains of any embedded system, and that is also the case with the health watch.

The microcontroller chosen for the health watch is from the Tiva C series microcontroller family developed by Texas Instruments, the TM4C123GH6PM. The TM4C123GH6PM microcontroller is recommended by Texas Instruments for use in low power hand held devices and medical instrumentation applications. The TM4C123GH6PM consists of a high-performance ARM Cortex M4 architecture, and has a strong ecosystem of software and development tools. It is supported by the TivaWare peripheral driver library that is rich with

examples and is helpful for prototyping and fast paced product development. Table 2 provides the performance and core features of the TM4C123GH6PM. The most important specifications as they pertain to the current health watch are the performance, PWM and power consumption. The 80MHz clock rate, 16 PWM outputs and hibernation mode in the TM4C123GH6PM make it an ideal candidate for the health watch.

Table 2 Performance and features of the TM4C123GH6PM MCU [45]

Feature	Description
Performance	
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with TivaWare™ for C Series software
Security	
Communication Interfaces	
Universal Asynchronous Receivers/Transmitter (UART)	Eight UARTs
Synchronous Serial Interface (SSI)	Four SSI modules
Inter-Integrated Circuit (I^2C)	Four I^2C modules with four transmission speeds including high-speed mode
Controller Area Network (CAN)	Two CAN 2.0 A/B controllers
Universal Serial Bus (USB)	USB 2.0 OTG/Host/Device
System Integration	
Micro Direct Memory Access (µDMA)	ARM® PrimeCell® 32-channel configurable µDMA controller
General-Purpose Timer (GPTM)	Six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
Watchdog Timer (WDT)	Two watchdog timers
Hibernation Module (HIB)	Low-power battery-backed Hibernation module
General-Purpose Input/Output (GPIO)	Six physical GPIO blocks
Advanced Motion Control	
Pulse Width Modulator (PWM)	Two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs.
Quadrature Encoder Interface (QEI)	Two QEI modules
Analog Support	
Analog-to-Digital Converter (ADC)	Two 12-bit ADC modules, each with a maximum sample rate of one million samples/second

3.2.3 Analog Front End (AFE)

The AFE4404 is used in the health watch for acquisition of the PPG signal. In the health watch, the AFE4404 provides two PPG signals via an I2C interface to the MCU, that are further summed and averaged in firmware to provide a single signal for processing. The following is a description of the AFE4404.

The AFE4404 is an analog front-end (AFE) for optical bio-sensing applications, such as heart-rate monitoring (HRM) and peripheral capillary oxygen saturation (SpO_2). The device supports three switching light-emitting diodes (LEDs) and a single photodiode, and the health watch utilizes two of the three LEDs. The current from the photodiode is converted into a voltage by the transimpedance amplifier (TIA) and digitized using an analog-to-digital converter (ADC). The ADC code can be read out using an I2C interface. The AFE also has a fully integrated LED driver with a 6-bit current control. The device has high dynamic range transmit and receive circuitry that helps with the sensing of very small signal levels. A block diagram of the system is shown in Figure 13 [46].

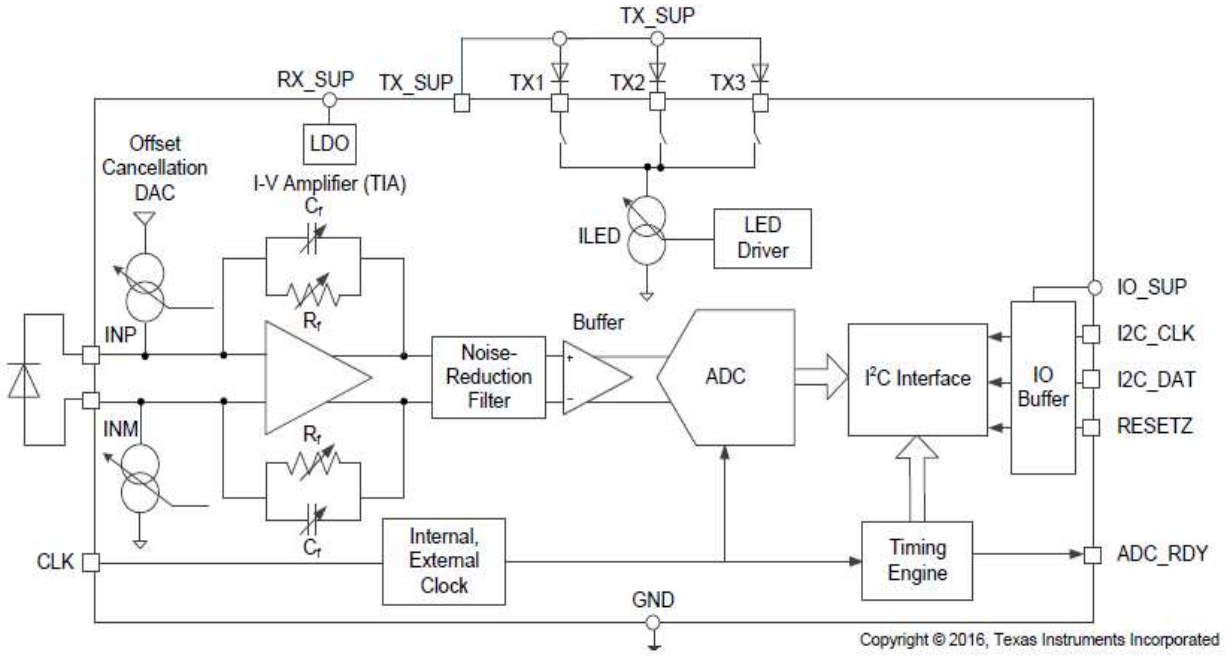

Figure 13 Block diagram of the AFE4404 [46]

3.2.4 Battery

The battery that powers the health watch is a 3.7 V, 500 mAh rechargeable lithium ion cell. Table 3 provides the performance specifications of the battery that are important in the context of the health watch design.

Table 3 Performance and metrics of the 3.7V battery [47]

No	Item	Rated performance	Remark
1	Capacity	Nominal 500mAh Minimum 450mAh	Standard discharge after standard charge
2	Nominal voltage	3.7V	Mean operation voltage during standard discharge after standard charge
3	Voltage at end of discharge	3.0V	Discharge cut-off voltage
4	Charging voltage	4.2V	
5	Standard charge	Constant current 0.2C_5A Constant voltage 4.2V Cut-off current 0.01C_5A	
6	Quick charge	Constant current 1C_5A Constant voltage 4.2V Cut-off current 0.01C_5A	
7	Standard discharge	Constant current 0.2 C_5A End voltage 3.0V	
8	Maximum continuous discharge current	1 C_5A	
9	Operation temperature range	Charge: 0~45℃ Discharge:-20~60℃	60±25%R.H
10	Cycle life	>500cycles	Charging/discharging in the below condition: Charge: standard charge Discharge:0.2C_5A to 3.0V Rest time between charge/discharge:30min Until the discharge capacity <60% of NC
11	Storage temperature	During 1 month: -5 ~ 35℃ During 6 months: -20 ~ 45℃	60±25%R.H

3.2.5 Power Management

The 3.7 V rechargeable Lithium Ion battery that powers the health watch is recharged via USB and is controlled by the MCP73831. The MCP73831 is a simple charge management system designed for use in smaller devices such as the health watch. It monitors the voltage and current from the USB and manages it in such a way to not overcharge the battery, thus protecting its longevity. The following is a description of the MCP73831.

"*The MCP73831 is a highly advanced linear charge management controller for use in space-limited, cost sensitive applications. Along with its small physical size, the low number of external components required makes the MCP73831 ideally suited for portable applications. For applications charging from a USB port, the MCP73831 adheres to all the specifications governing the USB power bus. The MCP73831 employs a constant current/constant voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V, or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73831 limits the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability*" [48].

3.2.6 Memory and Data Acquisition

The data from the health watch, which includes the heart rate, respiratory rate and heart rate variability is written to a 4GB SD-card. The SanDisk 4GB card utilizes NAND flash memory technology. The MCU writes to the SD card vis FATFS, which is a generic FAT-16 filesystem module for small embedded systems. It supports many of the standard input and output filesystem functions of ANSI C. Since the FATFS file size is up to 4 GB (non-high-capacity SD-card), this was the choice in an SD card memory size.

3.2.7 Sensor Array

The SFH 7070 sensor array manufactured by OSRAM is at the heart of the health watch, because the quality of the measurements depends on the quality of the optical PPG signal. The SFH 7070 is controlled and operated by the AFE4404 to provide the PPG signal to the MCU. The following is a detailed description of the SFH 7070.

"*The SFH 7070 (Figure 14) is an integrated optoelectronic sensor, specifically designed and optimized for reflective PPG measurements. It features two green LEDs and a large area photodiode (PD). The device design includes light barriers to minimize internal crosstalk between LEDs and the PD, thus enhancing the signal-to-noise ratio. The sensor allows the precise measurement of the heart rate thanks to its optimized design and technical specifications. The white package contributes to the overall total radiant flux. The photodiode suppresses Infrared (IR) light to reduce IR interference from external light sources, e.g. sunlight. The combination of these features make the SFH 7070 an excellent candidate for PPG measurements*" [49].

Figure 14 SFH 7070 image [49]

3.2.8 Accelerometer

The MMA8451Q developed by NXP Semiconductors was chosen for its broad range of functionalities. The current design utilizes the built in high pass filtering capability to ensure movement is only recorded above a specific frequency, which is set to 2Hz. It can continuously poll data and allows for the choice between 8-bit or 14-bit resolution. 8-bit resolution is used in this revision of the health watch, and data is polled and checked every millisecond. The accelerometers use in the design of the health watch is twofold: (1) it is used for thresholding for the motion of the user, and (2) it is used as a noisy input signal to the recursive least squares algorithm, as discussed in 5.4. If the accelerometer records values greater than ±.156 g's, then the system shuts off all data recordings and algorithm computation. This effectively reduces the potential problem from motion artefacts. Once the accelerometer values drop below the threshold, the system returns to normal operation. The following is an in-depth description of the MMA8451Q.

"*The accelerometer in the health watch is the MMA8451Q, 3 axis, 14-bit/8-bit digital accelerometer from NXP Semiconductors. The MMA8451Q is a smart, low-power, three-axis, capacitive, micromachined accelerometer with 14 bits of resolution. The MMA8451Q is packed with embedded functions with flexible user programmable options, configurable to two interrupt pins. Embedded interrupt functions allow for overall power savings relieving the host processor from continuously polling data. There is access to both low-pass filtered data as well as high-pass filtered data, which minimizes the data analysis required for jolt detection and faster transitions. The device can be configured to generate inertial wakeup interrupt signals from any combination of the configurable embedded functions allowing the MMA8451Q to monitor events and remain in a low-power mode during periods of inactivity*" [50].

3.3 Software

Without software, the hardware modules that make up the physical design of the health watch would not operate. Software provides the instructions for control over the entire system. Software produces the algorithms that in this case determine heart rate, respiratory rate and heart rate variability. Software at its lowest level consists of machine language instructions, which is known as executable code. For ease of use and efficiency, high level languages were developed and are now the standard. A compiler converts high level languages into machine instructions so a computer can execute them. The health watch was designed in a high-level language, which is known as the ANSI C language.

3.3.1 Compiler and C Language

Certain compilers work with different processors, and processor architectures. The health watch utilizes an ARM architecture. ARM architectures have many different compilers to choose from due to their popularity. The ARM compiler is specifically designed to optimize software running on ARM processors. It has been in development for over 20 years and has been the heart of billions of devices. The ARM compiler incorporates an assembler, linker and libraries for embedded software development.

The software development environment and compiler used in the health watch is Keil MDK. Keil is used for a wide range of ARM Cortex-M based microcontrollers, like the architecture in the TM4C123GH6PM. The Keil environment includes an IDE and debugger, both of which are vital to thorough embedded software design. Keil supports both ANSI C and C++, so it is an ideal choice for the health watch.

Chapter 4:

4 Design and Layout

Chapter 4 provides the full design and layout of each major module of the health watch. This includes a block diagram, software control flow, power, schematic design and PCB layout.

4.1 Block Diagram

Figure 15 is a high-level block diagram comprised of each hardware module controlling the health watch. The system is powered by a 3.7 V, 500 mAh lithium ion battery. The 3.7 V battery directly powers the AFE and sensor array. The battery is charged via a mini-USB interface and is controlled by a linear battery management IC. A low-dropout regulator (LDO) regulates the voltage to 3.3 V to power the MCU, accelerometer, AFE4404 and SD-card. The MCU provides a 6 MHz clock to and communicates via an I2C interface with the AFE4404. The MCU also communicates with the accelerometer via the same I2C interface and writes to the SD-card using FATFS. The AFE provides an interrupt signal to the MCU at a 100 Hz rate, which is a user controlled sample rate of the PPG signal. The AFE also modulates the sensor array LEDs and reads the resulting photodiode current.

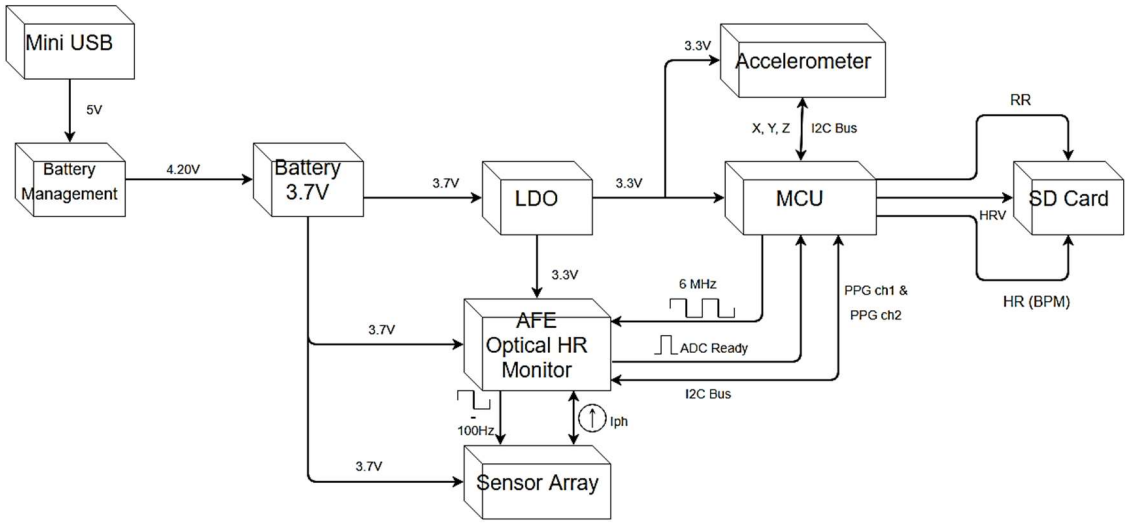

Figure 15 Block diagram of the health watch system design

4.2 System Control Overview

Figure 16 shows an overview of the control software, although this is not the entirety of the control flow, but only the higher-level structure. It starts with the main() function and proceeds as follows:

1. The PWM is initialized to provide the 6 MHz clock to the AFE.
2. The AFE system is then configured, and this includes the sample rate, LED current values, transimpedance gain, clock settings and offset cancelation DAC current settings.
3. The accelerometer is then initialized, which sets its sampling rate, the high pass filter cutoff and bit resolution.
4. The interrupt priority is set as follows:
    a. AFE4404 ADC_Ready interrupt
    b. Time

c. Accelerometer data

5. The ADC_ready interrupt is initialized.

6. Timer interrupts for the time and accelerometer data are initialized.

7. System calibration is performed. The calibration starts with the initial settings for the AFE at 25 mA, 2 MΩ transimpedance feedback gain and 0 offset cancelation DAC current. The values get updated until the signal at the ADC is between ±1V, which is .4 V less than the maximum range.

The system then enters the while (1) loop:

8. The activity count from the acceleration values are compared to a threshold, and if the values are above the threshold the system shuts down the interrupts and keeps checking until the motion has lessened. This ensures that corrupted data from motion artefacts are not recorded.

9. An ADC Ready signal is transmitted from the AFE4404, and if the interrupt has been detected the system stores the PPG signals in a buffer along with the associated time.

10. The Software Timer sends an interrupt if the timer overflows, and if interrupts have been detected, the system updates the time, and or updates the accelerometer data.

11. The system checks if 5 seconds of data has been recorded and stored in the buffers.

12. Noise is removed from the data using HPF and LPF filters.

13. Peaks and troughs are detected.

14. If a peak or trough is found then the system computes respiratory rate, heart rate and heart rate variability.

15. If 5 minutes has passed, heart rate variability is recorded to the SD-card, and if 5 heart beats have been detected, the HR and RR are also recorded to the SD-card.

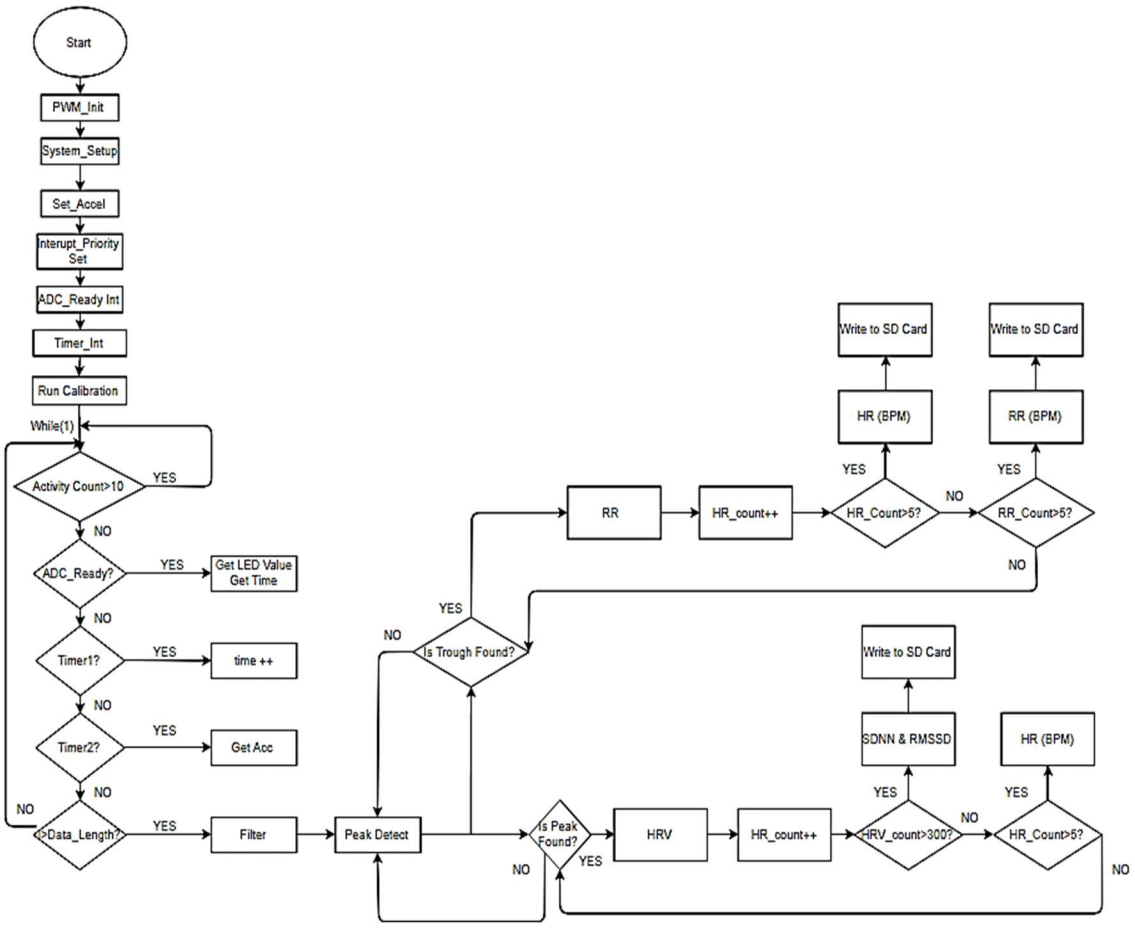

Figure 16 Software control overview diagram

4.3 Power

The AFE has two modes in which it can operate: (1) a dynamic system power down, and (2) an always-on mode. The dynamic system power down allows for the system to be shutoff after the PPG signals have been sampled. The always-on mode leaves the power on during these times. The health watch makes use of the dynamic system power down, and so this

power consumption is reported. The LED's have a duty cycle of 100 µs and the PPG signal is sampled at 100 Hz, which play an important role in current consumption. Table 4 provides a list of each IC, and their current consumption, as well as the total estimated current consumption.

Table 4 Device and current consumption overview

Device	Average Current Consumption, (µA)
AFE4404 (Analog Front End)	325
TM4C123GH6PM (Microcontroller)	4400
MMA8451 (Accelerometer)	165
LP2985 (LDO)	75
Total Current	**4965**

From Table 4 it can be deduced that the system should last approximately 100 hours using a 500 mAh battery. This indicates that it would behoove the design to choose a smaller capacity battery. For example, an 80 mAh battery would last roughly 16 hours, which is a standard day of use. Further information on current consumption can be found in the datasheets in the references.

4.4 Schematic/Circuit Design

The following represents the circuit design for each module in the system, which was produced in Multisim software. Figure 17 shows the circuit design for the LDO, power management system (MCP73831), analog front end (AFE4404), accelerometer (MMA8451Q), micro-USB adapter, SD-card adapter and the sensor array (SFH 7070).

