Methodology, Design, and Implementation of a Cardiac Pacemaker Prototype Using a Commercial Low Power Microcontroller

By

Sigfredo E. González Díaz

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

UNIVERSITY OF PUERTO RICO

MAYAGÜEZ CAMPUS

2006

Approved by:

Rogelio Palomera García, Ph.D. President, Graduate Committee	Date
Manuel Toledo Quiñónez, Ph.D. Member, Graduate Committee	Date
Eduardo Juan García, Ph.D. Member, Graduate Committee	Date
Pedro Resto, Ph.D. Representative of Graduate Studies	Date
Isidoro Couvertier Reyes, Ph.D. Chairperson of the Department	Date

ABSTRACT

Cardiac pacemakers are medical devices widely used to treat heart diseases. Modern pacemakers are ultra low power embedded systems with programmable functionalities, which include acquisition of heart signals samples and statistical histograms of paced, sensed and other events. This programmability is commonly offered by an internal custom-designed processor for the application. These custom-designed processors increase product cost and time to market. Research on pacemakers has been limited because of the lack of information available in open technical literature. This thesis presents a pacing system implemented in a general purpose low power microcontroller, the msp430f1611. Furthermore, a methodology for the development of the whole pacing system, including software flowcharts and control software source code is presented. Techniques to achieve low power consumption by the software are offered. The software consumes a maximum of 10 μA, with a typical value of 5 μA.

RESUMEN

Los marcapasos son dispositivos médicos usados ampliamente en el tratamiento de enfermedades cardiacas. Los marcapasos actuales son sistemas empotrados con ultra baja potencia y con funcionalidades programables. Algunas de estas funciones incluyen adquisición de muestras de la señal cardiaca e histogramas estadísticos de eventos específicos de la unidad. Esta programación es típicamente implementada por un procesador diseñado específicamente para esta aplicación en particular. Este procesador aumenta el costo del producto y el tiempo para salir al mercado. La investigación dentro del campo de marcapasos ha estado limitada debido a la falta de disponibilidad de documentos técnicos para el público en general. Esta tesis presenta un sistema de marcapasos implementado en un microcontrolador comercial, el msp430f1611. También se presenta una metodología para desarrollar todo el sistema, incluyendo flujogramas del programa y el código del mismo en lenguaje de ensamblador. Algunas técnicas son ofrecidas para alcanzar bajo consumo de potencia al ejecutar el programa. El programa consume 10 μA máximo, con un consumo típico de 5 μA.

DEDICATION

To Him who is the First and the Last, who died and came to life again.

To my parents, Sigfredo and Luz, for their support, guidance, encouragement and love.

To my sisters, Carmen and Idalia, for their patience and caring.

To my beloved Arlany, her unconditional love let me glimpse Eternity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

Introduction

Pacemakers are widely used today as a therapeutic tool to reestablish normal pacing of a diseased heart. A total of 300,000 are implanted every year [1], which indicates the popularity of these devices. Traditionally these devices have been powered by a battery to be implanted in a patient for a long period of time. For this reason a pacemaker should have very low power consumption and at the same time wide functionalities.

A pacemaker task is not computationally intensive, since a heart typically beats at 70 beats per minute and any general purpose microcontroller with a frequency of 500 kHz or above can accomplish this. However, pacemaker systems need very low power consumption microcontrollers. Typically pacemaker companies use custom-designed processors to implement pacemaker's programmability. Custom-designed processors increase the cost of the system significantly and also increase the time to market. A few years back a survey for a general purpose processor to be used for the design of a pacemaker was conducted [2]. However, the survey did not found any microcontroller that satisfies the requirements of a pacemaker application. Currently, advancements in technology have made new microcontrollers available with ultra low power consumption that can satisfy power constrains for the pacemaker

design. If pacing software can be developed using a general purpose microcontroller, it will reduce significantly cost and time to market. However, as noted in [3], there is little available information in open technical literature regarding the design details of cardiac circuits and even less, or non-existent, for control software source codes. This is probably due to the "closed" characteristics of the pacemaker industry (few companies with a long tradition) and the high competition among these companies.