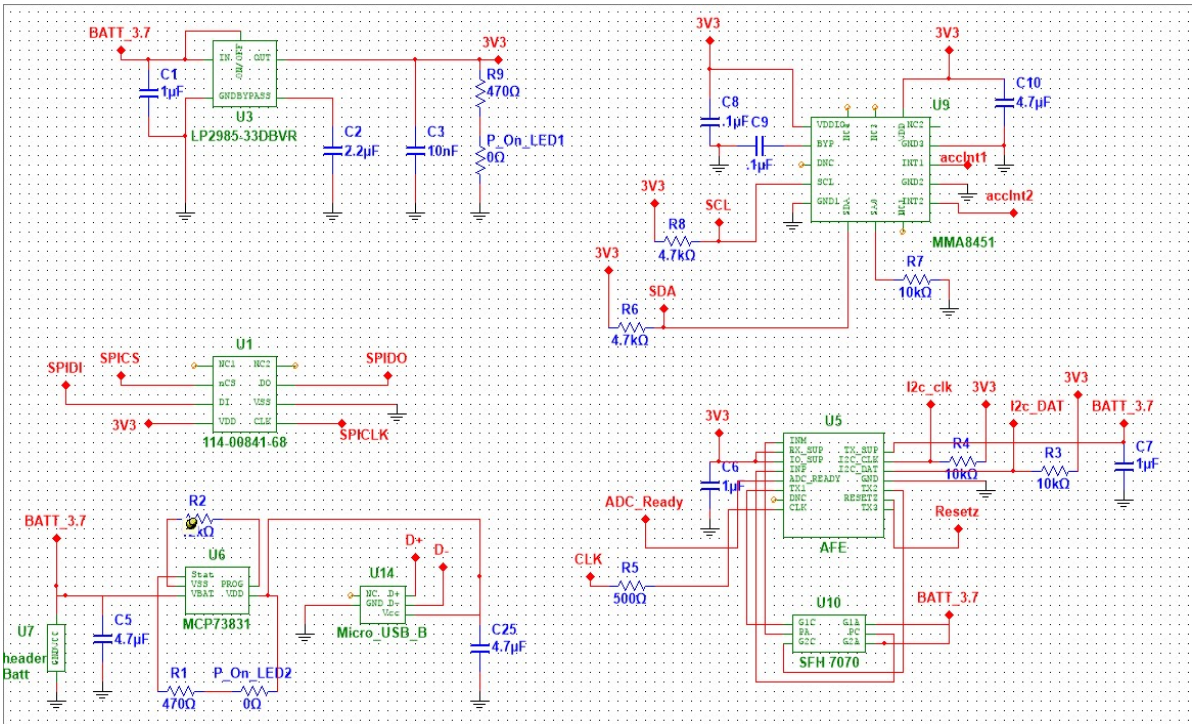

Figure 17 Schematic of the health watch modules

Figure 18 shows the circuit design for the MCU (TM4C123GH6PM). This includes the integrated circuit debugger interface (ICDI) used to load and debug software on the MCU, as well as two crystals, a 16.000 MHz crystal to provide the main clock source to the MCU, and a 32.768 kHz crystal to provide a real-time clock. The capacitor banks are for power supply decoupling.

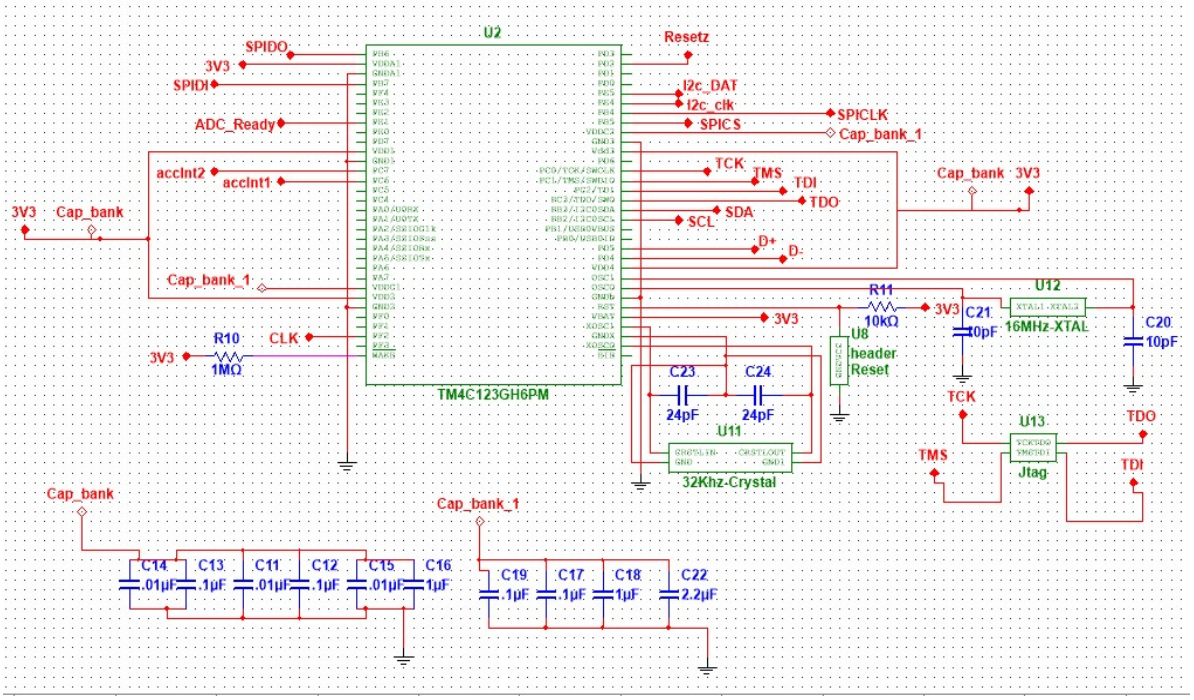

Figure 18 Schematic of the MCU

4.5 PCB design and Layout

The PCB design was performed using National Instruments Ultiboard software, version 14.0. The board consists of two conductor layers, a copper top and a copper bottom. The bottom layer is used mainly as a ground plane. The pads for the SFH 7070 are on the bottom of the board, and all other pads and components lie on the top. The final PCB layout is shown in Figure 19.

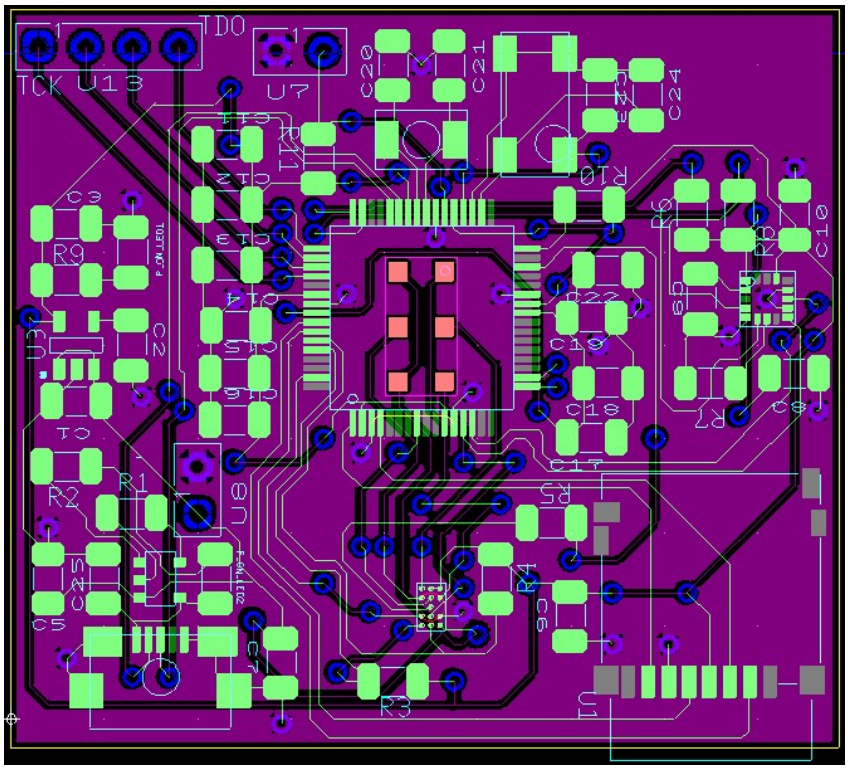

Figure 19. Health watch final PCB layout.

Chapter 5:

5 SIGNAL PROCESSING DESIGN and RESULTS

5.1 Overview

As discussed in 2.5 the PPG signal has AC and DC components that provide the information for heart rate, respiration rate and heart rate variability. Use of the following digital signal processing techniques and algorithms were implemented in the design to create a robust system to correct for false peaks resulting from motion artefacts.

- FIR Digital Filtering
- Kalman Filtering
- Peak and Trough Detection
- Recursive Least Squares Estimation
- Heart Rate Variability Computation
- Respiration Rate Computation

FIR digital filtering, the Kalman filter, peak and trough detection, and the HRV and RR computations are performed in real time and implemented in the health watch firmware. The recursive least squares algorithm was implemented in both firmware and MATLAB. MATLAB is used for post processing to test for an error rate of the signal and to test if it would behoove the design to fully implement it into the systems firmware.

5.2 FIR Digital Filtering

The health watch utilizes digital filtering, which is a very important part of digital signal processing. The use of a filter allows for a signal to be separated and or restored. Separation is used when a signal has been contaminated with interference, or noise. Restoration is needed when a signal has been distorted, and is of poor quality from the actual signal. Both pursue the same purpose, to obtain a clean signal that replicates exactly the one which was sampled, prior to the introduction of noise, interference or distortion. The health watch uses several different types of filtering techniques for separation and restoration. For separation, the health watch uses high pass and low pass filtering, to create a bandpass filter that eliminates any component of the signal outside of its frequency range. For restoration, the health watch uses Kalman filtering and a recursive least squares algorithm to predict and negate unwanted distortion in the signal.

5.2.1 Running Average Filter (LPF)

The running average filter is an averaging filter, which smooths the input data based on previous values. It has very low computational complexity and is well suited for real time processing. It is the most common filter in DSP, due to its ease of understanding and implementation. It is an optimal filter for reducing random noise while retaining a sharp step response. Mathematically the running average filter is a convolution of the input signal with a rectangular pulse of unity area. Figure 20 shows the basic structure of the running average filter, where N is the number of taps, also known as the number of delays or filter order. The greater number of taps, the sharper the filter's response in the frequency domain.

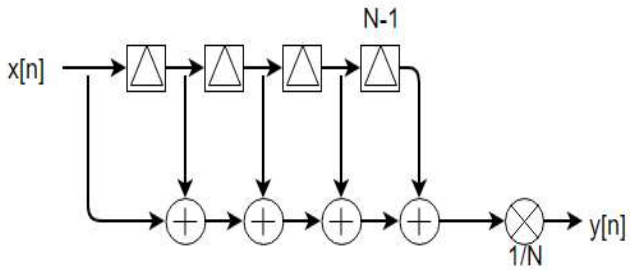

Figure 20. Running average filter diagram.

This filter works well in the time domain, but since it has an infinite impulse response (IIR) and there are N-1 spectral zeroes from 0 to f_s, it performs poor in frequency domain. Equation (3) defines the frequency response of the filter, which is mathematically a sinc function. Figure 21, shows the frequency response at 25 taps (filter order), which was calculated by means of Eq. (3). Here the signal crosses zero at the normalized frequency of .03 Hz. The normalized frequency relates to the actual PPG signal frequency of 3 Hz, which is roughly a 20 BPM heart rate. Equation (2) is the mathematical representation of the filter in the time domain that can be run recursively. Equation (1) is the z-transform of the impulse response of the filter. Equations (1) – (3) describe the filter and its response in the time and frequency domain, respectively.

$$H(z) = \left(1 + z^{-1} + \cdots z^{-(N-1)}\right) * \frac{1}{N} \tag{1}$$

$$y[n] = \frac{1}{M}\sum_{j=0}^{N-1} x[n-j] \tag{2}$$

$$H[f] = \frac{\sin(\pi f N)}{N * \sin(\pi f)},\ where\ f = \frac{f}{fs} \tag{3}$$

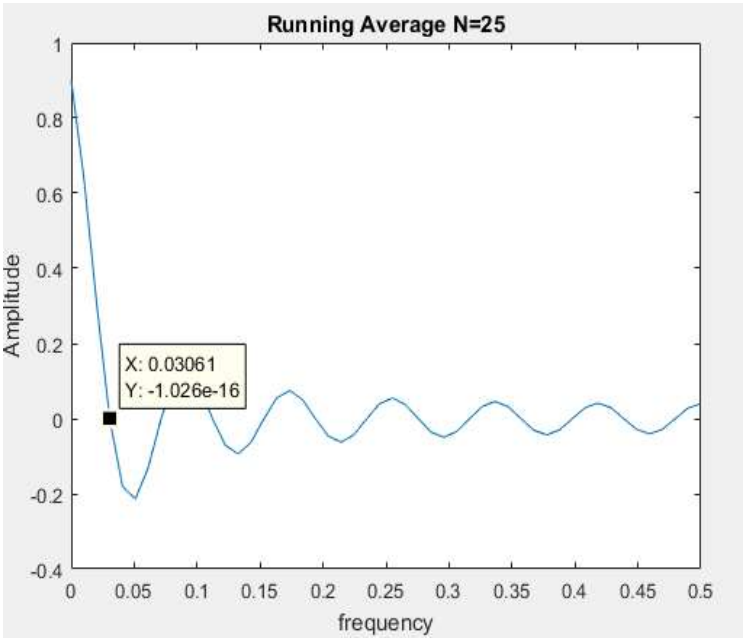

Figure 21 Running average filter frequency response

5.2.2 Difference Filter (HPF)

The system has a DC drift from the electronics that causes the baseline to fluctuate in a random manner. The DC drift introduces a baseline wander of the signal. The baseline wander is due to respiration of the patient and is a desired effect for the measurement. For this reason and for the sake of consistency, a basic high pass filter was designed to remove the drift. As shown in Figure 22, a simple differentiator filter was designed as the HPF. It has only a single tap and two weights (+1, −1) that are summed together to create the output signal.

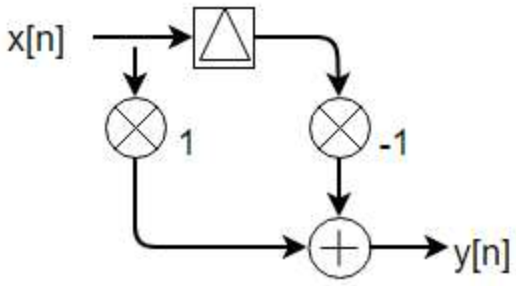

Figure 22 Difference filter diagram

The HPF negates the DC drift, but allows all other important frequencies to be part of the signal. Inputting a constant value, i.e. DC or 0 Hz, will result in no output appearing after an initial transient. Hence there is a spectral zero at 0 Hz where the gain is precisely 0. As shown in the figure, the DC component (the zero crossing) is completely filtered out, leaving just the desired signal. Also, shown in the figure is the unity normalized frequency, represented in angular frequency (ω). This corresponds to a frequency of approximately 0.23 Hz (260 BPM) and is well within the range of preserving the signal. Equation (6) describes the frequency response, which is shown in Figure 23. Equation (4) is the mathematical representation of the filter in the time domain that can be run recursively. Equation (5) is the z-transform of the impulse response of the filter. Equations (4) – (6) describe the filter and its response in the time and frequency domain, respectively.

$$y[n] = x[n] - x[n-1] \tag{4}$$

$$H(z) = 1 - z^{-1} \tag{5}$$

$$H\left(e^{jw}\right) = 1 - e^{-jw} \tag{6}$$

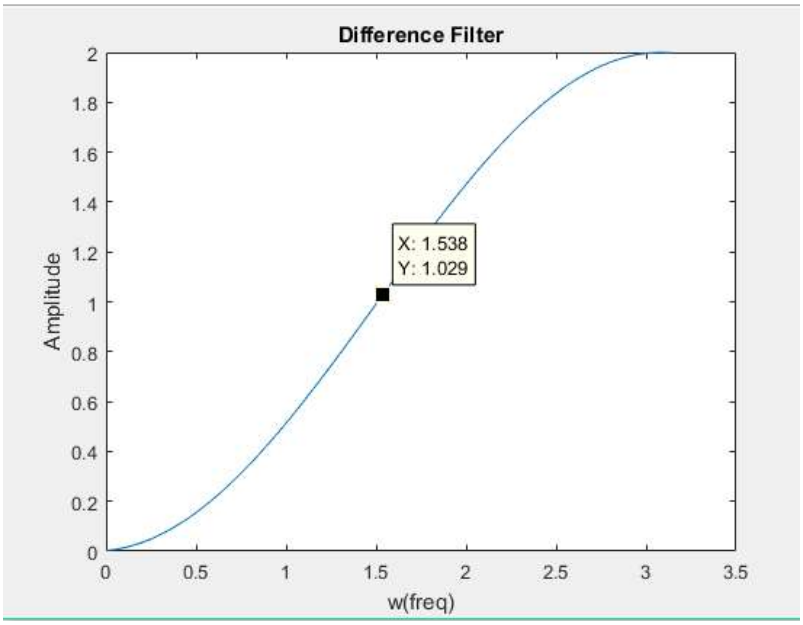

Figure 23 Difference filter frequency response

5.2.3 Cascaded Filters

Figure 24 - 25 represent the overall filter order and design. The HPF is cascaded with three LPF filters, creating a bandpass filter response. Three LPF filters are cascaded to improve the frequency response by lowering the side lobes and harmonic effects of the sinc function. When three running average filters are cascaded, the combination begins to approximate a Gaussian filter, by means of the central limit theorem. Gaussian Filters perform well in both the frequency and time domains. The design of a Gaussian filter is important for post processing techniques. Many techniques found in the literature, but not discussed in this thesis, use frequency domain processing to realize metrics embedded in the PPG signal. The Gaussian filter produces a cleaner frequency response, making post processing in the frequency domain possible.

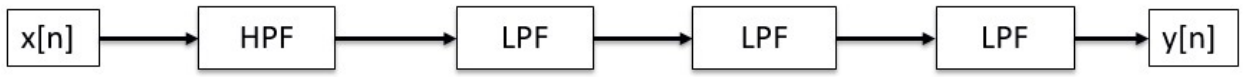

Figure 24 Cascaded filters block diagram

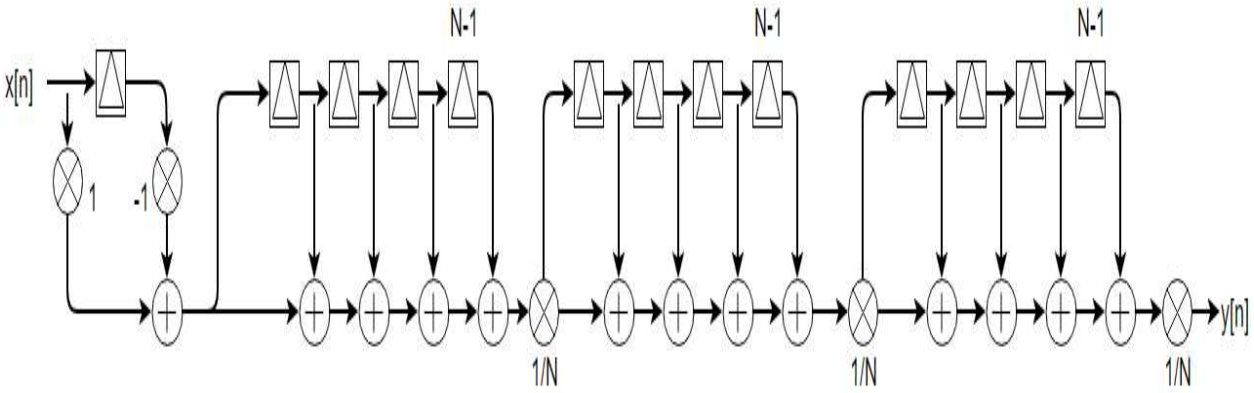

Figure 25 Cascaded filters diagram

5.2.4 Filtering Results

The measured results from the filtering stages are exactly what were expected. Figure 27 shows the signal prior to being filtered, taken directly from the AFE, and Figure 26 shows the signal after being filtered as described in the previous sections. The filtered signal has removed the DC offset, but still contains the baseline wander, allowing further processing to extract the wander from the signal. It is also shown that the signal has been smoothed out and the peaks and troughs are very apparent, which is important for future processing.

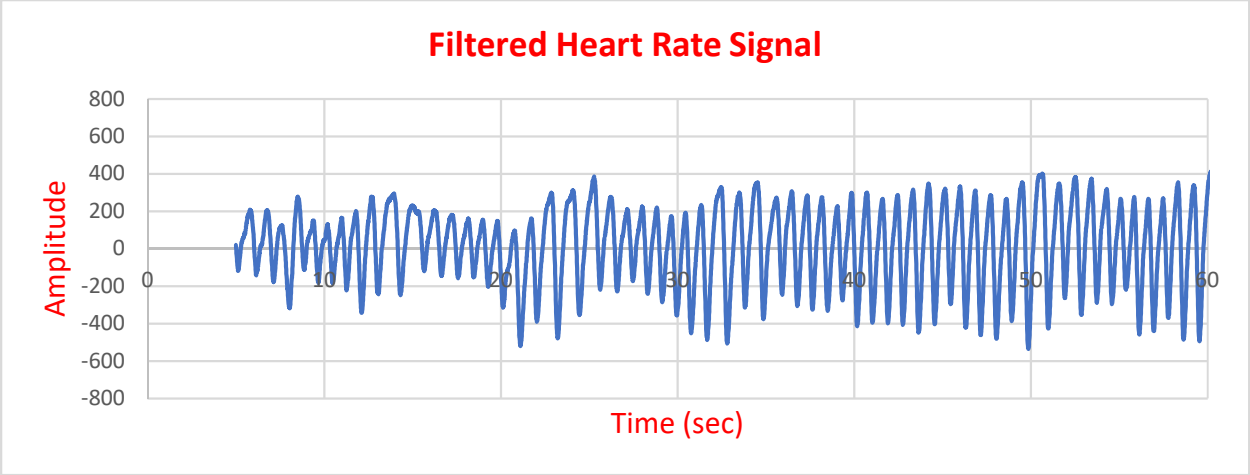

Figure 26 Filtered PPG signal

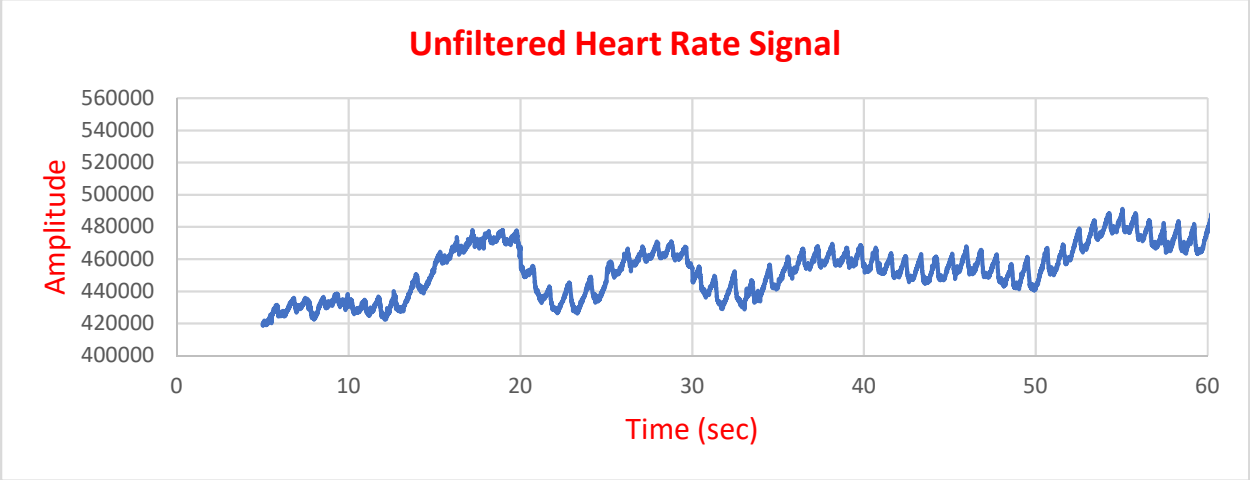

Figure 27 Unfiltered PPG signal

5.3 Kalman Filter

To address signal corruption due to motion artefacts, the Kalman Filter (KF) is used to help track the actual value of the heart rate, and remove outliers and noisy measurements. The KF is an optimal state estimation method that uses a series of noisy measurements observed over time to generate estimates of unknown states. The KF works best when used on dynamic systems where there may be uncertain information, but an educated guess can be made about what the system is going to do next. In the sense of HR, it acts as a dynamic system that

continually changes, and in many scenarios changing in an expected way. HR will increase and decrease gradually, usually over the span of 5 seconds, therefore whenever the HR jumps plus or minus 20 beats or more, it can be assumed to be an outlier. KFs work well on systems that are continuously changing, and they have the advantage of being "light" on memory, making them ideal for real time systems.

The discrete-time Kalman filter is a first-order recursive filter used to estimate the first and second order statistics of a signal in the presence of noise. A Kalman filter combines all available measurement data, plus prior knowledge about the system and the measuring device, to produce an optimal estimate of the desired variables in such a manner that the error is minimized statistically [51].

The Kalman filter model assumes that the state of a system at a time k, evolved from the prior state at time k-1, per the state transition matrix, Eq. (7).

$$x_k = Ax_{k-1} + B\mu_{k-1} + w_{k-1} \tag{7}$$

Measurements or observations of the system can also be performed, per the model

$$z_k = Hx_k + v_k. \tag{8}$$

Table 5 describes the dimensions and descriptions of the KF variables:

Table 5 Kalman Filter variables, dimensions and description

Variable	Dimensions	Description
x_k	Nx1	State Vector
u_k	Lx1	Input/control vector
w_k	Nx1	Process Noise Vector

z_k	Mx1	Observation Vector
v_k	Mx1	Measurement Noise Vector
A_k	NxN	State Transition Vector
B_k	NxL	Input/Control Matrix
H_k	MxN	Observation Matrix
Q_k	NxN	Process Noise Covariance Matrix
R_k	MxM	Measurement noise covariance matrix

The Kalman filter algorithm falls into two parts, the time update equations (or prediction equations) and the measurement update equations, and are explained in the following.

Time update equations:

It is assumed the process can be predicted by (a priori) state estimate in the form

$$\check{x}_k = A\check{x}_{k-1} + B\mu_{k-1}, \tag{9}$$

and the associated predicted (a priori) estimate covariance

$$\check{P}_k = AP_{k-1}A^T + Q. \tag{10}$$

Measurement Updates:

The weighting matrix, also referred to as the Kalman gain is defined as

$$K_k = \check{P}_k H^T \left(H \check{P}_k H^T + R\right)^{-1}. \quad (11)$$

The state estimation is updated (a posteriori) each iteration through, in the form

$$\hat{x}_k = \check{x}_k + K_k(z_k - H\check{x}_k). \quad (12)$$

And the error covariance matrix associated with the updated (a posteriori) estimate is

$$P_k = (I - K_k H)\check{P}_k. \quad (13)$$

Equation (9) represents the a-priori state estimate at the current time step k before a given measurement z_k (PPG signal). Equation (12) is an a-posteriori state estimate from the previous time step after a given measurement z_{k-1} (PPG signal). P_k and P_{k-1} are the error covariance's of a-priori and a-posteriori for each current and previous time step. K_k is the Kalman gain and is fluctuated each step through the algorithm to minimize the a-posteriori error covariance. Small values in the Kalman gain represent a trusted measurement, making the a-posteriori estimated covariance approximately equal to the a-priori estimated covariance. Large values of Kalman gain adjust the a-posteriori covariance, as shown in Eq. (13). The Kalman Filter algorithm is performed recursively with the previous a-posteriori estimates used to predict the new a-priori estimates. Figure 28 shows the recursive loop of the Kalman filter and its updates through time [52]. And an in-depth derivation for the Kalman Filter is presented in [53].

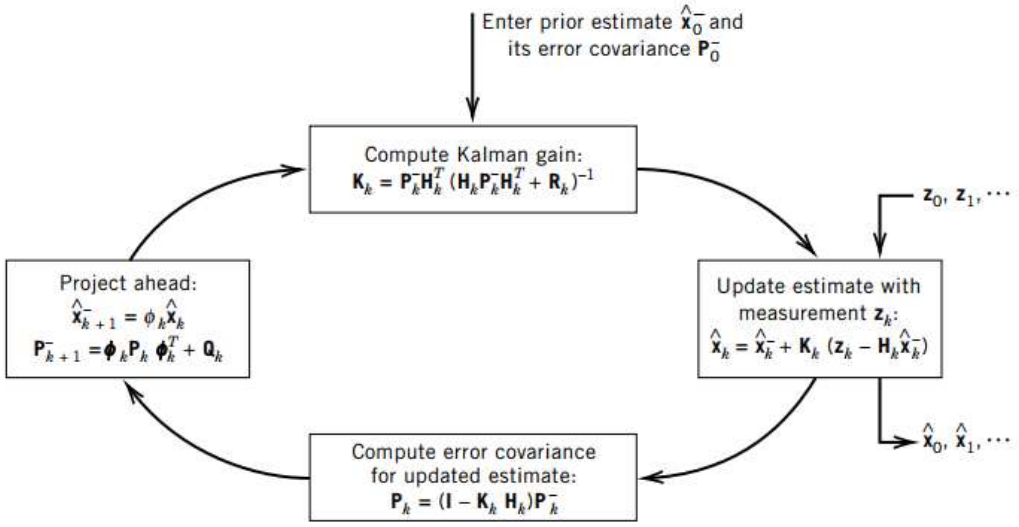

Figure 28 Kalman Filter Recursive Loop, taken from [52].

5.3.1 Kalman Filter Initialization for Heart Rate, and Filter Parameters Q and R

In the health watch the state being estimated is the heart rate of a patient. The heart rate, when averaged over a window of 5 seconds, or more, is approximately constant. Assumptions have been made that the heart rate of an individual, while averaged, should not change drastically between data points. This allows for the process to be modelled as a Markov process, and in this case the state vector is the HR in BPM and is a scalar. Therefore Eqs. (9) - (13), can be redefined as scalar quantities and not matrices, and variances instead of covariance's. Since each HR output is assumed to be approximately equal to the next beat, the state transition parameter A is set to 1. Also, there is no control input to the system making B=0, and the noisy measurements are directly observed setting H=1. The variance P in most literature is set to 10 times Q. R and Q are the only parameters that are detrimental to the filter

working efficiently, all other initialized values will only make the filter converge slower at the worst case.

In practice the measurement noise covariance R is measured by a ground truth device for initialization. But Q is much more difficult to determine since it requires prior knowledge of the process. The parameters chosen for the system were modelled from [54]. [54] measured the noise covariance R to be 41 BPM when the signal was artefact free, and 308 BPM when the signal was corrupted. They also determined that the noise covariance Q to be set at 9 BPM. With these settings, they could reduce HR errors by roughly 75%. Table 6 below lists all the initialized values.

Table 6 Kalman Filter parameters for HR [54]

Parameter	Value
A	1
B	0
H	1
P	90
Q	9
R	41 for clean, 308 for corrupted

5.3.2 Kalman Filter Initialization for Respiration Rate, and Filter Parameters Q and R

Like the heart rate, the respiration rate was estimated to be approximately constant across each breath to breath interval. This is an accurate assumption unless the patient has an illness or disease that produces rapidly varying respiration. As stated in 5.3.1 this allows for the Kalman filter to be described using constants instead of matrices. Like 5.3.1 the initialized parameter values were obtained from literature [54]. The Q value needs to be large enough to track the breathing, but small enough to produce smooth tracking. Q is chosen as an

assumption to the standard deviation of breaths per minute and is shown in the table below. R is much more difficult to determine since it needs to be generic and not patient specific. The R value was determined in [54] by use of a nasal thermistor as its ground truth, and the variance in error was chosen for the R value, shown below in Table 7.

Table 7 Kalman Filter parameters for RR [54]

Parameter	Value
A	1
B	0
H	1
P	20
Q	2
R	31

5.3.3 Kalman Filter Results

Figure 29 shows the results from Kalman filtering of the beat to beat heart rate in BPM. The figure shows how the KF smooths out the data predicting the most probable state form the given a-priori data. The quick fluctuations in the signal are smoothed, and the signal is more continuous, as would be expected from a HR tachogram.

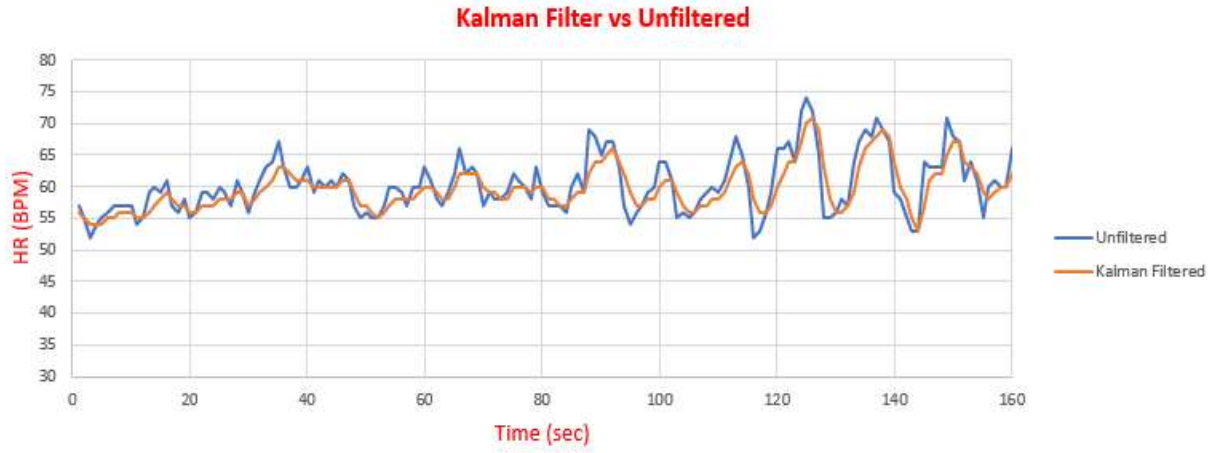

Figure 29 Results for Kalman Filtered HR vs Unfiltered HR

Figure 30 shows the KF output signal tested against a Garmin heart rate fitness tracker that uses a chest strap for HR monitoring. The data taken from the Garmin is only provided in graphical representation, so direct comparisons were not performed. As shown in the figure, the values and time of the recordings match nearly identically. The values are likely to be slightly different between the two systems because of different types of filtering and averaging in each system, but the trends should more or less be the same. The signal was acquired during different activities to test the algorithms robustness as follows:

- Min (1-3) sitting
- Min (3-5) walking
- Min (5-6) eating and quick movements
- Min (6-8) fast paced walking, nearly breaking into a jog
- Min (8-9:30) fast squats
- Min (9:30-11) sitting

In each instance the health watch recorded similar data and did a good job at removing motion artefacts and predicting the correct output.

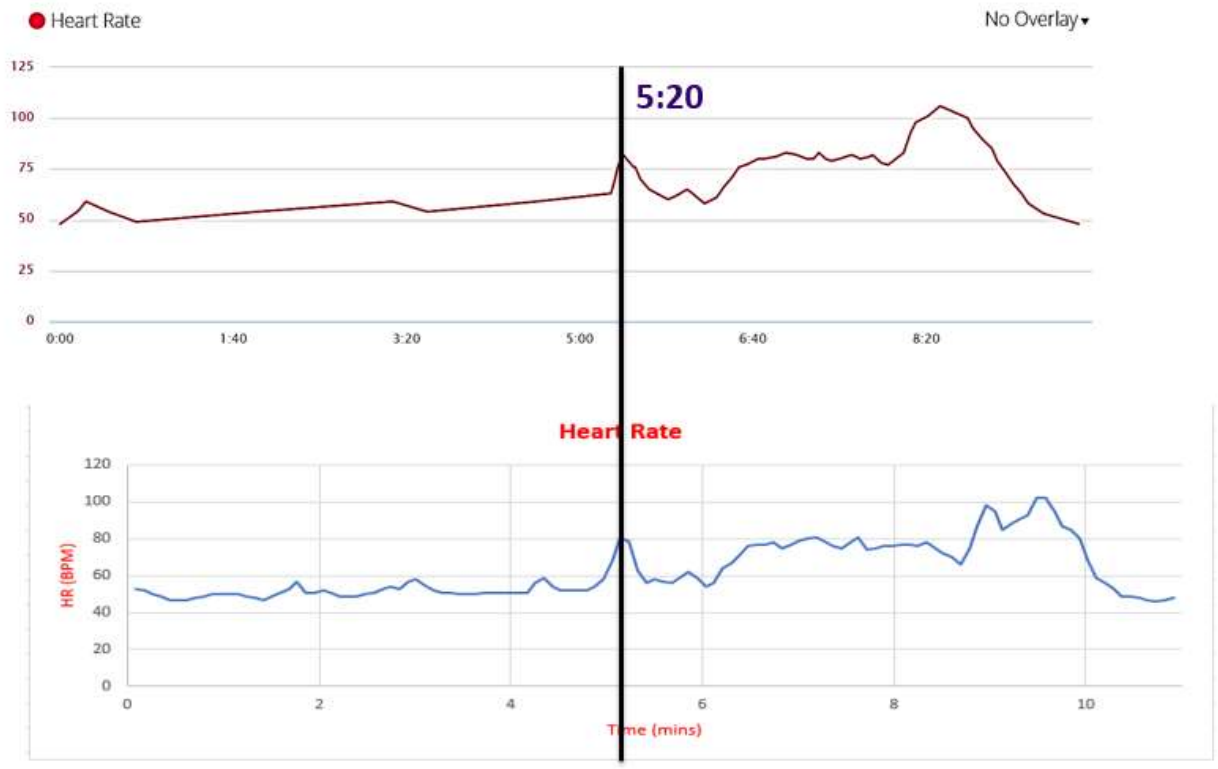

Figure 30 Results of HR in BPM vs Garmin EKG HR monitor

5.4 Recursive Least Squares (RLS)

RLS is an adaptive filter which recursively finds the coefficients that minimize a weighted linear least squares cost function, relating to the input signals. It has an extremely fast convergence, but a high computational complexity. RLS uses two inputs to recursively estimate the future state of the signal. The input signal containing noise is the PPG signal, and the noise reference signal is the accelerometer signal. Because RLS filter uses both signals for prediction, it is an optimal algorithm to help minimize motion artefacts using the signals due to motion. In

cases where motion artefacts are present, their characteristics can be captured from an accelerometer. Equations (14) – (18) describe the RLS algorithm, where x(n) represents the input signal containing noise, and u(n) represents the noise reference, aka the accelerometer data.

The estimation error signal is the difference of the noise reference signal multiplied by a filter coefficient, and the noisy PPG signal, defined as

$$e(n) = x(n) - u^T(n)w(n-1). \tag{14}$$

For computational efficiency in the design, the following parameter is then calculated

$$Pi(n) = P(n-1) * \mathrm{u(n)}. \tag{15}$$

For determination of the filter gain, which creates a weighting for the error signal, a vector update is computed as

$$k(n) = Pi(n) * \left(\lambda + \mathrm{u^T(n)} * \mathrm{Pi(n)}\right)^{-1}. \tag{16}$$

Next, the updated inverse correlation matrix is computed by

$$P(n) = \lambda^{-1}\left(\mathrm{P(n-1)} - \mathrm{k(n)u^T(n)} * \mathrm{P(n-1)}\right). \tag{17}$$

The filter coefficients adaption puts different weights on the noise reference signal each iteration and is defined as

$$w(n) = w(n-1) + k(n)e(n). \tag{18}$$

The recursive least squares algorithm was written both in firmware, as well as MATLAB. In firmware, it took too long for the algorithm efficiently to run because of the matrix multiplication routines. MATLAB is used instead as a post processing tool for the RLS.

5.4.2 Multi Stage RLS with Independent Noisy Channels

A multi-stage adaptive filtering based scheme was constructed. Figure 31 shows the three accelerometer channels being broken up into single x-, y-, and z-components, and the RLS being calculated on each channel in cascade. The purpose of this is because the motion in each unique direction x, y, and z are not highly correlated signals, so they most likely are having a single effect, instead of a net combined effect on the system. Therefore, at each stage the motion artefact due to its unique direction of motion will be removed. In the simplest form one can represent motion artefacts as a combination of three x-, y-, and z-components. This should in theory create a filtered signal, leaving only the clean PPG.

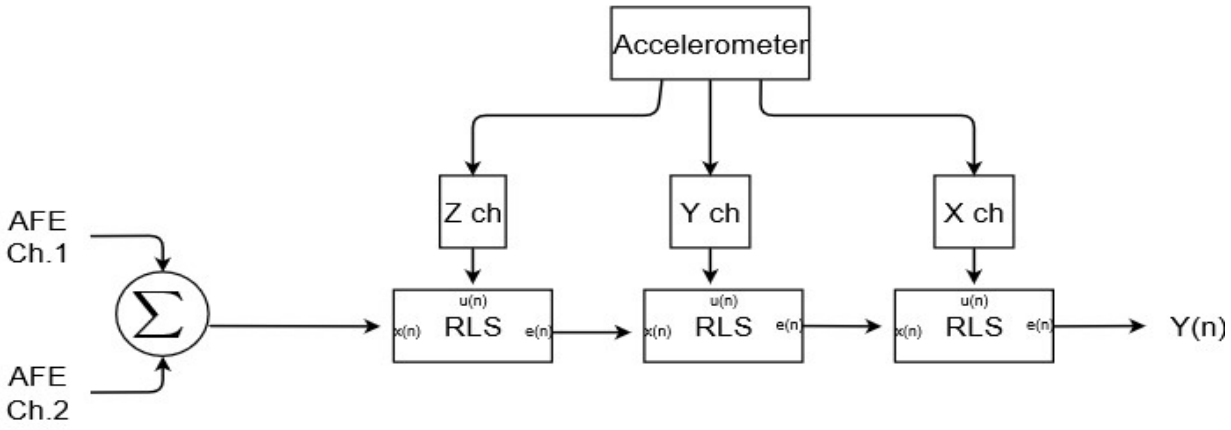

Figure 31 Multi stage RLS with noisy measurements block diagram

5.4.3 Multi Stage RLS with Independent Noisy Channels Results

Figure 32 shows the output in MATLAB of the multi-stage RLS algorithm. The orange signal in the figure represents the filtered data and the blue is the original noisy signal. The filtering takes out unwanted false peaks in the original noisy data. The false peaks are due to motion in the user's arm, and as mentioned before this is described by the accelerometer data for use as a noise reference. This result has great significance for applications. If false peaks in the signal are filtered and properly predicted, then all important health metrics can be computed with great accuracy. This will increase the confidence in the signal quality and the health metrics to accurately predict an individual's state of health.

Currently the algorithm is performed in MATLAB, but future versions may be made to transfer the code into the firmware of the microcontroller.

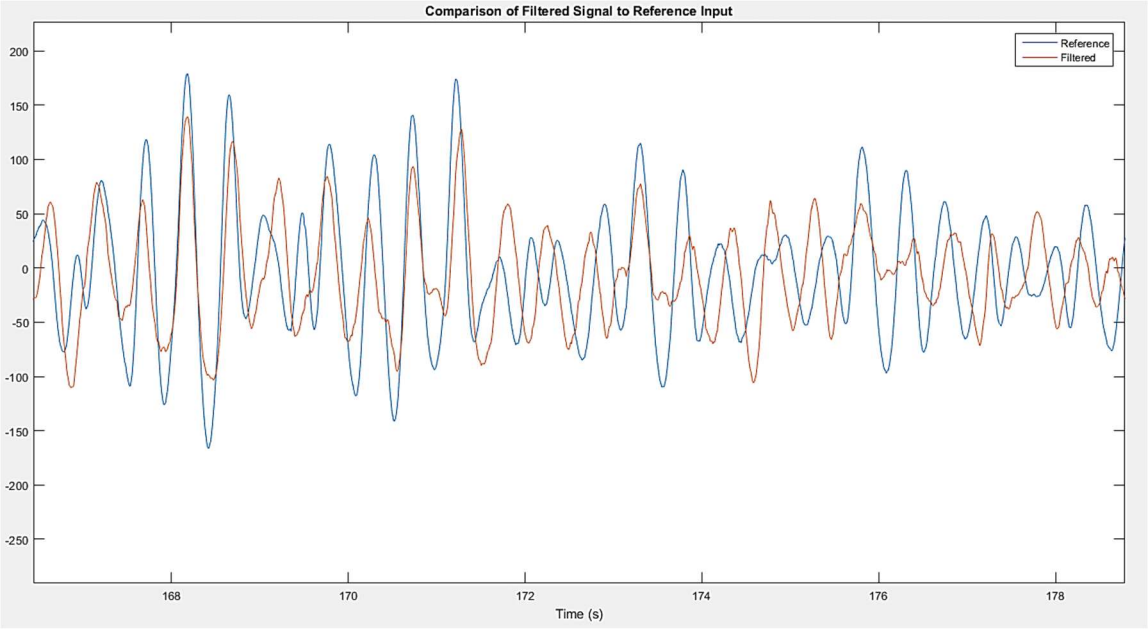

Figure 32 Results of multi stage RLS with noisy measurements from MATLAB

5.5 Peak and Trough Detection

Peak and trough detection has multiple purposes in the design of the health watch. First, it is used to extract the HR by finding the time between systolic peaks, and the associated troughs between each successive peak. It is also used in deriving the respiratory rate from the signal, and this will be discussed more in 5.7. The peak and trough detection algorithm was designed to be robust in determining actual peaks found when the signal is not of constant amplitude or frequency. When different individuals use the health watch, the signal will change due to physical and physiological differences in users. The signal also tends to drift and change shape through time, so the peak detection was designed to have an adaptive step threshold based on the signals prior history.