This research intent to contribute to open technical literature by developing a design of a cardiac pacemaker control software. This software is implemented in a low power general purpose microcontroller. For this reason an extensive review of current low power techniques for software design was performed.

This thesis is organized as follows: An broad discussion of a cardiac pacemaker, including the electrical physiology of the heart and previous research in cardiac pacemaker is presented in Chapter 2. Chapter 3 discuss general purpose microcontrollers with emphasis in important aspects for low power consumption like memory organization, CPU architectural configurations, and instruction set philosophies. Chapter 4 presents a methodology for the development of a pacing system, including a definition of pacemaker programmable parameters along with a comparison of available microcontroller and their low power performance. Chapter 5 and Chapter 6 presents the design and implementation of the pacing system in a general purpose low power microcontroller, the msp430f1611. Finally some important remarks are made at the conclusion in Chapter 7.

CHAPTER 2

Implantable Cardiac Pacemakers

The cardiac pacemaker is one of the greatest inventions of the last decades. The concept it brought to the mind of inventors, scientist, engineers, and doctors was revolutionary: to aid the human body using microelectronic technology. Since the first implantable pacemaker appeared, many different implantable devices have been developed to encourage a higher quality of life in patients with disabilities. Cardiologists around the world have used cardiac pacemakers as an efficient treatment to patients suffering from bradycardia. Furthermore, implantable pacemakers have helped to expand the knowledge of the electrical behavior of the heart by acquiring intracardiac electrograms.

This chapter introduces the biological and electrical behavior of the heart. The discussion intends to give the reader a more comprehensive idea of the end purpose of the pacemaker device. In addition, a historical background of the development of the pacemaker and previous research will be presented. Finally, the approach followed for this research is discussed.

2.1 The human heart

The human heart is normally situated slightly to the left of the middle of the thorax, underneath the sternum (breastbone). It is enclosed by a sac known as the pericardium and is surrounded by the lungs. In normal adults, it weighs between 250 g and 350 g, but extremely diseased hearts can weigh up to 100 g.

2.1.1 The biological behavior of the heart

The heart has four chambers: two atria (singular: atrium) and two ventricles. Figure 2.1 shows all the components of the heart.

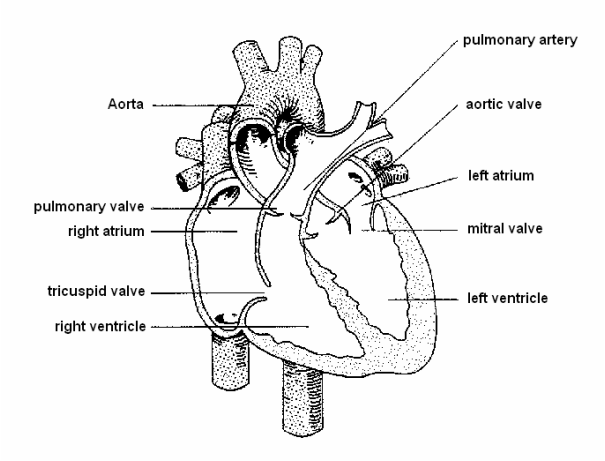


Figure 2.1. The Human Heart.

Oxygen-depleted or deoxygenated blood from the body enters the right atrium through two great veins, the superior vena cava which drains the upper part of the body and the inferior vena cava that drains the lower part. The blood then passes through the tricuspid valve to the right ventricle. The right ventricle pumps the deoxygenated blood to the lungs, through the pulmonary artery. Gaseous exchange takes place in

the lungs where the blood releases carbon dioxide into the lung cavity and picks up oxygen. The oxygenated blood then flows through pulmonary veins to the left atrium. From here, this newly oxygenated blood passes through the mitral valve to enter the left ventricle, which pumps the blood through the aorta to the entire body. Even the lung takes some blood supply from the aorta via bronchial arteries.