As the pseudo code below in Figure 33 shows, when the peak or trough is greater or less than the running value, it gets set to the PPG value. Then, a binary bit gets flipped each time a trough or peak is found, and this indicates that the next maximum or minimum must either be a trough or a peak. To determine if a maximum peak or minimum trough has been found, the value is tested against the current peak and the adaptive step threshold. If this is true, the maximum or minimum has been found. The step threshold is found by averaging over several maximum peaks and dividing the average by a set divisor, and this ensures that minor interferences in the signal will not be registered as a peak.

```
Peak_Detect(LED_val, min_max)
{
If LED_val > peak
      Set peak to LED_val
Else if LED_val < trough
      Set Trough to LED_val
Else if last value was either max peak or min trough
      If LED_val < peak-delta
            Max peak found
            Trough is LED_val
      Else if LED_val>trough+delta
            Min trough found
            Peak is LED_val
Return peak & trough
}
```

Figure 33 Peak and trough detect pseudo code

5.4.1 Peak and Trough Detection Results

Figure 34 shows the results from the peak and trough detection algorithm. As shown, the algorithm finds each peak and trough over time. It also indicates its ability to find the peak and trough with a strong baseline wander. Having a properly functional peak and trough detection is vital in the application domain to ensure that there is no peak missed, or no false peak detected. If the peak is missed, all subsequent calculations will be wrong and could provide false information to the user.

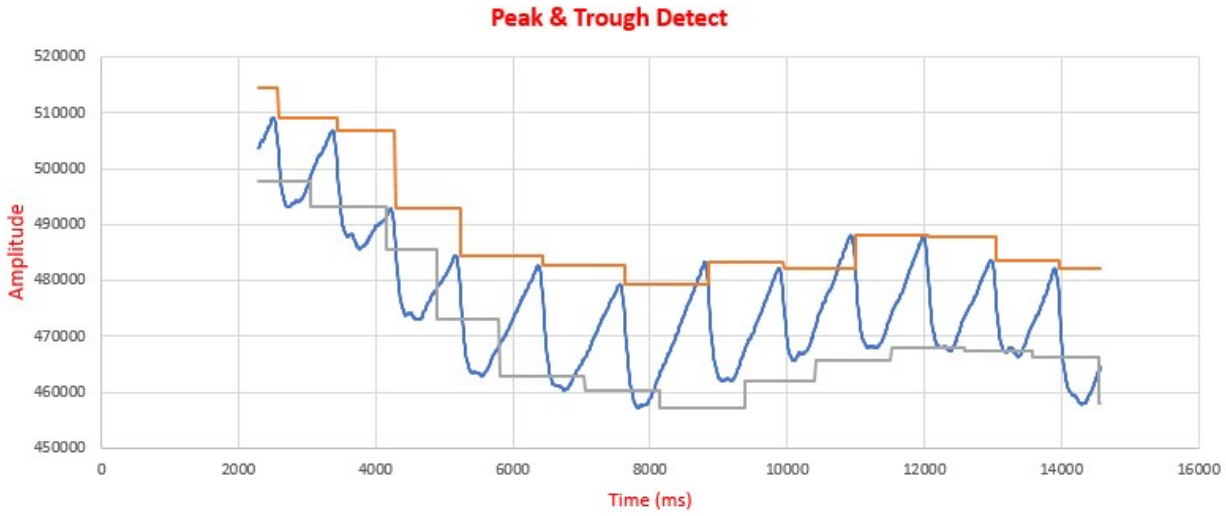

Figure 34 Results from peak and trough detect

5.6 HRV

As discussed in 2.3, there are three main calculations in the time domain to quantify heart rate variability of an individual that meet medical standards set by the *Task Force of The European Society of Cardiology and The North American Society of Pacing and Electrophysiology*. These are SDNN, RMSSD and PNN50. The three calculations are described by Eqs. (19) – (21), where I_n is the nth data sample in the 5-minute N-N vector, and $\breve{I}$ is the mean of the 5-minute vector. N is the size of the 5-minute vector of N-N intervals. In the health watch, five minutes of data are recorded and then the calculations are performed.

$$\boldsymbol{S}DNN = \frac{1}{N}\sum_{0}^{N}\left[I_n - \breve{I}\right]^2 \tag{19}$$

$$RMSSD = \frac{1}{N-1}\sum_{1}^{N}[I_n - I_{n-1}]^2 \tag{20}$$

$$PNN50 = \frac{NN50\ count}{total\ count} \tag{21}$$

5.6.2 HRV Results

The following graphs show the results of the HRV outputs. Figure 35 shows a basic representation of HRV as a HR tachogram. In the figure, the left side of the red bars represent a period of performing arithmetic and reading, creating moments of deep concentration and frustration. In between the two bars represent a period of performing controlled breathing techniques and the end of arithmetic and reading. The right side of the red bars is a period of meditative type deep breathing. The results are what is expected from performing each activity. As discussed in 2.3, HRV is a response to the autonomic nervous system (parasympathetic and sympathetic nerves) which are stimulated differently in each one of these activities. It is found that when controlled and deep breathing ensues in an individual, the results are a quasi-periodic response, as shown in the figures. The quasi periodic response is due to a combination of both RSA and vagal nerve stimulation, as discussed earlier.

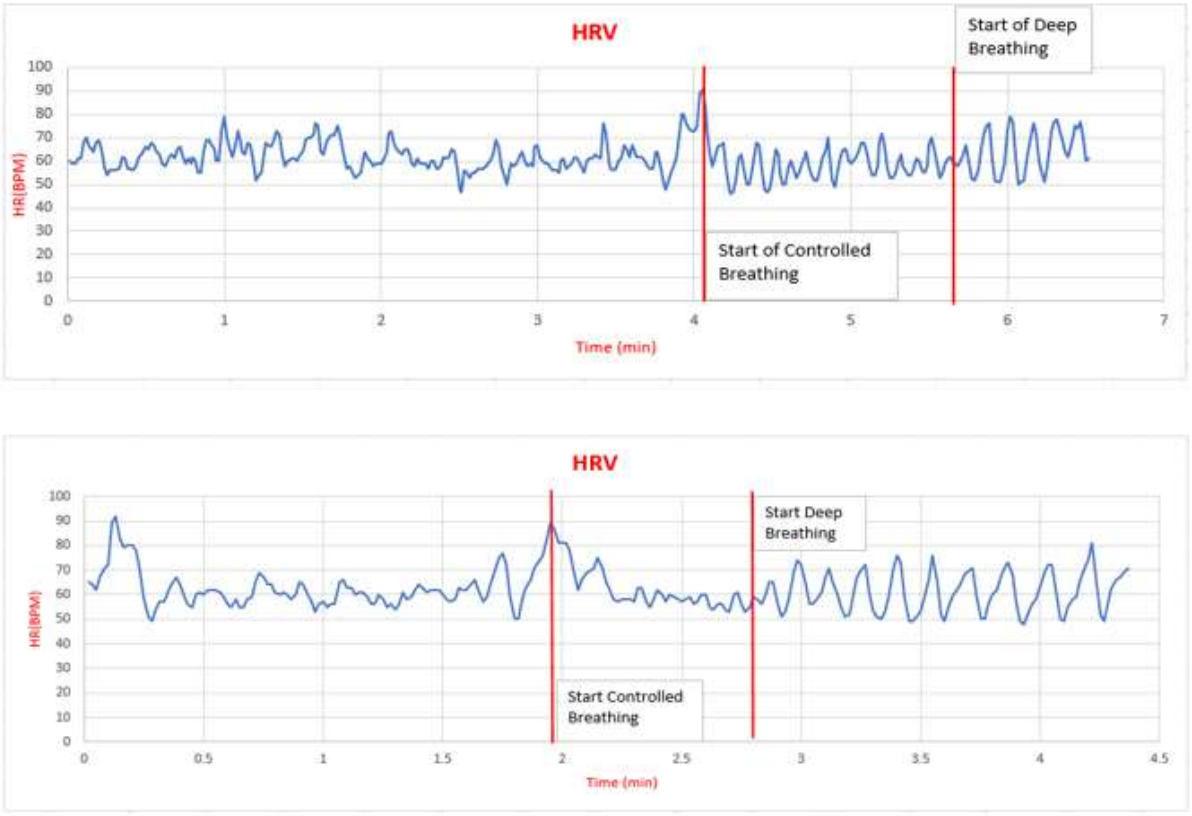

Figure 35 Results HRV, shown as HR tachogram

Figure 36 – 38 show the time domain calculations of SDNN, RMSSD and PNN50. These were each recorded in a similar testing manner as the HR tachograms above. Initially, arithmetic and reading activities were performed followed by moments of relaxation and controlled breathing. For the SDNN and RMSSD measurements, the peak values occurred during activities of lying down and resting. These results as well as the previous show expected responses in the HRV measurements, which indicates accuracy and potential confidence in the system.

Figure 36 Results HRV index (PNN50)

Figure 37 Results SDNN calculations for HRV

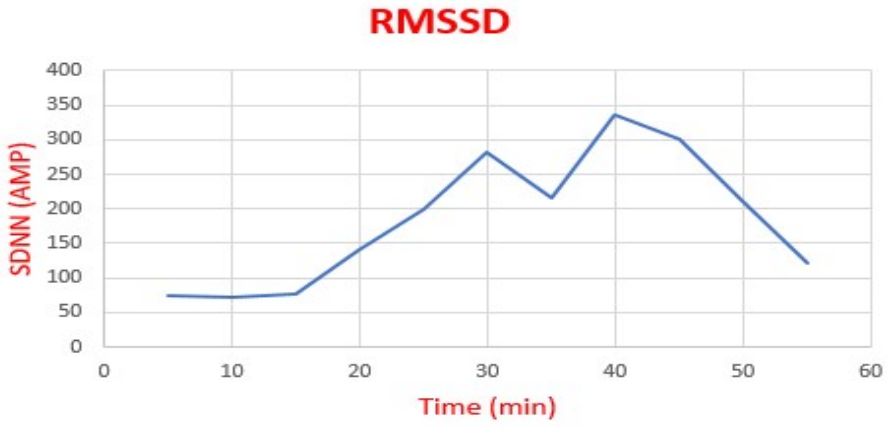

Figure 38 Results RMSSD calculations for HRV

5.7 Respiratory Rate

Respiratory rate can be deduced three ways, from baseline wander, amplitude modulation and frequency modulation. Low-frequency respiratory-induced intensity variations contain the baseline wander. Heart rate modulations, resulting from respiratory sinus arrhythmia, carry frequency modulation. The frequency modulated signal also originates from a vagal heart rate reflex. Respiratory-induced amplitude variations are caused by varying cardiac stroke volume [55].

Baseline wander is calculated in two ways, one from LPF filtering down to 0.3 Hz and the other from averaging the peak and trough signals. Amplitude modulation is calculated from the difference between the trough and peak. Frequency modulation is calculated from the difference in peak time values. Since the health watch calculates four values for the respiration rate, it makes use of all four to find the final rate. After all rates have been recorded, an average is found, and then the rate which has the greatest deviation from the average is subtracted, and the average is then recalculated. This ensures if any value is vastly different it does not have a great weighting on the result. Figure 39 – 41 and Eqs. (22) – (24), show the computations used for the baseline wander, frequency modulation and amplitude modulation techniques.

$$BW = \frac{Peak + Trough}{2} \tag{22}$$

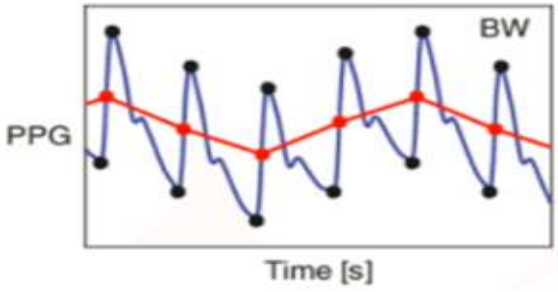

Figure 39 is a representation of the baseline wander that is calculated with (22)

$$FM = NN_n - NN_{n-1} \tag{23}$$

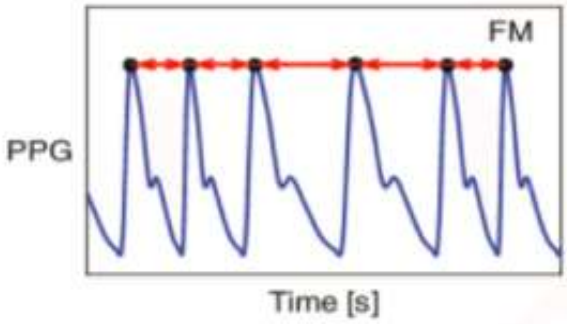

Figure 40 is a representation of the frequency modulation, which by inspection can also be determined by the difference in NN intervals described by RMSSD, and that is calculated with (23).

$$AM = Peak - Trough \tag{24}$$

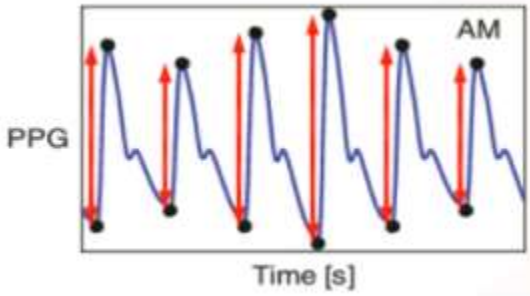

Figure 41 is a representation of the Amplitude Modulation that is calculated with equation (24)

5.7.2 Respiratory Rate Results

Figure 42 shows the results of the respiratory rate signal taken from the health watch compared to a respiratory rate sensor. The respiratory rate sensor signal is in blue and the respiratory signal from the health watch is in orange. In all three instances the peaks of the test sensor and the peaks from the health watch are approximately simultaneous in time. This indicates that all three methods are roughly equivalent in the determination of the respiration rate.

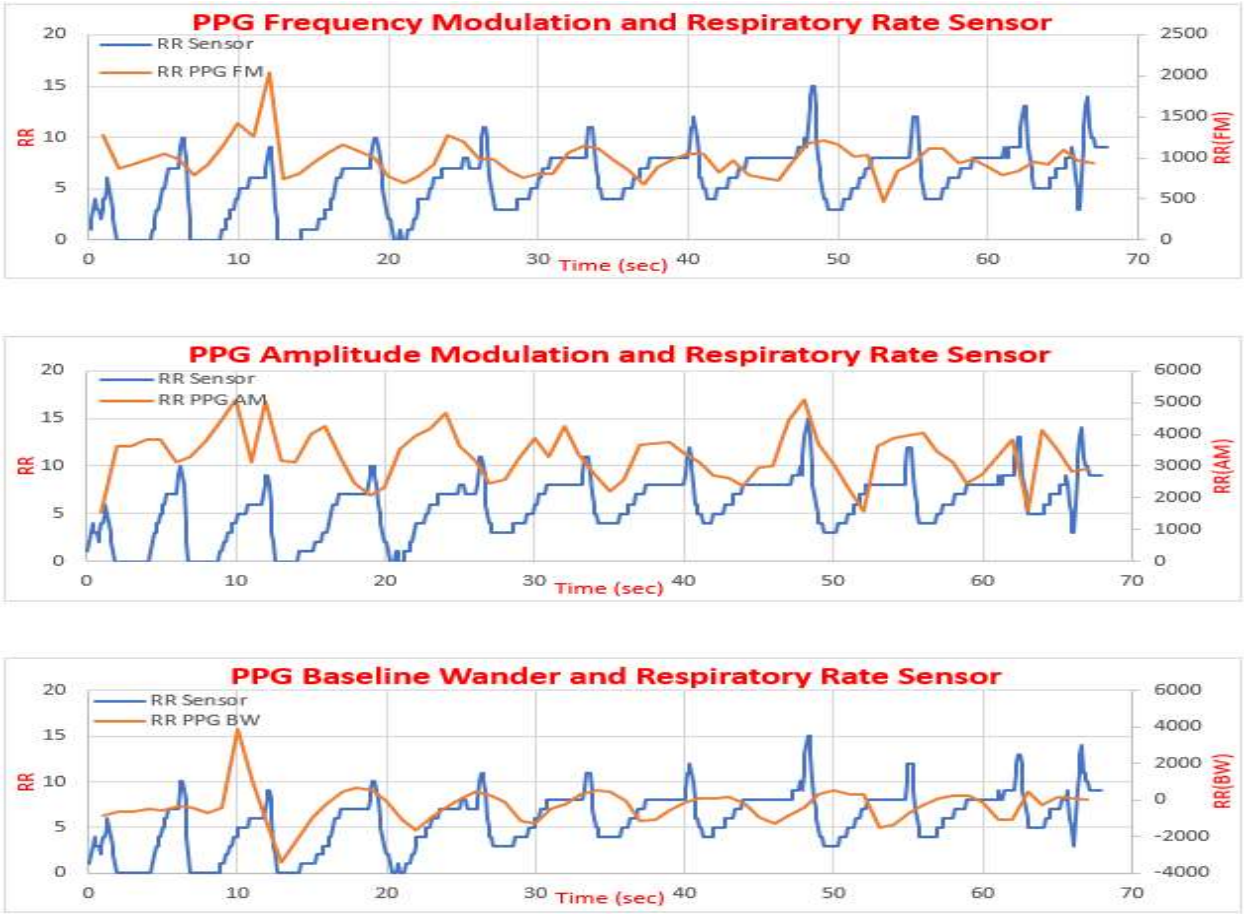

Figure 42 Results from health watch respiration rate using baseline wander, frequency modulation and amplitude modulation vs respiratory rate sensor

Figure 43 shows the relationship between respiratory sinus arrhythmia and heart rate variability. The amplitude of the RSA signal is shown to increase with an increase of HRV and the heart rate. This is because they are linked through vagal stimulation as discussed earlier.

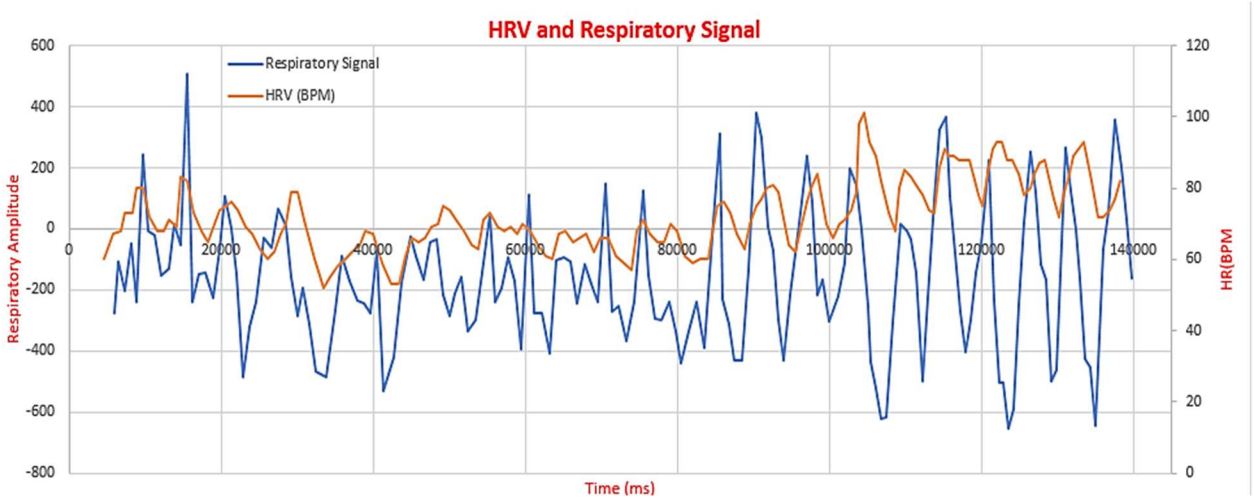

Figure 43 Results of HRV and RSA, showing the correlation between the two signals

Chapter 6:

6 DISCUSSION

Since the health watch adheres to medical standards set by *Task Force of The European Society of Cardiology and The North American Society of Pacing and Electrophysiology* it has the potential to be used as a medical device administered by physicians and doctors. If it can be adopted by the industry it has a huge potential in increasing the gap between patient and doctor relationships. This could improve healthcare efficiency, and potentially aid in reducing health care costs by reducing stress linked illness. It also will bring awareness to stress, which now is a neglected killer, potentially saving lives.

The results from the system indicate that the design goals of tracking HRV, HR and RR have been met, although the overall system accuracy has not yet been tested. For the health watch to advance to the point to become a clinical device for use by medical professionals, it must be further tested and statistically compared to ground truth predicate devices through clinical trials.

To aid in increasing the system accuracy, motion artefacts must be mitigated. As shown in the results, the RLS algorithm shows promise in estimating the real signal, and removing these false peaks. Therefore, the RLS algorithm needs to be more fully implemented in firmware for use in real time. Once this is done and a confidence in the accuracy of the signal has been established, a clinical study must be held to obtain data that supports the claim for the health watch to become a medical device.

REFERENCES

[1] Statistic Brain, "Stress Statistics," 18 May 2017. [Online]. Available: http://www.statisticbrain.com/stress-statistics.

[2] Cleveland Clinic, "Stress and Physical Health," 2014. [Online]. Available: https://my.clevelandclinic.org/health/articles/stress-physical-health.

[3] K. L. Chambers, "Obesity: An American Epidemic," May 2009. [Online]. Available: http://www.ast.org/pdf/305.pdf.

[4] J. Dennis Thompson, "The Link Between Stress and Obesity," 4 Aug 2009. [Online]. Available: https://www.everydayhealth.com/diet-nutrition/food-and-mood/stress-and-dieting/stress-and-other-causes-of-obesity.aspx.

[5] APA, "Latest APA Survey Reveals Deepening Concerns About Connection Between Chronic Disease and Stress," 11 Jan 2012. [Online]. Available: http://www.apa.org/news/press/releases/2012/01/chronic-disease.aspx.

[6] F. HANSEN, "Bad Stress vs. Good Stress," 29 March 2015. [Online]. Available: https://adrenalfatiguesolution.com/bad-stress-vs-good-stress/.

[7] G. Vijay, "How Machine Learning and Big Data Will Change the Future of Medicine," 23 Sep 2016. [Online]. Available: http://nhsjs.com/2016/how-machine-learning-and-big-data-will-change-the-future-of-medicine/.

[8] Farlex, "The Free Dictionary: Systol and Diastole," 2017. [Online]. Available: http://medical-dictionary.thefreedictionary.com/Systole+and+Diastole.

[9] Universal Health Solutions, "Features of the pulse Waveform," 2012. [Online]. Available: http://www.uhsmed.com/bp-management/features-of-the-pulse-waveform.php.

[10] HeartMath Institute, "Science of the Heart," 2017. [Online]. Available: https://www.heartmath.org/research/science-of-the-heart/heart-brain-communication/.

[11] O. Camero, "Visceral Sensory Neuroscience: Interception," Oxford University Press, New York, 2002.

[12] D. C. Randall, "Interactions within the intrinsic cardiac nervous system contribute to chronotropic regulation," *American Journal of Physiology - Regulatory, Integrative and Comparative Physiology,* vol. 285, no. 5, pp. R1066-R1075, 2003.

[13] CDC, "Heart Disease Facts," 10 Aug 2015. [Online]. Available: https://www.cdc.gov/heartdisease/facts.htm.

[14] C. Nordqvist, "Heart Disease: Definition, Causes, Research," 7 April 2016. [Online]. Available: http://www.medicalnewstoday.com/articles/237191.php.

[15] NIH, "What Is Coronary Heart Disease?," 22 June 2016. [Online]. Available: https://www.nhlbi.nih.gov/health/health-topics/topics/cad.

[16] S. P. Edwards, "The Amygdala: The Body's Alarm Circuit," May 2005. [Online]. Available: http://www.dana.org/Publications/Brainwork/Details.aspx?id=43615.

[17] B. L. Seaward, "Physiology of Stress," Jones and Bartlett, Boulder, 2006.

[18] T. Taylor, "Endocrine System," 2017. [Online]. Available: http://www.innerbody.com/image/endoov.html.

[19] M. Randall, "The Physiology of Stress: Cortisol and the Hypothalamic-Pituitary-Adrenal Axis," *Dartmouth Undergraduate Journal of Science,* 2011.

[20] P. Low, "Overview of the Autonomic Nervous System," [Online]. Available: http://www.merckmanuals.com/home/brain,-spinal-cord,-and-nerve-disorders/autonomic-nervous-system-disorders/overview-of-the-autonomic-nervous-system.

[21] K. A. Zimmermann, "Endocrine System: Facts, Functions and Diseases," 11 March 2016. [Online]. Available: https://www.livescience.com/26496-endocrine-system.html.

[22] C. Bergland, "Cortisol: Why "The Stress Hormone" Is Public Enemy No. 1," 22 Jan 2013. [Online]. Available: https://www.psychologytoday.com/blog/the-athletes-way/201301/cortisol-why-the-stress-hormone-is-public-enemy-no-1.

[23] John Hopkins Medicine, "Adrenal Glands," [Online]. Available: http://www.hopkinsmedicine.org/healthlibrary/conditions/endocrinology/adrenal_glands_85,p00399/.

[24] S. Porges, "The Polyvagal Perspective," *Biol Psychol,* vol. 74, no. 2, pp. 116-43, 2007.

[25] D. P. Rosch, "Stress and Heart Disease," AIS, [Online]. Available: https://www.stress.org/stress-and-heart-disease/.

[26] A. Rosengren, "Association of psychosocial risk factors with risk of acute myocardial infarction in 11 119 cases and 13 648 controls from 52 countries (the INTERHEART study): case-control study," *The Lancet,* vol. 364, no. 9438, pp. 953-962, 2004.

[27] P. Karen A. Matthews and P. M. Brooks B. Gump, "Chronic Work Stress and Marital Dissolution Increase Risk of Posttrial Mortality in Men From the Multiple Risk Factor Intervention Trial," *Arch Intern Med,* pp. 309-315, 2002.

[28] APA, "2015 Stress in America," APA, 2015. [Online]. Available: http://www.apa.org/news/press/releases/stress/2015/snapshot.aspx.

[29] C. A. Malik M, "Heart Rate Variability," Futura Pub Co Inc, Armonk, 1995.

[30] L. S. Hon E.H, "Electronic evaluation of the fetal heart rate patterns preceding fetal death," *Am J Obstet Gynecol,* vol. 87, pp. 814-826, 1965.

[31] H. Braune, "Measurement of heart rate variations: influencing factors, normal values and diagnostic impact on diabetic autonomic neuropathy," *Diabetes Clin Pract,* vol. 29, no. 3, pp. 179-187, 1995.

[32] M. Wolf, "Sinus arrythmia in acute mycardial infarction," *Medical Journal of Australia ,* vol. 2, pp. 52-53, 1978.

[33] NARC, "The Bainbridge Reflex," NARC4U, [Online]. Available: http://www.narc4u.com/you-asked-we-answered/72-the-bainbridge-reflex.

[34] Wikipedia, "Baroreflex," 22 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Baroreflex.

[35] R. D. Miller, "Millers Anesthesia," *Anesthesia,* vol. 1 & 2, no. 7, p. 409, 2009.

[36] J. Hayano, "Respiratory Sinus Arrhythmia," *American Heart Association,* pp. 842-847, 1996.

[37] M. Rovere, "Baroreflex sensitivity and heart-rate variability in prediction of total cardiac mortality after myocardial infarction," *The Lancet,* vol. 351, no. 9101, pp. 478-484, 1998.

[38] S. Wegerif, "What Exactly is the Influence of Breathing Rate and Depth on HRV?," ithlete, 26 Nov 2015. [Online]. Available: http://www.myithlete.com/47426-2/.

[39] Wikipedia, "Circadian rhythm," Wikipedia, July 2017. [Online]. Available: https://en.wikipedia.org/wiki/Circadian_rhythm.

[40] M. Nakagawa, "Circadian rhythm of the signal averaged," *Heart,* vol. 79, pp. 493-496, 1998.

[41] F. S. Rollin McCraty, "Heart Rate Variability: New Perspectives on Physiological Mechanisms, Assessment of Self-regulatory Capacity, and Health risk," *GLOBAL ADVANCES IN HEALTH AND MEDICINE,* vol. 4, no. 1, pp. 46-61, 2015.

[42] Task Force of The European Society of Cardiology and The North American, "Heart rate variability Standards of measurement, physiological interpretation, and clinical use," *European Heart Journal,* vol. 17, pp. 354-381, 1996.

[43] Vizbara, "Comparison of green, blue and infrared light in wrist," in *Biomedical Engineering*, Lithuania.

[44] T. Tamura, "Wearable Photoplethysmographic Sensors—Past and Present," *Electronics,* vol. 3, pp. 282-302, 2014.

[45] T. I. Incorporated, *Tiva™ TM4C123GH6PM Microcontroller,* Texas Instruments Incorporated, 2015.

[46] T. I. Incorporated, *AFE4404 Ultra-Small, Integrated AFE for Wearable, Optical, Heart-Rate Monitoring and Bio-Sensing,* Texas Instruments Incorporated, 2016.

[47] R. Batteries, *Rechargeable Lithium Ion Polymer Battery Pack 3.7 V with Safety Circuit,* Renata Batteries.

[48] M. T. inc, *Miniature Single Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controller,* Microchip Technology inc, 2005.

[49] OSRAM, *SFH 7070,* OSRAM Opto Semiconductors GmbH, 2016.

[50] N. Semiconductors, *MMA8451Q, 3-axis, 14-bit/8-bit,* 2017.

[51] P. S. Maybeck, "Stochastic Models, Estimation and Control," Academic Press Inc, 1979.

[52] R. G. Brown, Introduction to Random Signals and Applied Kalman Filtering, John Wiley & Sons, Inc., 2011.

[53] G. A. Terejanu, "Discrete Kalman Filter Tutorial," Buffalo, Department of Computer Science and Engineering.

[54] H.-P. M. Wei-Jheng Lin, "A physiological Information Extraction Method Based on Wearable PPG Sensors wit Motion Artifact Removal," IEEE, 2016.

[55] D. J. Meredith, "Photoplethysmographic derivation of respiratory rate: a review of relevant physiology," *Journal of Medical Engineering & Technology,* vol. 36, no. 1, pp. 1-7, 2012.

[56] G. Vijay, "How Machine Learning and Big Data Will Change the Future of Medicine," 23 Sep 2016. [Online]. Available: http://nhsjs.com/2016/how-machine-learning-and-big-data-will-change-the-future-of-medicine/.

APPENDICES

Appendix A: C code for TM4C123GH6PM

Appendix A contains the entirety of the C code developed for the heath watch. This includes the main() program, AFE4404 initialization and setup, the accelerometer initialization and setup, I2C and algorithmic code.

- **The following code is the main program that runs the health watch, with system and interrupt initializations.**

```
//This code is the main() function for the Health Watch
//Include needed libraries
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdbool.h>
#include <float.h>
#include <stdlib.h>
#include "rom.h"
#include "sysctl.h"
#include <string.h>
#include "hw_memmap.h"
#include "gpio.h"
#include "interrupt.h"
#include "pin_map.h"
#include "timer.h"
#include "hw_ints.h"
#include "TM4C123GH6PM.h"
#include "pwm.h"
#include "i2c.h"
#include "fpu.h"
#include "AFE4404_REG_SETUP_2.h"
#include "rom_map.h"
#include "hw_types.h"
#include "uart.h"
#include "hibernate.h"
#include "uartstdio.h"
#include "Algos.h"
#include "fatfs/src/ff.h"
#include "fatfs/src/diskio.h"

#ifndef max
#define max(a,b) ((a) > (b) ? (a) : (b))
#endif
```

```
	#define data_length 500 //Sample rate is 100Hz, so this will be 5
seconds of data each run
	//#define p_size 10
	//#define N_ 250 //use for repiratory rate. To get filter length
	// and frequency from: Fco=.442947/sqrt(N^2-1),
N=sqrt(.196202+Fco^2)/Fco. Respiratory rate lies between .2-.33. F=f/fs
	#define N_ 100//.003Hz
	#define N 25 //use for HR  normal is 1Hz
	#define row 4
	#define HRV_length 60
	//FATFS definitions
	FRESULT fr;
	FATFS fatfs;
	FIL fil;

	//**********Constants*********************
	unsigned long long sec=0;
	unsigned long long subsec=0;
	int j=0;
	int s=0;
	int n=0;
	bool start=false;
	static signed int hrv[4]={0};
	int ind=0;
	float NBFS=1;
	float PBFS=1;
	static signed int V[2]={0};
	float CI=0;
	short m;
	short ma;
	static signed int filt_sig[5][data_length]={0};
	typedef struct RR{
		signed int BW;
		signed int FM;
		signed int AM;
		signed int peak;
		signed int time;
		short counter;
		signed int rate;
		signed int trough;
		signed int average;
	}respiratory;

	//respiratory RRpeak,RRp_old,r_,R_,rate,t_,RR_BW,RR_AM,RR_FM;
	respiratory RR_={0,0,0,0,0,0,10,0,0};
	respiratory RR_old={0,0,0,0,0,0,10,0};
	respiratory rr_={0,0,0,0,0,0,10,0};
	respiratory rr_old={0,0,0,0,0,0,10,0};
	respiratory RR_1={0,0,0,0,0,0,10,0,0};
	respiratory RR_old1={0,0,0,0,0,0,10,0};
	respiratory rr_1={0,0,0,0,0,0,10,0};
	respiratory rr_old1={0,0,0,0,0,0,10,0};
	respiratory RR_2={0,0,0,0,0,0,10,0,0};
	respiratory RR_old2={0,0,0,0,0,0,10,0};
	respiratory rr_2={0,0,0,0,0,0,10,0};
	respiratory rr_old2={0,0,0,0,0,0,10,0};
```

```
typedef struct HR{
      signed int peak;
      signed int trough;
      signed int time;
      short counter;
      signed int rate;
      signed int BPM;
      signed int average;
}HEART;
HEART HR_={10,0,0,0,60,60};
HEART HR_OLD={10,0,4000,0,60,60};
HEART HR={10,0,0,0,60,60,0};
HEART HR_old={10,0,0,0,60,60,0};

int mean=0;
signed int HRV_STATS[2][HRV_length]={0};
int SDNN_COUNT=0;
float SDNN_VALUE=0;
float RMSSD=0;
signed int HR_M[row][data_length]={0};
signed char ACC[3][data_length]={0};
signed int HRV;
static int i=0;
int value_sum=0,value_sum_=0,val_sum=0;
int rvalue_sum=0,rvalue_sum_=0,rval_sum=0;
int filter[4][N]={0};
int resp_filter[4][N_]={0};
int Index=0,Index_=0,Index_3=0;;
int resp_Index=0,resp_Index_=0,resp_Index_3=0;
signed int signal=0,signal1=0,signal2=0;
signed int xyz=127;
signed char x=127,y=127,z=127;
signed int LED_Val_1=0;
signed int LED_AMB_Val_1=0;
signed int millis=0;
signed int t_old=0;
//signed int time_t;
unsigned char i2c_buff1[4]={0};
unsigned long ulPeriod;
unsigned char temp_st[4] = { 0 };



//*****************************************

//*************Functions*******************
unsigned char I2CRead(signed int base,unsigned char addr,unsigned char
device_register, unsigned char* data, unsigned char len);
void init_MMA8451_Accelerometer(void);
signed int two_comp_32(uint32_t input,uint8_t num_bits);
void Timer_init(void);
void HR_time(void);
void SetMode(void);
void ACCEL_INT(void);
void ConfigureUART(void);
void ADC_Ready_Int_init(void);
```

```
      void ADC_Ready(void);
      void SetupI2C(void);
      uint8_t AFE4404_init( uint8_t address );
      void HR_calibration_(void);
      //void button(void);
      //void Health_Button_init(void);
      //void memory_init(void);
      //*****************************************


      //*****************************************************************
********
      //
      // Moves to end of file for append operation
      //
      //*****************************************************************
********
      FRESULT open_append (
          FIL* fp,            /* [OUT] File object to create */
          const char* path    /* [IN]  File name to be opened */
      )
      {
          FRESULT fr;

          /* Opens an existing file. If not exist, creates a new file. */
          fr = f_open(fp, path, FA_WRITE | FA_OPEN_ALWAYS);
          if (fr == FR_OK) {
              /* Seek to end of the file to append data */
              fr = f_lseek(fp, f_size(fp));
              if (fr != FR_OK)
                  f_close(fp);
          } else{
            }
          return fr;
      }


      //Conversion of 32 and 8 bit 2's complement

      signed int two_comp_32(uint32_t input,uint8_t num_bits){
      signed      int two_comp;
            signed int shift_in;
      two_comp= input;
            shift_in = num_bits==8? input>>7:input>>23;
            if(shift_in==1){
                  two_comp=    num_bits==8? (signed
char)(~input)+1:~(input&0x1FFFFF)+1;
            return two_comp;
            }
            else{
            return input;
            }
      }
```

```
	//Initialize timers for Interupts. The timer interupt to keep track of
BPM
	void Timer_init(void){


		signed int Period=24000;// every millisecond the timer overflows
		signed int Period1=2400000; //every 10 micro seconds timer
overflows
		SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
		SysCtlDelay(3);
		SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
		SysCtlDelay(3);
		TimerClockSourceSet(TIMER1_BASE,SysCtlClockGet());//clock is from
system
		TimerClockSourceSet(TIMER2_BASE,SysCtlClockGet());//clock is from
system
		TimerDisable(TIMER1_BASE,TIMER_A);
		TimerDisable(TIMER2_BASE,TIMER_A);
		TimerConfigure(TIMER1_BASE,TIMER_CFG_PERIODIC);
		TimerConfigure(TIMER2_BASE,TIMER_CFG_PERIODIC);
		TimerLoadSet(TIMER1_BASE,TIMER_A,Period);//this is for 100 msec
interupt counter
		TimerLoadSet(TIMER2_BASE,TIMER_A,Period1);//this is for 100 msec
interupt counter
		TimerLoadSet(TIMER1_BASE, TIMER_A, Period -1);
		TimerLoadSet(TIMER2_BASE, TIMER_A, Period1 -1);
		TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
		TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
	  TimerIntRegister(TIMER1_BASE, TIMER_A, HR_time);
		TimerIntRegister(TIMER2_BASE, TIMER_A, ACCEL_INT);
		TimerEnable(TIMER1_BASE,TIMER_A);
		TimerEnable(TIMER2_BASE,TIMER_A);
	}

	//Interupt Handler for HR counter
	// Creates counter every millisecond and counter increses
	void HR_time(void)
	{
	  signed int status=0;
	  status = TimerIntStatus(TIMER1_BASE,true);
	  TimerIntClear(TIMER1_BASE,status);
	  millis++;
	}

	//Accelerometer Interupt, check if data is in buffer before reading out
	void ACCEL_INT(void){
		signed int status=0;
		status = TimerIntStatus(TIMER2_BASE,true);
	  TimerIntClear(TIMER2_BASE,status);
		if(I2CRead(I2C0_BASE,0x1D,0x00,i2c_buff1, 1)==255){
		xyz=get_xyz_data();
		x=two_comp_32((uint8_t)(xyz>>16),8);
		y=two_comp_32((uint8_t)(xyz>>8)&0x00FF,8);
		z=two_comp_32((uint8_t)xyz,8);
		}
	}
```

```
	//Interupt Should trigger every 10000 us with PRF=39999
	void ADC_Ready_Int_init(void){
			GPIOIntDisable(GPIO_PORTE_BASE, GPIO_PIN_1);		//
Disable interrupt
		GPIOIntClear(GPIO_PORTE_BASE, GPIO_PIN_1);	// Clear pending
interrupts
		GPIOIntRegister(GPIO_PORTE_BASE, ADC_Ready);	// Register our
handler function
		GPIOIntTypeSet(GPIO_PORTE_BASE, GPIO_PIN_1,GPIO_RISING_EDGE); //
Configure  rising edge trigger
		GPIOIntEnable(GPIO_PORTE_BASE, GPIO_PIN_1);	// Enable interrupt
f
	}
	/* Button to record moments of noticeable stress
	void Health_Button_init(void){
			GPIOIntDisable(GPIO_PORTD_BASE, GPIO_PIN_6);		//
Disable interrupt
		GPIOIntClear(GPIO_PORTD_BASE, GPIO_PIN_6);	// Clear pending
interrupts
		GPIOIntRegister(GPIO_PORTD_BASE, button);	// Register our
handler function
		GPIOIntTypeSet(GPIO_PORTD_BASE, GPIO_PIN_6,GPIO_RISING_EDGE); //
Configure  rising edge trigger
		GPIOIntEnable(GPIO_PORTD_BASE, GPIO_PIN_6);	// Enable interrupt
f
	}

	void button(void){
		 if(GPIOIntStatus(GPIO_PORTD_BASE,false) & GPIO_PIN_6 ){
		 GPIOIntClear(GPIO_PORTD_BASE, GPIO_PIN_6);
	//UARTprintf("Button Pushed\n");
	}
	}
	*/

	void system_setup( void )
	{
			//Set System address
		uint8_t address = 0x58;
			//Start Comm
			 SetupI2C();
			//Enable all GPIO Pins
			 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
			 SysCtlDelay(3);
			 GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_2);
			 GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_6);
			 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
			 SysCtlDelay(3);
			 GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_1);
			 GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_2,0);
			 SysCtlDelay(1000);//delay for roughly 40us
			 GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_2,GPIO_PIN_2);
			 SysCtlDelay(30000);//delay for 1.25us after reset prior to
I2c startup
			//Initialize AFE4404 with all settings
			 AFE4404_init( address);
```

```
}

void PWM_init(){

  //Configure PWM Clock to match system
            SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
            SysCtlDelay(3);
            ulPeriod=SysCtlPWMClockGet();
    // Enable the peripherals used by this program.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        SysCtlDelay(3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);  //The Tiva Launchpad
has two modules (0 and 1). Module 1 covers the LED pins
    SysCtlDelay(3);
    //Configure PF1,PF2,PF3 Pins as PWM
    GPIOPinConfigure(GPIO_PF2_M1PWM6);
        GPIOPinTypePWM(GPIO_PORTF_BASE,GPIO_PIN_2);
    //Configure PWM Options
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_UP_DOWN |
PWM_GEN_MODE_NO_SYNC);
    //Set the Period (expressed in clock ticks)
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 6);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,6/2); //set 20% duty cycle
    // Enable the PWM generator
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);
    // Turn on the Output pins
             PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);

//PWM init end

}

//Configure UART for Serial data. Used for initial testing
/*
void
ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        SysCtlDelay(3);

    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        SysCtlDelay(3);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

```
            //
            //
           // UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
                    UARTClockSourceSet(UART0_BASE, UART_CLOCK_SYSTEM);
                    SysCtlDelay(3);

            //UART_CLOCK_SYSTEM
            // Initialize the UART for console I/O.
            //
            UARTStdioConfig(0, 115200, 24000000);}
      */
      //Calibration Routine
      void HR_calibration_(void){
      /* This should be done at start up
      Set timing for sampling pulse duration ?? this is for power saving
      Set TI Gain to 1M ohm set averaging mode between 4-8
      Set LED to desired lowest current settings
      */
      int T_GAIN=7;
      int current=15; //Current is set to 25mA
      int T_count=0;
      int bit_offset=0;
      int bit_sign=1;
      int bit_offset_2=0;
      int bit_sign_2=1;
      int average=0;
      int non_converge=0;
      int count=0;
      int buff=0;
      //Initially calculate average PPG value
      while (count<50){
            buff+=two_comp_32(hr_get_led1_val(),32);
            count++;
      }
      average=buff/count;
      count=0;
      buff=0;
      current=15;
      //While the PPG value is not within +-1V range for ADC for PPG 1
Channel
      while((abs(average) > 1677720) ){ //2097151=1.2V, 1677720=1.0V
      //If value is not within range after 5 iterations, update AFE settings
            if(non_converge>5){
                        bit_offset++;
                        T_count++;

      hr_set_offdac_settings(0,3,bit_sign,bit_offset,bit_sign,bit_offset,1,7)
;
                        SysCtlDelay(30000);
                        non_converge=0;
            }
            //If IDAC has stepped through all values, flip bit to change sign
            if(bit_offset==7){
                  bit_offset=0;
                  bit_sign=1-bit_sign;

            }
```

```
            //If all IDAC values have stepped through update gain and current
            if(T_count==30 && T_GAIN>0){
              T_GAIN--;
                  current=current-2;
                  current<=1?current=15:current;
               hr_set_tia_gain( false, T_GAIN, 0);
                   hr_set_led_currents( current,current,0);
                   SysCtlDelay(30000);
                   T_count=0;
            }
            T_GAIN==0 ? T_GAIN=7:T_GAIN;