The left ventricle is much more muscular (1.3 - 1.5 cm thick) than the right one (0.3 - 0.5 cm thick) as it has to pump blood around the entire body, which involves exerting a considerable force to overcome the vascular pressure. The right ventricle needs to pump blood only to the lungs, so it requires less muscle.

The contractile nature of the heart is due the presence of cardiac muscle in its wall, which can continuously work without fatigue. The heart wall is made of three distinct layers. The first is the outer epicardium which is composed of a layer of flattened epithelial cells (cells that most of the internal organs of the body) and connective tissue. Beneath, a much thicker myocardium made up of cardiac muscle exists. The endocardium is a further layer of flattened epithelial cells and connective tissue which lines the chambers of the heart. The blood supply to the heart itself is supplied by the left and right coronary arteries, which branch off from the aorta.

The function of the heart is to pump blood around the body. Every single beat of the heart involves a sequence of events known as the cardiac cycle, which consists of two major stages: cardiac diastole and cardiac systole. Basically the diastole is the period of time when the heart relaxes and refills with blood. The systole is the period of time when the heart contracts to push out blood to the different parts of the body. Figure 2.2 shows the cardiac cycle of a healthy heart (75 beats per minute $\approx 800ms$) with reference to the electrocardiogram (ECG). The ECG is a signal generated by the electric conduction of the heart. Section 2.1.2 will discuss the ECG in more detail.

The picture presented in Figure 2.2 displays the cardiac cycle divided in 7 phases. The three first phases comprise the systole, i. e., (1) atrial contraction, (2) isovolumetric contraction, and (3) rapid ejection. On the other hand, phases (4) reduced ejection, (5) isovolumetric relaxation, (6) rapid filling, and (7) reduced filling, joint to form the diastole.

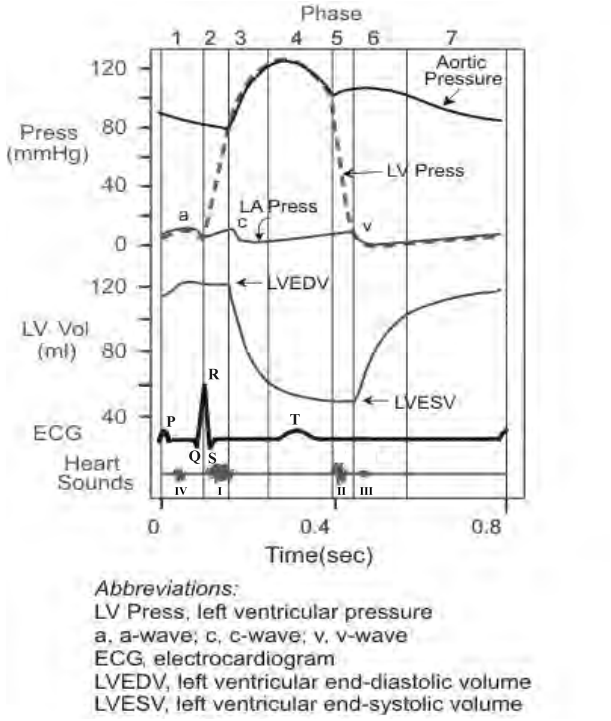


Figure 2.2. Cardiac Cycle of a Human Heart [4].

In the lower part of Fig. 2.2 there is a curve known as Electrocardiogram (ECG); the figure shows one cycle. This curve determines the different phases. Each phase produces anatomical and electrical changes to the heart, as described next.

- Phase 1 - atrial contraction

  This phase starts the cardiac cycle and covers what is denoted as the p-wave on the ECG, going from a small peak to the start of the large spike. As the atria contracts, the pressure within the atrial chambers increases causing a rapid

flow of blood into the ventricles due to difference of pressure between the two chambers.

- Phase 2 - isovolumetric contraction

  This phase is determined by the QRS complex. The QRS complex is the large spike on the ECG and it causes the ventricle to contract. This contraction causes an abrupt rise in pressure causing the A-V valves to close as intraventricular pressure exceeds atrial pressure. Closure of the A-V valves results in the First Heart Sound (S1)depicted in Figure 2.2. This sound is normally split ($\approx 0.04sec$) because mitral valve closure precedes tricuspid closure (see Figure 2.1).