            //Reconfigure PPG value
                  while (count<50){
                        buff+=two_comp_32(hr_get_led1_val(),32);
                        count++;
                  }

            non_converge++;
            average=buff/count;
            count=0;
            buff=0;
      }
      non_converge=0;
      count=0;
      buff=0;
      //check for PPG 2 channel
      while (count<50){
            buff+=two_comp_32(hr_get_led2_val(),32);
            count++;
      }
      average=buff/count;
      while((abs(average) > 1677720) ){ //2097151=1.2V, 1677720=1.0V
            if(non_converge>5){
                  bit_offset_2++;

      hr_set_offdac_settings(bit_sign_2,bit_offset_2,bit_sign,bit_offset,bit_
sign,bit_offset,bit_sign_2,bit_offset_2);
                  SysCtlDelay(30000);
                  non_converge=0;
            }
            if(bit_offset_2==15){
            bit_offset_2=0;
                  bit_sign_2=1-bit_sign_2;

            }
                  while (count<50){
                        buff+=two_comp_32(hr_get_led2_val(),32);
                        count++;
                  }

              non_converge++;
              average=buff/count;
                  count=0;
                  buff=0;

      }
```

```
	}

	//Interupt to read LED Values for each timing routine
	void ADC_Ready(void){
			if(GPIOIntStatus(GPIO_PORTE_BASE,false) & GPIO_PIN_1 ){
		//Tristate Clock during Power Down
		PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, false);
		SysCtlDelay(115);
		PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);

		GPIOIntClear(GPIO_PORTE_BASE, GPIO_PIN_1);
				LED_Val_1=two_comp_32(hr_get_led1_val(),32);
		if(abs(LED_Val_1)> 1677720 ){
		SysCtlReset();
		}

	 else if (n%1==0){                    //downsample using n to save
computation

HR_M[0][i]=(two_comp_32(hr_get_led1_amb1_val(),32)+two_comp_32(hr_get_led2_am
b2_val(),32))/2;
		HR_M[1][i]=millis;//Get Timer time
		HR_M[2][i]=HibernateRTCGet();//Get RTC time
		HR_M[3][i]=HibernateRTCSSGet();//Get RTC time
		ACC[0][i]=x*.0156; //Convert to g's
		ACC[1][i]=y*.0156;
		ACC[2][i]=z*.0156;

		i++;
		n++;
	}
	else{
	n++;
	}
	}
		}



	//****************Start Program*****************************
	int main(void){
		//Lazy stacking for efficiency
	ROM_FPULazyStackingEnable();

	    //Set the clock
	SysCtlClockSet(SYSCTL_SYSDIV_1| SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_8MHZ);;  // SET SYSTEM CLOCK FOR 24Mhz, off of PLL

	SysCtlDelay(3);

	   //enable the hibernation module for RTC
	  SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);

	  //SYSCTL_OSC_EXT32 is the rate of the clock supplied to the
Hibernation
	  HibernateEnableExpClk(SYSCTL_OSC_EXT32);
```

```
        HibernateClockConfig(HIBERNATE_OSC_HIGHDRIVE);//Highdrive is 24pF cap
on 32kHz line
         HibernateRTCDisable();

          //set the value to 0
          HibernateRTCSet(0);
          HibernateRTCEnable();

          //get starting values
        sec = HibernateRTCGet();
        subsec =  HibernateRTCSSGet();

      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);

      //SD Card Initialization
      fr = f_mount(&fatfs, "", 0);

      //ConfigureUART();

      //initialize pwm for clock source
      PWM_init();

      //Setup System
      system_setup();

      //IntPrioritySet(INT_GPIOD_TM4C123,3);
      IntPrioritySet(INT_TIMER1A_TM4C123, 1);
      IntPrioritySet(INT_TIMER1B_TM4C123, 2);
      IntPrioritySet(INT_GPIOE_TM4C123,0);

      //Set Accel Mode
      SetMode();

      HR_calibration_();
      ADC_Ready_Int_init();
      Timer_init();
      //Health_Button_init();
      HR_OLD.peak=1000;


          while(1){
            //While the physical activity count is greater than 10, do
nothing to negate possible motion artefacts
      while
(physical_activity_count(two_comp_32(x,8),two_comp_32(y,8),two_comp_32(z,8))>
10){
            TimerIntDisable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
              GPIOIntDisable(GPIO_PORTE_BASE, GPIO_PIN_1);
                  i=i;
            SysCtlDelay(3000000);
            }

                        GPIOIntEnable(GPIO_PORTE_BASE, GPIO_PIN_1);
                        TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

      if(i>=data_length){
            i=0;
```

```
      GPIOIntDisable(GPIO_PORTE_BASE, GPIO_PIN_1);        // Disable
interrupts during algorithm comp
      TimerIntDisable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
      TimerIntDisable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);

            for(j=1;j<data_length;j++){
                  //Start of Filtering, shift out value and shift in new for
running average, for every N values
            value_sum-=filter[0][Index];
          filter[0][Index]=HR_M[0][j]-HR_M[0][j-1]; //Difference HPF Filter
            //filter[0][Index]=(HR_M[0][j]-HR_M[0][j-2])/2; High Pass Filter
with diff frequency response
          value_sum+=filter[0][Index];
          Index++;
          if (Index>=N){
            Index=0;
              }
              filt_sig[0][j]=value_sum/N;
              //2nd order running average
              value_sum_-=filter[2][Index_];
              filter[2][Index_]=filt_sig[0][j];
          value_sum_+=filter[2][Index_];

          Index_++;
          if (Index_>=N){
            Index_=0;
              }
              filt_sig[0][j]=value_sum_/N;

              //3rd order running average, Gaussian Filter
              val_sum-=filter[3][Index_3];
          filter[3][Index_3]=filt_sig[0][j];
          val_sum+=filter[3][Index_3];
        Index_3++;
          if (Index_3>=N){
            Index_3=0;
              }
              filt_sig[0][j]=val_sum/N;


              //respiration Filtering start, to get baseline wander

              rvalue_sum-=resp_filter[0][resp_Index];
          resp_filter[0][resp_Index]=HR_M[0][j]-HR_M[0][j-1];
          rvalue_sum+=resp_filter[0][resp_Index];
          resp_Index++;
          if (resp_Index>=N_){
            resp_Index=0;
              }
              filt_sig[3][j]=rvalue_sum/N_;
              rvalue_sum_-=resp_filter[1][resp_Index_];
               resp_filter[1][resp_Index_]=filt_sig[3][j];
          rvalue_sum_+=resp_filter[1][resp_Index_];
          resp_Index_++;
          if (resp_Index_>=N_){
            resp_Index_=0;
              }
```

```
            filt_sig[3][j]=rvalue_sum_/N_;

            rval_sum-=resp_filter[2][resp_Index_3];
          resp_filter[2][resp_Index_3]=filt_sig[3][j];
          rval_sum+=resp_filter[2][resp_Index_3];
            resp_Index_3++;
          if (resp_Index_3>=N_){
            resp_Index_3=0;
            }
            filt_sig[3][j]=rval_sum/N_;
            }

      if(start==true){  //allows for filtering to get established

      for(j=1;j<data_length;j++){ //begin HR,HRV and RR comp
            //RLS Filtering using constants not matrix mults
            signal=RLS(filt_sig[0][j],ACC[0][j]);
            signal1=RLS(signal,ACC[1][j]);
            signal2=RLS(signal1,ACC[2][j]);

            //Peak detection for HR
      HR.peak=(Peak_Detect((filt_sig[0][j]),1));
      if(HR_OLD.peak!=HR.peak ){
            HR.counter++;
            hrv[ind]=HR_M[1][j];
            ind++;
                  //HRV, SDNN and RMSSD
            if((HR_M[1][j]-t_old)>450&&(HR_M[1][j]-t_old)<2050){
                        HRV_STATS[0][SDNN_COUNT]=HR_M[1][j]-t_old;
                        t_old=HR_M[1][j];
                        mean+=HRV_STATS[0][SDNN_COUNT];
                        SDNN_COUNT++;
                  if(SDNN_COUNT>=HRV_length){
                  mean=mean/SDNN_COUNT;
                        for(s=1;s<HRV_length;s++){
                              SDNN_VALUE+=(HRV_STATS[0][s]-
mean)*(HRV_STATS[0][s]-mean);
                              RMSSD+=(HRV_STATS[0][s]-HRV_STATS[0][s-
1])*(HRV_STATS[0][s]-HRV_STATS[0][s-1]);
                        }
                  SDNN_VALUE=sqrt(SDNN_VALUE/(HRV_length-1));
                  RMSSD=sqrt(RMSSD/(HRV_length-1));
                  SDNN_COUNT=0;
                  mean=0;
                  SDNN_VALUE=0;
                  RMSSD=0;
                  }
            }
                  else{
                  t_old=HR_M[1][j];
                  }
      //PNN50 count for HRV
                        if (ind==4){
                         // HRV=(abs(abs(hrv[0]-hrv[1])-abs(hrv[1]-
hrv[2]))+abs(abs(hrv[2]-hrv[3])-abs(hrv[1]-hrv[2])))/2;
                          abs(hrv[0]-hrv[1])>abs(hrv[1]-hrv[2])?V[0]=1:0;
                              abs(hrv[0]-hrv[1])<abs(hrv[1]-hrv[2])?V[0]=2:0;
```

```
                                abs(hrv[0]-hrv[1])==abs(hrv[1]-
hrv[2])?V[0]=0:0;
                                abs(hrv[1]-hrv[2])>abs(hrv[2]-hrv[3])?V[1]=1:0;
                                abs(hrv[1]-hrv[2])<abs(hrv[2]-hrv[3])?V[1]=2:0;
                                abs(hrv[1]-hrv[2])==abs(hrv[2]-
hrv[3])?V[1]=0:0;
                                V[0]==1 &&V[1]==1?PBFS++:0;
                                V[0]==2 &&V[1]==2?PBFS++:0;
                                V[0]==1 &&V[1]==2?NBFS++:0;
                                V[0]==2 &&V[1]==1?NBFS++:0;
                                V[0]==0 &&V[1]==1?NBFS++:0;
                                V[0]==1 &&V[1]==0?NBFS++:0;
                                V[0]==0 &&V[1]==2?NBFS++:0;
                                V[0]==2 &&V[1]==0?NBFS++:0;
                                V[0]==0 &&V[1]==0?NBFS++:0;
                              ind=0;
                            }
                        HR.average+=HR.peak;
                        HR_old.counter++;
                              //update peak detect steo averaged over 5 beats
                        if(HR_old.counter>5){
                        HR.average=abs(HR.average/HR_old.counter);
                        deltad=HR.average/2;
                        HR.average=0;
                        HR_old.counter=0;
                        }
                        //Update peak detect step
                        deltad=Kalman(deltad,(HR_.rate+HR.rate)/2,1);
                        HR.peak!=0?HR_OLD.peak=HR.peak:HR_OLD.peak;
                        //Get HR averaged over 5 beats
                              if(HR.counter>=5){
                                HR.rate=(60000*HR.counter)/(HR_M[1][j]-HR.time);
                                    HR.time=HR_M[1][j];
                                     HR.counter=0;
                                    //Thresholding for HR
                                    if
(HR.rate<(HR_old.rate+HR_old.rate*.75)&&HR.rate>(HR_old.rate-
HR_old.rate*.75)&&HR.rate>35&&HR.rate<140){
                                    HR_old.rate=HR.rate;
                                    }
                                          else{
                                                      HR.rate=HR_old.rate;
                                          }

                            }
      }
      //Trough detection for HR
      HR_.trough=(Peak_Detect((filt_sig[0][j]),0));

      if(HR_OLD.trough!=HR_.trough ){
                  HR_.counter++;
            //Thresholding for RR frequency modulation
            if((HR_M[1][j]-HR_OLD.time)>450&&(HR_M[1][j]-HR_OLD.time)<2050){
            RR_.FM=(HR_M[1][j]-HR_OLD.time);
            HR_OLD.time=HR_M[1][j];
            }
            else{
```

```
            HR_OLD.time=HR_M[1][j];
            }
            //Baseline wander
                  RR_.BW=(HR_.trough+HR.peak)/2;
            //Amplitude modulation for RR
                  RR_.AM=(HR.peak-HR_.trough);
                  HR_OLD.trough=HR_.trough;
            //Get HR averaged over 5 beats
                        if(HR_.counter>=5){
                         HR_.rate=(60000*HR_.counter)/(HR_M[1][j]-HR_.time);
                              HR_.time=HR_M[1][j];
                                    HR_.counter=0;
                              if
(HR_.rate<(HR_OLD.rate+HR_OLD.rate*.75)&&HR_.rate>(HR_OLD.rate-
HR_OLD.rate*.75)&&HR_.rate>35 &&HR_.rate<140){
                              HR_OLD.rate=HR_.rate;
                              HR.BPM=(HR_.rate+HR.rate)/2;
                              //Kalman filter HR
                              HR_.BPM=Kalman(deltad,(HR_.rate+HR.rate)/2,0);
                              }
                                    else{
                                                HR_.rate=HR_OLD.rate;
                                    }
                                    //write HR, RR, HRV and time to SD card
                        fr = open_append(&fil, "logfile.csv");

      f_printf(&fil,"%d,%d,%d,%d,%d\n",HR_.BPM,rr_.average,(int)CI,HR_M[2][j]
,HR_M[3][j]);
                        fr = f_close(&fil);

                        }
      }

            //The following peak and trough detects the respiratory rate
signals to acquire RR
      rr_.peak=(Peak_Detect2(RR_.FM,1));
      if(rr_old.peak!=rr_.peak  ){
                  rr_.counter++;
                  rr_old.peak=rr_.peak;
              RR_.average+=rr_.peak;
                  rr_old.counter++;
            if(rr_old.counter>2){
                        deltad2=abs(RR_.average/(rr_old.counter*10));
                  RR_.average=0;
                  rr_old.counter=0;
                  }
                        if(rr_.counter>=3){
                         rr_.rate=(60000*rr_.counter)/(HR_M[1][j]-rr_.time);
                              rr_.time=HR_M[1][j];
                               rr_.counter=0;

                        }
      }

      RR_.trough=(Peak_Detect2(RR_.FM,0));
      if(RR_old.trough!=RR_.trough  ){
                  RR_.counter++;
```

```
                RR_old.trough=RR_.trough;
                      if(RR_.counter>=3){
                       RR_.rate=(60000*RR_.counter)/(HR_M[1][j]-RR_.time);
                            RR_.time=HR_M[1][j];
                        RR_.counter=0;
                                  RR_.rate=(RR_.rate+rr_.rate)/2;

                      }
      }



      rr_1.peak=(Peak_Detect3(RR_.BW,1));
      if(rr_old1.peak!=rr_1.peak  ){
                  rr_1.counter++;
                  rr_old1.peak=rr_1.peak;
            RR_1.average+=rr_1.peak;
                  rr_old1.counter++;
            if(rr_old1.counter>2){
                  deltad3=abs(RR_1.average/(rr_old1.counter*8));
                  RR_1.average=0;
                  rr_old1.counter=0;
                  }
                      if(rr_1.counter>=3){
                       rr_1.rate=(60000*rr_1.counter)/(HR_M[1][j]-
rr_1.time);
                            rr_1.time=HR_M[1][j];
                        rr_1.counter=0;

                      }
      }

      RR_1.trough=(Peak_Detect3(RR_.BW,0));
      if(RR_old1.trough!=RR_1.trough  ){
                  RR_1.counter++;
                  RR_old1.trough=RR_1.trough;
                      if(RR_1.counter>=3){
                       RR_1.rate=(60000*RR_1.counter)/(HR_M[1][j]-
RR_1.time);
                            RR_1.time=HR_M[1][j];
                        RR_1.counter=0;
                                  RR_1.rate=(RR_1.rate+rr_1.rate)/2;
                            //Kalman(deltad,(RR_.rate+rr_.rate)/2,1);
                      }
      }


      rr_2.peak=(Peak_Detect4(RR_.AM,1));
      if(rr_old2.peak!=rr_2.peak  ){
                  rr_2.counter++;
                  rr_old2.peak=rr_2.peak;
            RR_2.average+=rr_2.peak;
                  rr_old2.counter++;
            if(rr_old2.counter>2){
                  deltad4=abs(RR_2.average/(rr_old2.counter*8));
                  RR_2.average=0;
                  rr_old2.counter=0;
```

```
                  }
                        if(rr_2.counter>=3){
                         rr_2.rate=(60000*rr_2.counter)/(HR_M[1][j]-
rr_2.time);
                              rr_2.time=HR_M[1][j];
                          rr_2.counter=0;

                        }
      }

      RR_2.trough=(Peak_Detect4(RR_.AM,0));
      if(RR_old2.trough!=RR_2.trough  ){
                  RR_2.counter++;
                  RR_old2.trough=RR_2.trough;
                        if(RR_2.counter>=3){
                         RR_2.rate=(60000*RR_2.counter)/(HR_M[1][j]-
RR_2.time);
                              RR_2.time=HR_M[1][j];
                          RR_2.counter=0;
                                    RR_2.rate=(RR_2.rate+rr_2.rate)/2;
                              //Kalman(deltad,(RR_.rate+rr_.rate)/2,1);
                        }
      }
      //Updates RR based on negating the largest outlier
      rr_.average=(RR_2.rate+RR_1.rate+RR_.rate)/3;
      m=abs(RR_2.rate-rr_.average)>abs(RR_1.rate-
rr_.average)?RR_2.rate:RR_1.rate;
      ma=abs(RR_.rate-rr_.average)>abs(m-rr_.average)?RR_.rate:m;
      rr_.average=((rr_.average*3)-(ma))/2;

      if (rr_.average>5 && rr_.average<18){
            //Kalman Filter RR rate
            rr_.average=Kalman2(deltad3,rr_.average,0);
      rr_old.average=rr_.average;
      }
      else{
            rr_.average=rr_old.average;
      }

      }
      //Central Index to compute PNN50
      CI=(PBFS/(PBFS+NBFS))*100;
      TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
      TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
      GPIOIntEnable(GPIO_PORTE_BASE, GPIO_PIN_1);     // Enable interrupt

      }
      start=true;
      i=0;
      }

      }

      }
```

- **The following is code to initialize and setup the accelerometer.**

```
//Accel Functions and setup

#include <stdint.h>
#include "HRV_I2C_.h"
uint16_t* data;
uint8_t slave_addr=0x1D;
//uint8_t slave_addr=0x1C;
uint16_t XMSB=0x01;
uint16_t YMSB=0x03;
uint16_t ZMSB=0x05;
//uint16_t CTRL_REG1=0x2A;
uint16_t XYZ_Data_config=0x0E;
unsigned char F_READ=0x09;
unsigned char i2c_buff[2]={0};


//Put Accel into standby mode for register writes
void standby(uint8_t slave_addr,unsigned char *data){

      I2Cwrite(I2C0_BASE,slave_addr, data, 2);

}

//Put accel into active mode
void active(int slave_addr,unsigned char *data){
      I2Cwrite(I2C0_BASE,slave_addr, data, 2);
}

void SetMode(void){
//standby
      i2c_buff[0]=0x2A;
      i2c_buff[1]=0x00;
      standby(slave_addr,i2c_buff);
//set Read bit for 8 bit resolution
      i2c_buff[0]=0x2A;
      i2c_buff[1]=0x02;
      I2Cwrite(I2C0_BASE,slave_addr, i2c_buff, 2);
      //set HPF data
      i2c_buff[0]=0x0E;
      i2c_buff[1]=0x10;
      I2Cwrite(I2C0_BASE,slave_addr, i2c_buff, 2);
      //set HPF cut off
      i2c_buff[0]=0x0F;
      i2c_buff[1]=0x03; //2 Hz HPF
      I2Cwrite(I2C0_BASE,slave_addr, i2c_buff, 2);
//make active
      i2c_buff[0]=0x2A;
      i2c_buff[1]=0x01;
      active(slave_addr,i2c_buff);
//F_read is zero for proper auto increment address
      i2c_buff[0]=F_READ;
      i2c_buff[1]=0x0000;
      I2Cwrite(I2C0_BASE,slave_addr, i2c_buff, 2);
```

```
}

uint32_t get_xyz_data( void ){

            uint8_t data[4]={0};

            uint32_t ret_value=0;

            I2CRead(I2C0_BASE,0x1D,0x00,data, 4);

            ret_value |= (uint32_t)data[0] <<24;//register 0
            ret_value |= (uint32_t)data[1] <<16;//XMSB
            ret_value |= (uint32_t)data[2] <<8;//YMSB reg
            ret_value |= (uint32_t)data[3] <<0;//ZMSB reg

return ret_value;
}
```

- **The following is the code to setup I2C communication.**

```
//I2C initialization, read and write commands

#include <stdint.h>
#include <stdbool.h>
#include <float.h>
#include <stdlib.h>
#include "rom.h"
#include "sysctl.h"
#include <string.h>
#include "hw_memmap.h"
#include "fpu.h"
#include "gpio.h"
#include "interrupt.h"
#include "pin_map.h"
#include "TM4C123GH6PM.h"
#include "i2c.h"
#include "hw_types.h"
#include "HRV_I2C_.h"
#include "uart.h"
#include "uartstdio.h"


#define GPIO_PE4_I2C2SCL        0x00041003
#define GPIO_PE5_I2C2SDA        0x00041403
#define I2C_SLAVE_ADDRESS       0x58
uint32_t k;
//to check if reading and writing is finished
unsigned long WaitI2CDone( unsigned int long ulBase){
    // Wait until done transmitting
    while( I2CMasterBusy(I2C2_BASE));
    // Return I2C error code
    return I2CMasterErr( I2C2_BASE);
}

//Setup I2C2 for Initialization
```

```
void SetupI2C(void){

     // I2C Setting

      SysCtlPeripheralEnable( SYSCTL_PERIPH_I2C2);
      SysCtlPeripheralEnable( SYSCTL_PERIPH_I2C0);
      //reset I2C module
      SysCtlPeripheralReset(SYSCTL_PERIPH_I2C2);
      SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

      GPIOPinConfigure(GPIO_PE4_I2C2SCL);
      GPIOPinConfigure(GPIO_PE5_I2C2SDA);
      GPIOPinConfigure(GPIO_PB2_I2C0SCL);
      GPIOPinConfigure(GPIO_PB3_I2C0SDA);

// Configure the pin muxing for I2C2 functions on port E4 and E5.
    GPIOPinTypeI2CSCL(GPIO_PORTE_BASE, GPIO_PIN_4);
    GPIOPinTypeI2C(GPIO_PORTE_BASE, GPIO_PIN_5);
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

   // Enable and initialize the I2C2 master module.  Use the system
clock for
      // the I2C2 module.  The last parameter sets the I2C data
transfer rate.
      // If false the data rate is set to 100kbps and if true the data
rate will
      // be set to 400kbps.

    I2CMasterInitExpClk( I2C2_BASE, SysCtlClockGet(), true);
            I2CMasterInitExpClk( I2C0_BASE, SysCtlClockGet(), true);
//clear I2C FIFOs
    SysCtlDelay(10000);
}



unsigned char I2CRead(uint32_t base,unsigned char addr,unsigned char
device_register, unsigned char* data,unsigned char len)
{
    if (len < 1) { // Assume I2C Recieving will always return data
        return -1;
            }
    // Set address and register to write to *****This may not be needed
for initialization*******
    I2CMasterSlaveAddrSet(base, addr, false);
            I2CMasterDataPut(base, device_register);
            I2CMasterControl(base, I2C_MASTER_CMD_SINGLE_SEND);
      while(I2CMasterBusy(base));
            k=I2CMasterErr( base);
```

```
      //*****************************************************************************
**********************

                I2CMasterSlaveAddrSet(base, addr, true);
          // Check to see if pointer is to an array
          if (len == 1){
              I2CMasterControl(base, I2C_MASTER_CMD_SINGLE_RECEIVE);
              k=  WaitI2CDone(base);

              *data = I2CMasterDataGet(base);
              return *data;

          }

          // Begin reading consecutive data
          I2CMasterControl(base, I2C_MASTER_CMD_BURST_RECEIVE_START);
                  WaitI2CDone(base);

          *data = I2CMasterDataGet(base);// *data is the actual value in
memory

          len--;
          data++;//address ++, this goes to the next address and stores the
*data there

          // Continue reading consecutive data
          while(len > 1){
              I2CMasterControl(base, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
              while(I2CMasterBusy(base));
                  k=I2CMasterErr( base);

              *data = I2CMasterDataGet(base);
              len--;
              data++;

          }

          // Read last character of data
          I2CMasterControl(base, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
            while(I2CMasterBusy(base));
                  k=I2CMasterErr( base);


          *data = I2CMasterDataGet(base);

                  return *data;
      }



      void I2Cwrite(uint32_t base, uint8_t addr, unsigned char *data,
unsigned int len)
      {

          I2CMasterSlaveAddrSet(base, addr, false);
```

```
    I2CMasterDataPut(base, *data);

    if (len == 1){
        I2CMasterControl(base, I2C_MASTER_CMD_SINGLE_SEND);
        while(I2CMasterBusy(base));
        return;
    }

    // Start sending consecutive data
    I2CMasterControl(base, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(base));


    len--;
    data++;

    // Continue sending consecutive data
    while(len > 1){
        I2CMasterDataPut(base, *data);
        I2CMasterControl(base, I2C_MASTER_CMD_BURST_SEND_CONT);
        while(I2CMasterBusy(base));
                  k=I2CMasterErr( base);

        len--;
        data++;
    }

 // Send last piece of data
    I2CMasterDataPut(base, *data);
    I2CMasterControl(base, I2C_MASTER_CMD_BURST_SEND_FINISH);
    while(I2CMasterBusy(base));
            k=I2CMasterErr( base);

}
```

- **The following code are the functions defining the algorithms in the health watch.**

```
//These are the algorithms for resp and HRV
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


//RLS initialization

static double lambda=.999999;
float w_n=0;
float P_n=10;
float pi_n=10;
float k_n=10;
float e_n=10;
```

```
//.Initialize Kalman Filter Values


int kcount=0;

int kcount2=0;

typedef struct Kalm{
      float Xo;
      float Po;
      float H;
      float Q;
  float R;
      float A;
  float Xn;
      float K;
      float Pn;
}Kalm_Filt;

Kalm_Filt kalman={60,90,1,9,41,1,60,1,1};
Kalm_Filt kalman2={800,600,1,60,1100,1,200,200,1};
Kalm_Filt kalman3={800,600,1,60,1100,1,200,200,1};
Kalm_Filt kalman4={10,20,1,2,31,1,10,1,1};

//Peak Detection
typedef struct peak_detect{
      signed int max;
      signed int min;
      signed int maxmax;
      signed int minmin;
      bool find_max;
}PEAK;
PEAK peak={-100,100,0,0,true};
PEAK peak2={-100,100,0,0,true};
PEAK peak3={-100,100,0,0,true};
PEAK peak4={-100,100,0,0,true};

int deltad=800;
int deltad2=100;
int deltad3=100;
int deltad4=100;

//Function Initialozation
signed int Kalman(signed int LED_val,signed int LED_val1,int order);
signed int Kalman2(signed int LED_val,signed int LED_val1,int order);
signed int Peak_Detect(signed int LED_val,int min_max);
signed int Peak_Detect2(signed int LED_val,int min_max);
signed int Peak_Detect3(signed int LED_val,int min_max);
signed int Peak_Detect4(signed int LED_val,int min_max);
signed char  physical_activity_count(signed char x,signed char
y,signed char  z);
signed int RLS(signed int LED_val,signed char accel);


//H is observation variable, A is state variable,     Q is covarience
for state noise, R is observation noise, x_n is state and Posterior mean,
```

```
        //P_n is covariance of x and posterior covarience
        //Kalman Filter for HR
        signed int Kalman(signed int LED_val,signed int LED_val1,int order){

        if(kcount==0){

        kalman2.Xo=LED_val1;
        kalman.Xo=LED_val;

        }
        kcount=1;
        //compute state
        kalman2.Xn=kalman2.A*kalman2.Xo;
        kalman.Xn=kalman.A*kalman.Xo;
        //compute covarience of state
        kalman2.Pn=kalman2.A*kalman2.Po*kalman2.A+kalman2.Q;
        kalman.Pn=kalman.A*kalman.Po*kalman.A+kalman.Q;
        //compute Kalman Gain
        kalman2.K=kalman2.Pn*kalman2.H/(kalman2.H*kalman2.Pn*kalman2.H+(kalman2
.R));
        kalman.K=kalman.Pn*kalman.H/(kalman.H*kalman.Pn*kalman.H+(kalman.R));
        //compute Posterior Covarience
        kalman2.Pn=(1-kalman2.K*kalman2.H)*kalman2.Pn;
        kalman.Pn=(1-kalman.K*kalman.H)*kalman.Pn;
        //Compute Posterior state
        kalman2.Xn=kalman2.Xn+kalman2.K*(LED_val1-kalman2.H*kalman2.Xn);
        kalman.Xn=kalman.Xn+kalman.K*(LED_val-kalman.H*kalman.Xn);
        //update
        kalman2.Xo=kalman2.Xn;
        kalman.Xo=kalman.Xn;
        kalman2.Po=kalman2.Pn;
        kalman.Po=kalman.Pn;

        if(order){

        return (signed int)kalman.Xn;
        }
              else

              return (signed int)kalman2.Xn;

        }
        //Kalmna Filter for RR
        signed int Kalman2(signed int LED_val,signed int LED_val1,int order){

        if(kcount2==0){

              kalman3.Xo=LED_val1;
              kalman4.Xo=LED_val;
        }
        kcount2=1;
        //compute state
        kalman3.Xn=kalman3.A*kalman3.Xo;
        kalman4.Xn=kalman4.A*kalman4.Xo;
        //compute covarience of state
        kalman3.Pn=kalman3.A*kalman3.Po*kalman3.A+kalman3.Q;
        kalman4.Pn=kalman4.A*kalman4.Po*kalman4.A+kalman4.Q;
```

```
      //compute Kalman Gain
      kalman3.K=kalman3.Pn*kalman3.H/(kalman3.H*kalman3.Pn*kalman3.H+(kalman3
.R));
      kalman4.K=kalman4.Pn*kalman4.H/(kalman4.H*kalman4.Pn*kalman4.H+(kalman4
.R));
      //compute Posterior Covarience
      kalman3.Pn=(1-kalman3.K*kalman3.H)*kalman3.Pn;
      kalman4.Pn=(1-kalman4.K*kalman4.H)*kalman4.Pn;
      //Compute Posterior state
      kalman3.Xn=kalman3.Xn+kalman3.K*(LED_val1-kalman3.H*kalman3.Xn);
      kalman4.Xn=kalman4.Xn+kalman4.K*(LED_val-kalman4.H*kalman4.Xn);
      //update
      kalman3.Xo=kalman3.Xn;
      kalman4.Xo=kalman4.Xn;
      kalman3.Po=kalman3.Pn;
      kalman4.Po=kalman4.Pn;

      if(order){
            return (signed int)kalman3.Xn;
      }
            else
            return (signed int)kalman4.Xn;

      }

      //Peak Detection Algorithm

      signed int Peak_Detect(signed int LED_val,int min_max){

      //initially should always be true
       if (LED_val>peak.max){
            peak.max=LED_val;

        }
      else if (LED_val<peak.min){
            peak.min=LED_val;

      }
      else if (peak.find_max==true  ) { //check to see if min or max was last
found if (find_max==true){

        if (LED_val<peak.max-deltad){//If value is less than last peak value
and a step size, step helps control variability
         //Create a counter to track time between beats here, or possibly in
individual function
          peak.maxmax=peak.max;

          peak.min=LED_val; //update min, needed to update max

          peak.find_max=false;
          }
       }
       else   if(LED_val>peak.min+deltad){

            peak.minmin=peak.min;

            peak.max=LED_val; // update max
```

```
      peak.find_max=true;
    }

      if(min_max){

return peak.maxmax;
      }

      else
            return peak.minmin;

}


//Peak Detection Algorithm

signed int Peak_Detect2(signed int LED_val,int min_max){

//initially should always be true
 if (LED_val>peak2.max){
      peak2.max=LED_val;

  }
else if (LED_val<peak2.min){
      peak2.min=LED_val;

}
else if (peak2.find_max==true  ) { //check to see if min or max was
last found if (find_max==true){

   if (LED_val<peak2.max-deltad2){//If value is less than last peak
value and a step size, step helps control variability
    //Create a counter to track time between beats here, or possibly in
individual function
     peak2.maxmax=peak2.max;

     peak2.min=LED_val; //update min, needed to update max

     peak2.find_max=false;
     }
 }
 else   if(LED_val>peak2.min+deltad2){

      peak2.minmin=peak2.min;

      peak2.max=LED_val; // update max

      peak2.find_max=true;
    }

      if(min_max){

return peak2.maxmax;
      }

      else
```

```
                return peak2.minmin;

        }


        //Peak Detection Algorithm

        signed int Peak_Detect3(signed int LED_val,int min_max){

        //initially should always be true
         if (LED_val>peak3.max){
               peak3.max=LED_val;

          }
        else if (LED_val<peak3.min){
               peak3.min=LED_val;

        }
        else if (peak3.find_max==true  ) { //check to see if min or max was
last found if (find_max==true){

          if (LED_val<peak3.max-deltad3){//If value is less than last peak
value and a step size, step helps control variability
           //Create a counter to track time between beats here, or possibly in
individual function
            peak3.maxmax=peak3.max;

            peak3.min=LED_val; //update min, needed to update max

            peak3.find_max=false;
            }
         }
         else if(LED_val>peak3.min+deltad3){

               peak3.minmin=peak3.min;

               peak3.max=LED_val; // update max

               peak3.find_max=true;
            }

               if(min_max){

        return peak3.maxmax;
               }

               else
                     return peak3.minmin;

        }



        //Peak Detection Algorithm

        signed int Peak_Detect4(signed int LED_val,int min_max){
```

```
      //initially should always be true
       if (LED_val>peak4.max){
            peak4.max=LED_val;

        }
      else if (LED_val<peak4.min){
            peak4.min=LED_val;

      }
      else if (peak4.find_max==true  ) { //check to see if min or max was
last found if (find_max==true){

        if (LED_val<peak4.max-deltad4){//If value is less than last peak
value and a step size, step helps control variability
         //Create a counter to track time between beats here, or possibly in
individual function
          peak4.maxmax=peak4.max;

          peak4.min=LED_val; //update min, needed to update max

          peak4.find_max=false;
          }
       }
       else if(LED_val>peak4.min+deltad4){

            peak4.minmin=peak4.min;

            peak4.max=LED_val; // update max

            peak4.find_max=true;
          }

            if(min_max){

      return peak4.maxmax;
            }

            else
                  return peak4.minmin;

      }

      signed char  physical_activity_count(signed char x,signed char
y,signed char  z){

            //if xyz are less then a cetain threshold then return 0

            //Activity Count Vector Magnitude
      return sqrt((x*x)+(z*z)+(y*y));

      }


      signed int RLS(signed int LED_val,signed char accel){

            e_n=LED_val-w_n*accel;
            k_n=(accel)/(lambda*P_n*P_n+accel*accel);
```

```
        P_n=lambda*P_n*P_n+accel*accel;
        w_n=w_n+k_n*e_n;
        return (signed int)e_n;

}
```

- **The following is the Recursive Least Squares Implementation in C using matrices.**

```
    static const signed char iv0[256] = { 10, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10,
0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
10, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 10, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 10,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
        10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10 };

    int cr;
    int ic;
    int br;
    int ib;
    int ia;

    float b_y[p_size*p_size]={0};
    float P[p_size*p_size]={0.0};
    float ky[p_size*p_size]={0};
    float kyP[p_size*p_size]={0};
    float w_[p_size]={0};
    float  pi[p_size]={0};
    signed short Y[p_size]={0};
    float e[data_length]={0};


    memset(&w_[0], 0, sizeof(int) << 4);
    for(step=0;step<(data_length+1)-(p_size);step++){

    for(j=0;j<p_size;j++){
        Y[j]=filt_sig[1][(step-j)+(p_size-1)];

    }
```

```
v=0;

for(j=0;j<p_size;j++){
      v+=w_[j]*Y[j];
      pi[j]=0.0;
}
e[step+(p_size-1)]=filt_sig[0][step+(p_size-1)]-v;
for(j=0;j<p_size;j++){
pi[j]=0.0;
      for(m=0;m<p_size;m++){
      pi[j] += P[j + (m << 4)] * Y[m];
      }
}
v=0;
for(j=0;j<p_size;j++){
      v+=pi[j]*Y[j];
}
for(j=0;j<p_size;j++){
      pi[j]/=lambda+v;
      for(m=0;m<p_size;m++){
            ky[j + (m << 4)] = pi[j] * Y[m];
      }

}

memset(&kyP[0], 0, sizeof(double) << 8);
for(cr=0;cr<p_size;cr++){
for(j=0;j<p_size;j++){
            for(m=0;m<p_size;m++){
                  kyP[br]+=ky[cr*p_size+m]*P[j+m*p_size];
            }
            br++;
}

}
br=0;

memset(&b_y[0], 0, sizeof(int) << 8);
    for (cr = 0; cr <= p_size*p_size-(p_size+1); cr += p_size) {
      for (ic = cr; ic + 1 <= cr + p_size; ic++) {
        b_y[ic] = 0.0;
      }
    }

    br = 0;
    for (cr = 0; cr <= (p_size*p_size)-(p_size-1); cr += p_size) {
      j = 0;
      i = br + p_size;
      for (ib = br; ib + 1 <= i; ib++) {
        if (P[ib] != 0.0) {
          ia = j;
          for (ic = cr; ic + 1 <= cr + p_size; ic++) {
            ia++;
            b_y[ic] += P[ib] * ky[ia - 1];
          }
        }
```

```
            j += p_size;
          }

          br += p_size;
        }

      for(m=0;m<p_size*p_size;m++){
        P[m]-=b_y[m];
        P[m]*=1/lambda;

      }

      for(m=0;m<p_size;m++){
      w_[m]+=pi[m]*e[step+(p_size-1)];
      }

      filt_sig[2][step]=(signed int)e[step+(p_size-1)];

      }
```

- **The following code is the AFE4404 system setup, with all needed register functions. The AFE4404 was initialized for 100Hz sampling.**

```
      //Initialization of TI AFE4404 HR Analog Front end

      #include <stdint.h>
      #include "AFE4404_REG_SETUP_2.h"
      #include "HRV_I2C_.h"
      #define I2C_SLAVE_ADDRESS       0x58


      uint8_t AFE4404_init( uint8_t address)
      {
          //hr_hal_init( address );
              SetupI2C();
            //enable software Reset and Disable counter reset and disable
register readout mode
          hr_set_settings( );

                  // The following is for NO DIVISION OF CLOCK TO TIMING
ENGINE CLOCK (CLKDIV_PRF = 1)

          hr_set_led2_start_end( 0, 399 );
          hr_set_led2_sample_start_end( 80, 399 );
          hr_set_adc_reset0_start_end( 401, 407 );
          hr_set_led2_convert_start_end( 408, 1467 );
          hr_set_led3_start_stop( 400, 799 );
          hr_set_led3_sample_start_end( 480, 799 );
          hr_set_adc_reset1_start_end( 1469, 1475 );
          hr_set_led3_convert_start_end( 1476, 2535 );
          hr_set_led1_start_end( 800, 1199 );
          hr_set_led1_sample_start_end( 880, 1199 );
          hr_set_adc_reset2_start_end( 2537, 2543 );
```

```
          hr_set_led1_convert_start_end( 2544, 3603 );
          hr_set_amb1_sample_start_end( 1279, 1598 );
          hr_set_adc_reset3_start_end( 3605, 3611 );
          hr_set_amb1_convert_start_end( 3612, 4671 );
          hr_set_pdn_cycle_start_end( 5471, 39199 );
          hr_set_prpct_count( 39999 );//Clock timing for CLKDIV_PRF  = 1

          //ADC Average Set to 3 for Timing Control
          hr_set_timer_and_average_num( true, 7 );

          hr_set_seperate_tia_gain( true, 7, 0 ); //Allows for two sets of
TIA gain settings for the channels, when true args 2 and 3 control cf2 and
rf2 for LED2,3

          hr_set_tia_gain( false, 7, 0);// set TIA resistor value and
capacitor
      /* ********************************************************************
       * LED RF and CF Bit values                    *
       * ******************************************************************** *
       *             Register Values      |              Rf k ohm
       *                   0              |              500
       *                   1              |              250
       *                   2              |              100
       *                   3              |              50
       *                   4              |              25
       *                   5              |              10
       *                                                   6
                                                          1M
                                                          7
                                                               2M
       ********************************************************************
       * ******************************************************************** *
       *                  Cap Values      |              Cf pf
       *                   0              |              5
       *                   1              |              2.50
       *                   2              |              10
       *                   3              |              7.5
       *                   4              |              20
       *                   5              |              17.5
       *                                                   6
                                                          25
                                                          7
                                                               22.5


**********************************************************************/


                //100 mA LEDS ,GREENMAX=25 REDMAX=40 IRMAX= 60
          hr_set_led_currents( 15,15,0);

      /* ********************************************************************
       * LED current control is a 24 bit register where                     *
       * LED1: bits 0-5 LED2: bits 6-11 LED3: bits 12-17 and the rest are 0 *
       * ******************************************************************** *
       * LED1, LED2, LED3 Register Values | LED Current Setting (mA)         *
       *                   0              |              0                   *
```

```
        *                    1              |              0.8                *
        *                    2              |              1.6                *
        *                    3              |              2.4                *
        *                   ...             |              ...                *
        *                    63             |               50                *

*********************************************************************/



                //division ratio for clk_ext div=1 4MHZ
                hr_set_inp_inn_settings(5);

                /*
                Transmitter disabled,
                LED 0-100,
                ADC pwrd dwn,
                Ext clk,
                TIA NPDn,
                RADC NPDn,
                Normal Mode,
                Normal Mode
                */

      hr_set_dynamic_settings( 0,1,1,0,0,0,0,0 );



      hr_set_offdac_settings(0,0,0,0,0,0,0,0);
      /*
      Set DAC Offsets to be LED 2 > -1.4,LED3 > -1.87 and LED1 > -.93
      Order is
LED2(pol,dac_v),Ambient(pol,dac_v),Led1(pol,dac_v),LED3(pol,Dac_v)
      * ************************************************************
       *  I_OFFDAC Register Settings                                 *
       * ************************************************************
       * * Reg. Settings | Offset Cancellation | Offset Cancellation  |
       * *               |   POL_OFFDAC = 0    |   POL_OFFDAC = 1     |
       * ************************************************************
       *        0        |          0          |           0          *
       *        1        |       0.47          |       -0.47          *
       *        2        |       0.93          |       -0.93          *
       *        3        |        1.4          |        -1.4          *
       *        4        |       1.87          |       -1.87          *
       *        5        |       2.33          |       -2.33          *
       *        6        |        2.8          |        -2.8          *
       *        7        |       3.27          |       -3.27          *
       *        8        |       3.73          |       -3.73          *
       *        9        |        4.2          |        -4.2          *
       *       10        |       4.67          |       -4.67          *
       *       11        |       5.13          |       -5.13          *
       *       12        |        5.6          |        -5.6          *
       *       13        |       6.07          |       -6.07          *
       *       14        |       6.53          |       -6.53          *
       *       15        |          7          |          -7          *
```

```
 ***************************************************************/


    // hr_set_clkout_div( false, 2 );
     hr_set_int_clk_div( 0 );

     return 0;
}

uint8_t hr_set_settings( void )
{
     uint8_t reg = DIAGNOSIS;
     uint8_t temp[4] = { 0 };

             temp[0]  = reg;
     temp[3] |= ( 1 << DIAG_SW_RST );
     I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );

     return 0;
}

uint8_t hr_set_led2_sample_start_end( uint16_t start, uint16_t end )
{
     uint8_t reg_st = SMPL_LED2_ST;
     uint8_t reg_end = SMPL_LED2_END;
     uint8_t temp_st[4] = { 0 };
     uint8_t temp_end[4] = { 0 };

     if( start > 65535 || end > 65535 ) //Checking to see if the bit
count is > 16
         return -1;
             temp_st[0] = reg_st;
     temp_st[2] = start >> 8;
     temp_st[3] = (uint8_t)start;

     I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,  temp_st, 4 );
             temp_end[0] = reg_end;
     temp_end[2] = end >> 8;
     temp_end[3] = (uint8_t)end;

     I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

     return 0;

}

uint8_t hr_set_led1_start_end( uint16_t start, uint16_t end )
{
     uint8_t reg_st = LED1_ST;
     uint8_t reg_end = LED1_END;
     uint8_t temp_st[4] = { 0 };
     uint8_t temp_end[4] = { 0 };

     if( start > 65535 || end > 65535 )
         return -1;
```

```
    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_end, 4 );

    return 0;

}

uint8_t hr_set_led3_sample_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = SMPL_LED3_ST;
    uint8_t reg_end = SMPL_LED3_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,  temp_end, 4 );

    return 0;

}

uint8_t hr_set_led1_sample_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = SMPL_LED1_ST;
    uint8_t reg_end = SMPL_LED1_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );
```

```
    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,  temp_end, 4 );

    return 0;

}

uint8_t hr_set_led2_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = LED2_ST;
    uint8_t reg_end = LED2_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_amb1_sample_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = SMPL_AMB1_ST;
    uint8_t reg_end = SMPL_AMB1_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;


    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,  temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;
```

```
    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_led2_convert_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = LED2_CONV_ST;
    uint8_t reg_end = LED2_CONV_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_led3_convert_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = LED3_CONV_ST;
    uint8_t reg_end = LED3_CONV_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;


    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_end, 4 );

    return 0;
```

```
}

uint8_t hr_set_led1_convert_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = LED1_CONV_ST;
    uint8_t reg_end = LED1_CONV_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;


    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_amb1_convert_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = AMB1_CONV_ST;
    uint8_t reg_end = AMB1_CONV_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_adc_reset0_start_end( uint16_t start, uint16_t end )
```

```
{
    uint8_t reg_st = ADC_RST_P0_ST;
    uint8_t reg_end = ADC_RST_P0_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_adc_reset1_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = ADC_RST_P1_ST;
    uint8_t reg_end = ADC_RST_P1_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_end, 4 );

    return 0;

}

uint8_t hr_set_adc_reset2_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = ADC_RST_P2_ST;
    uint8_t reg_end = ADC_RST_P2_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };
```

```
    if( start > 65535 || end > 65535  )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_st,4);

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_adc_reset3_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = ADC_RST_P3_ST;
    uint8_t reg_end = ADC_RST_P3_END;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;


    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;

}

uint8_t hr_set_prpct_count( uint16_t count )
{
    uint8_t reg = PRPCT;
    uint8_t temp[4] = { 0 };

    if( count > 65535 )
        return -1;

            temp[0] = reg;
    temp[2] = count >> 8;
```

```
        temp[3] = (uint8_t)count;

        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );

        return 0;

    }

    //*************** The Following Sections Need Work**************

    uint8_t hr_set_timer_and_average_num( bool enable, uint8_t av_num )
    {
        uint8_t reg = TIM_NUMAV;
        uint8_t temp[4] = { 0 };
                temp[0] = reg;
        if( av_num > 15 || av_num < 0 )
            return -1;

        if( enable )
        {
            temp[2] |= ( 1 << TIMEREN );//1 gets set Byte 2, 8 bit in reg
with no shift
            temp[3] |= ( av_num << NUMAV ); //av_num gets set Byte 1, no
shift needed
            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );
        }
        else
        {
            temp[3] |= ( av_num << NUMAV );
            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );
        }

        return 0;

    }

    uint8_t hr_set_seperate_tia_gain( bool seperate, uint8_t cf_setting,
                                      uint8_t gain_setting )
    {
        uint8_t reg = TIA_GAINS2;
        uint8_t temp[4] = { 0 };
                temp[0] = reg;

        if( cf_setting > 7 || gain_setting > 7 )
            return -1;

        if( seperate )
        {
            temp[2] = TIA_ENSEPGAIN;
            temp[3] |= ( cf_setting << TIA_CF_SEP );
            temp[3] |= ( gain_setting << TIA_GAIN_SEP );
            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );
        }
        else
        {
            temp[3] |= ( cf_setting << TIA_CF_SEP );
            temp[3] |= ( gain_setting << TIA_GAIN_SEP );
```

```
        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );
    }

    return 0;

}

uint8_t hr_set_tia_gain( bool replace, uint8_t cf_setting,
                            uint8_t gain_setting )
{
    uint8_t reg = TIA_GAINS1;
    uint8_t temp[4] = { 0 };
            temp[0] = reg;
    if( cf_setting > 7 || gain_setting > 7 )
        return -1;

    if( replace )
    {
        temp[2] = TIA_PROG_TG_EN;
        temp[3] |= ( cf_setting << TIA_CF );
        temp[3] |= ( gain_setting << TIA_GAIN );
        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );
    }
    else
    {
        temp[2] = 0;
        temp[3] |= ( cf_setting << TIA_CF_SEP );
        temp[3] |= ( gain_setting << TIA_GAIN_SEP );
        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4);
    }

    return 0;

}

uint8_t hr_replace_adc( bool replace )
{
    uint8_t reg = TIA_GAINS1;
    uint8_t temp_read[3] = {0};
            uint8_t temp_write[4] = {0};
            temp_write[0] = reg;
            //enable reading of regs
    hr_read_enable();
    I2CRead(I2C2_BASE,I2C_SLAVE_ADDRESS,  reg, temp_read, 3 );
    hr_read_disable();

    if( replace )
    {
        temp_write[2] |= TIA_PROG_TG_EN;
        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_write, 4 );
    }
    else
    {
        temp_write[2] &= ~( TIA_PROG_TG_EN );
        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_write, 4 );
    }
```

```
        return 0;

    }

    uint8_t hr_set_led_currents( uint8_t led1_current, uint8_t
led2_current,
                                 uint8_t led3_current )
    {
        uint8_t reg = LED_CONFIG;
        uint8_t temp[4] = { 0 };
        unsigned long currents = 0;
                temp[0] = reg;

        if( led1_current > 63 ||
            led2_current > 63 ||
            led3_current > 63 )
            return -1;

        currents |= ( led3_current << ILED3 );
        currents |= ( led2_current << ILED2 );
        currents |= led1_current << ILED1;

        temp[3] |= currents;
        temp[2] |= currents >> 8;
        temp[1] |= currents >> 16;


        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );

        return 0;

    }

    uint8_t hr_set_dynamic_settings( uint8_t Dyn1, uint8_t ILED_2X,uint8_t
Dyn2,uint8_t OSC_ENABLE,uint8_t Dyn3,
    uint8_t Dyn4,uint8_t PDNRX,uint8_t PDNAFE )
    {
        uint8_t reg = SETTINGS;
                    unsigned long buffer = 0;
        uint8_t temp[4] = { 0 };
                temp[0] = reg;

        buffer |= ( Dyn1 << STT_DYNMC1 );
        buffer |= ( ILED_2X << STT_ILED_2X );
        buffer |= ( Dyn2 << STT_DYNMC2 );
        buffer |= ( OSC_ENABLE<< STT_OSC_EN );
        buffer |= ( Dyn3 << STT_DYNMC3 );
        buffer |= ( Dyn4 << STT_DYNMC4 );
        buffer |= ( PDNRX << STT_PDNRX );
        buffer |= ( PDNAFE << STT_PDNAFE );

        temp[3] |= buffer;
        temp[2] |= buffer >> 8;
        temp[1] |= buffer >> 16;


        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );
```

```
        return 0;
    }
    //Only used for internal clock mode
    uint8_t hr_set_clkout_div( bool enable, uint8_t div )
    {
        uint8_t reg = CLKOUT;
        uint8_t temp[4] = { 0 };
                temp[0] = reg;

        if( div > 15 )
            return -1;

        if( enable )
        {
            temp[2] = ( 1 << CLKOUT_EN );
            temp[3] = ( div << CLKOUT_DIV );
            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );
        }
        else
        {
            temp[3] = ( div << CLKOUT_DIV );
            I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );
        }

        return 0;

    }

    uint32_t hr_get_led1_val( void )
    {
        uint8_t reg = LED1VAL;
        uint8_t temp[3] = { 0 };
        uint32_t retval = 0;

        I2CRead(I2C2_BASE,I2C_SLAVE_ADDRESS,  reg, temp, 3 );

        retval |= (uint32_t)temp[0] << 16;
        retval |= (uint32_t)temp[1] << 8;
        retval |= (uint32_t)temp[2];
    //          if (retval>>23==1) // check if the ADC value is positive or
negative
    //{

    //retval &= 0x003FFFFF; // convert it to a 22 bit value
    //return (retval^0xFFC00000);
    //}

        return retval;

    }

    uint32_t hr_get_led2_val( void )
    {
        uint8_t reg = LED2VAL;
        uint8_t temp[3] = { 0 };
        uint32_t retval = 0;
```

```
            I2CRead(I2C2_BASE,I2C_SLAVE_ADDRESS,  reg, temp, 3 );

            retval |= (uint32_t)temp[0] << 16;
            retval |= (uint32_t)temp[1] << 8;
            retval |= (uint32_t)temp[2];
              //    if (retval & 0x00200000) // check if the ADC value is
positive or negative
      //{
      //retval &= 0x003FFFFF; // convert it to a 22 bit value
      //return (retval^0xFFC00000);
      //}

            return retval;

      }

      uint32_t hr_get_led3_val( void )
      {
            uint8_t reg = LED3VAL;
            uint8_t temp[3] = { 0 };
            uint32_t retval = 0;

            I2CRead( I2C2_BASE,I2C_SLAVE_ADDRESS, reg, temp, 3 );

            retval |= (uint32_t)temp[0] << 16;
            retval |= (uint32_t)temp[1] << 8;
            retval |= (uint32_t)temp[2];
              //    if (retval & 0x00200000) // check if the ADC value is
positive or negative
      //{
      //retval &= 0x003FFFFF; // convert it to a 22 bit value
      //return (retval^0xFFC00000);
      //}

            return retval;

      }

      uint32_t hr_get_amb1_val( void )
      {
            uint8_t reg = ALED1VAL;
            uint8_t temp[3] = { 0 };
            uint32_t retval = 0;

            I2CRead(I2C2_BASE, I2C_SLAVE_ADDRESS, reg, temp, 3 );

            retval |= ( temp[0] << 16 );
            retval |= ( temp[1] << 8 );
            retval |= temp[2];

              //    if (retval & 0x00200000) // check if the ADC value is
positive or negative
      //{
      //retval &= 0x003FFFFF; // convert it to a 22 bit value
      //return (retval^0xFFC00000);
      //}
```

```
        return retval;

    }

    uint32_t hr_get_led2_amb2_val( void )
    {
        uint8_t reg = LED2_ALED2VAL;
        uint8_t temp[3] = { 0 };
        uint32_t retval = 0;

        I2CRead(I2C2_BASE,I2C_SLAVE_ADDRESS,  reg, temp, 3 );

        retval |= ( temp[0] << 16 );
        retval |= ( temp[1] << 8 );
        retval |= temp[2];

    //          if (retval & 0x00200000) // check if the ADC value is
positive or negative
    //{
    //retval &= 0x003FFFFF; // convert it to a 22 bit value
    //return (retval^0xFFC00000);
    //}

        return retval;

    }

    uint32_t hr_get_led1_amb1_val( void )
    {
        uint8_t reg = LED1_ALED1VAL;
        uint8_t temp[3] = { 0 };
        uint32_t retval = 0;

        hr_read_enable();
        I2CRead(I2C2_BASE,I2C_SLAVE_ADDRESS,  reg, temp, 3 );
        hr_read_disable();

        retval |= ( temp[0] << 16 );
        retval |= ( temp[1] << 8 );
        retval |= temp[2];

        return retval;

    }

    uint8_t hr_is_pd_shorted( void )
    {
        uint8_t reg = PD_SHORT_FLAG;
        uint8_t temp[3] = { 0 };

        I2CRead(I2C2_BASE,I2C_SLAVE_ADDRESS,  reg, temp, 3 );

        if( temp[2] == 1 )
            return -1;
        else
            return 0;
```

```
}

//uint8_t hr_set_inp_inn_settings( inm_inn_t* inp_inn_setting )
uint8_t hr_set_inp_inn_settings(uint8_t CLKDIV_EXT_SET)
{
    uint8_t reg = PD_INP_EXT;
    uint8_t temp[4] = { 0 };
            temp[0] = reg;

   // if( inp_inn_setting->ext_div > 7 )
   //     return -1;

    //temp[2] = //( inp_inn_setting->pd_setting << PD_DISCONNECT );
    //temp[3] = //( inp_inn_setting->short_setting <<
ENABLE_INPUT_SHORT );
    //temp[3] = 0x05;//inp_inn_setting->ext_div;
            temp[3] =   CLKDIV_EXT_SET << CLKDIV_EXTMODE;
    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );

    return 0;

}

uint8_t hr_set_pdn_cycle_start_end( uint16_t start, uint16_t end )
{
    uint8_t reg_st = PDNCYCLESTC;
    uint8_t reg_end = PDNCYCLEENDC;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,  temp_end, 4 );

    return 0;


}

uint8_t hr_set_prgrmmbl_timing_start_end( uint16_t start, uint16_t end
)
{
    uint8_t reg_st = PROG_TG_STC;
    uint8_t reg_end = PROG_TG_ENDC;
    uint8_t temp_st[4] = { 0 };
```

```
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp_st, 4 );

    temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;


}

uint8_t hr_set_led3_start_stop( uint16_t start, uint16_t end )
{
    uint8_t reg_st = LED3LEDSTC;
    uint8_t reg_end = LED3LEDENDC;
    uint8_t temp_st[4] = { 0 };
    uint8_t temp_end[4] = { 0 };

    if( start > 65535 || end > 65535 )
        return -1;

    temp_st[0] = reg_st;
    temp_st[2] = start >> 8;
    temp_st[3] = (uint8_t)start;

I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_st, 4 );

   temp_end[0] = reg_end;
    temp_end[2] = end >> 8;
    temp_end[3] = (uint8_t)end;

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp_end, 4 );

    return 0;


}

uint8_t hr_set_int_clk_div( uint8_t div )
{
    uint8_t reg = CLKDIV_PRF;
    uint8_t temp[4] = { 0 };
            temp[0] = reg;

    if( div > 7 )
        return -1;
```

```
        temp[3] = div;
        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );

        return 0;

    }

    uint8_t hr_set_offdac_settings( uint8_t offdac_pol_led2,uint8_t
offdac_set_led2,uint8_t offdac_pol_amb1,
    uint8_t offdac_set_amb1,uint8_t offdac_pol_led1,uint8_t offdac_set_led1
,uint8_t offdac_pol_amb2,uint8_t offdac_set_amb2)
    {
        uint8_t reg = DAC_SETTING;
                    unsigned long buffer = 0;
        uint8_t temp[4] = { 0 };
                temp[0] = reg;


        buffer |= ( offdac_pol_led2 << POL_OFFDAC_LED2 );
        buffer |= ( offdac_set_led2 << I_OFFDAC_LED2 );
        buffer |= ( offdac_pol_amb1 << POL_OFFDAC_AMB1 );
        buffer |= ( offdac_set_amb1 << I_OFFDAC_AMB1 );
        buffer |= ( offdac_pol_led1 << POL_OFFDAC_LED1 );
        buffer |= ( offdac_set_led1 << I_OFFDAC_LED1 );
        buffer |= ( offdac_pol_amb2 << POL_OFFDAC_LED3 );
        buffer |= ( offdac_set_amb2 << I_OFFDAC_LED3 );

        temp[3] |= buffer;
        temp[2] |= buffer >> 8;
        temp[1] |= buffer >> 16;


        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );

        return 0;

    }


    //Enables Registers to be read from setting bit in reg 0x00
    uint8_t hr_read_enable( void )
    {
        uint8_t reg = DIAGNOSIS;
        uint8_t temp[4] = { 0 };
                temp[0] = reg;

        temp[3] |= ( 1 << DIAG_REG_READ );

        I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS, temp, 4 );

        return 0;

    }

    uint8_t hr_read_disable( void )
    {
```

```
    uint8_t reg = DIAGNOSIS;
    uint8_t temp[4] = { 0 };
            temp[0] = reg;

    temp[3] &= ~( 1 << DIAG_REG_READ );

    I2Cwrite(I2C2_BASE,I2C_SLAVE_ADDRESS,temp, 4 );

    return 0;

}
```

Appendix B: MATLAB code

```
num1=xlsread('MYHRV.csv');
num_samples=length(num1);
x=num1(2,1:num_samples)';
n=num1(4,1:num_samples)';
n1=num1(5,1:num_samples)';
n2=num1(6,1:num_samples)';
t1 = linspace(0,length(x)/100,length(x));
plot(t1,n2)
%--------------------------------------------------------------------------
% Filtering
%--------------------------------------------------------------------------
% Filter Parameters

p       = 155;                % filter order
lambda  = .9999;              % forgetting factor
laminv  = 1/lambda;
delta   = 10.0;              % initialization parameter

% Filter Initialization
w       = zeros(p,1);       % filter coefficients
P       = delta*eye(p);     % inverse correlation matrix
e       = xnew*0;              % error signal
tic
for m = p:length(x)

    % Acquire chunk of data
    y = n(m:-1:m-p+1);

    % Error signal equation
    e(m) = x(m)-w'*y;
    g=w'*y;
    % Parameters for efficiency
    Pi = P*y;

    % Filter gain vector update
    k = (Pi)/(lambda+y'*Pi);

    % Inverse correlation matrix update
    P = (P - k*y'*P)*laminv;
```

```
    % Filter coefficients adaption
    w = w + k*e(m);

end
e2=e;
for m = p:length(e2)

    % Acquire chunk of data
    y = n1(m:-1:m-p+1);

    % Error signal equation
    e(m) = e2(m)-w'*y;

    % Parameters for efficiency
    Pi = P*y;

    % Filter gain vector update
    k = (Pi)/(lambda+y'*Pi);

    % Inverse correlation matrix update
    P = (P - k*y'*P)*laminv;

    % Filter coefficients adaption
    w = w + k*e(m);


end

e3=e;
for m = p:length(e3)

    % Acquire chunk of data
    y = n2(m:-1:m-p+1);

    % Error signal equation
    e(m) = e3(m)-w'*y;

    % Parameters for efficiency
    Pi = P*y;

    % Filter gain vector update
    k = (Pi)/(lambda+y'*Pi);

    % Inverse correlation matrix update
    P = (P - k*y'*P)*laminv;

    % Filter coefficients adaption
    w = w + k*e(m);

end


toc

%-------------------------------------------------------------------------
----

% Plot
```

```
%-------------------------------------------------------------------------
----

fs=100;
% Plot filter results
t = linspace(0,length(x)/fs,length(x));
figure;

%plot(t,x,t,e,t,num1.sig(1,:)');
plot(t,x,t,e);
title('Result of RLS Filter')
xlabel('Time (s)');
legend('Reference', 'Filtered', 'Location', 'NorthEast');
title('Comparison of Filtered Signal to Reference Input');
```

Appendix C: Health Watch Pictures

The following figures are of the finished health watch PCB in its 3d printed housing.

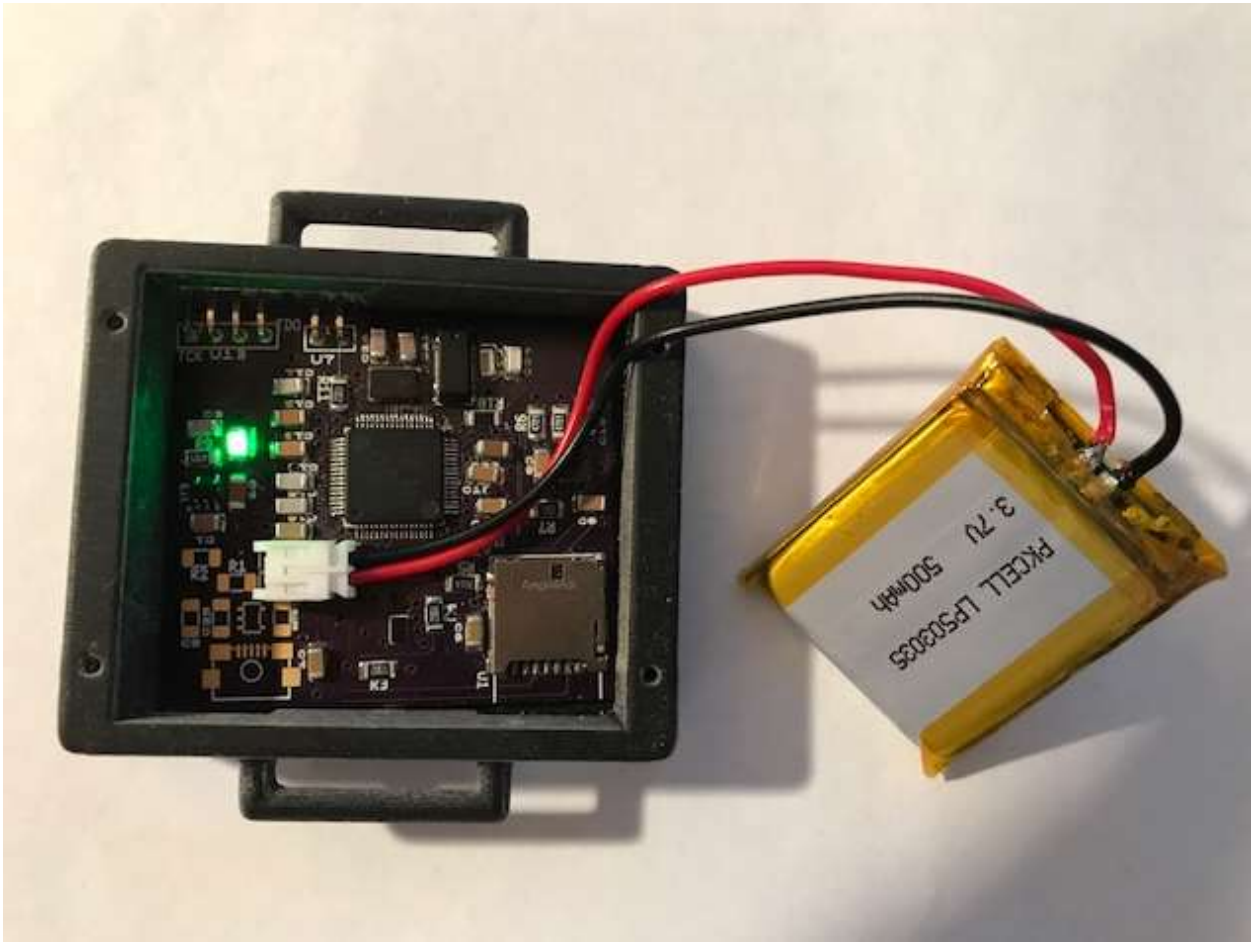

Figure 44 Health watch PCB and 3D printed housing

Figure 45 Health watch PCB housing bottom, with Sensor Array SFH7070