- Phase 3 - Rapid Ejection

  When the intraventricular pressures exceed the pressures within the aorta and pulmonary artery, the aortic and pulmonic valves open and blood is ejected out of the ventricles. Maximal outflow velocity is reached early in the ejection phase, and maximal aortic and pulmonary artery pressures are achieved. In this phase the ECG stays flat since all the anatomical changes are due to pressure differences.

- Phase 4 - Reduced Ejection

  Approximately 150-200 milliseconds (ms) after the QRS, ventricular repolarization occurs, yielding a small wave in the ECG, known as the T-wave. Although ventricular pressure falls slightly below outflow tract pressure, the outward flow still occurs due to inertial energy of the blood.

- Phase 5 - Isovolumetric Relaxation

  Isovolumetric relaxation occurs when pressure of the ventricles is less than the

pressure of the outflow tracks. At this point, the aortic and pulmonic valves abruptly close (first aortic, then pulmonic) generating the Second Heart Sound (S2). It is called isovolumetric because the volume in the ventricles remains constant.

- Phase 6 - Rapid Filling

  As the ventricular pressures fall below atrial pressures, the AV valves open and the ventricular starts filling up. The ventricles continue to relax despite the inflow, which causes intraventricular pressure to continue falling down. This phase produces the Third Heart Sound (S3).

- Phase 7 - Reduced Filling

  The ventricles continue to fill with blood and start expanding out and the intraventricular pressures rise up. This reduces the pressure difference across the atrioventricular valves so that the filling rate falls down.

The rhythmic sequence of contractions is coordinated by the sinoatrial (SA) and atrioventricular (AV) nodes located at the upper and lower walls, respectively, of the right atrium. The SA node, often known as the cardiac pacemaker, is responsible for the wave of electrical stimulation that initiates atria contraction. Once the wave reaches the atrioventricular node, it is conducted through the "bundles of His" and causes contraction of the ventricles. The time that takes for the wave to travel from the SA nerve to the AV node creates a delay between contractions of the two chambers and ensures that each contraction is coordinated simultaneously throughout the heart. In the event of severe pathology, the Purkinje fibers can also act as a pacemaker. Nevertheless, this is usually not the case because the rate of spontaneous firing of the Purkinje fibers is considerably lower than that of the other pacemakers, and hence is overridden.

2.1.2 The electrical behavior of the heart

The contractions of the heart are controlled by electrical impulses. These electrical impulses fire at a rate which controls the beat of the heart. The cells that create these rhythmical impulses are called pacemaker cells, and they directly control the heart rate. If damage to the body's intrinsic conduction system occurs, an artificial device also called pacemaker can be used to produce these impulses synthetically.

Although all of the heart's cells possess the ability to generate these electrical impulses or action potentials, a specialized portion of the heart called the SA node is responsible for the whole heart's beat.

The SA node is composed of a group of modified cardiac myocytes. Myocytes possess some contractile filaments, though they do not contract. Cells in the SA node will naturally discharge at about 70-80 min^{-1}. Since the SA node is responsible for the rest of the heart's electrical activity, it is sometimes called the primary pacemaker. Figure 2.3 shows the location of the SA node relative to the whole conduction network of the heart. If the SA node does not function, or the impulse generated in the SA node is blocked before it travels down the electrical conduction system, a group of cells further down the heart will become the heart's pacemaker. These cells form the atrioventricular node (AV node), which can be seen in Figure 2.3.

The cells of the AV node normally discharge at about 40-60 min^{-1}, and are called the secondary pacemaker. Further down the electrical conducting system of the heart, the left and right branches of the Bundle of His, and the Purkinje fibers, will also produce a spontaneous action potential if they are not inhibited by other electrical activity. These tertiary pacemakers fire at a rate between 30-40 min^{-1}. Even individual cardiac muscle cells can contract rhythmically.

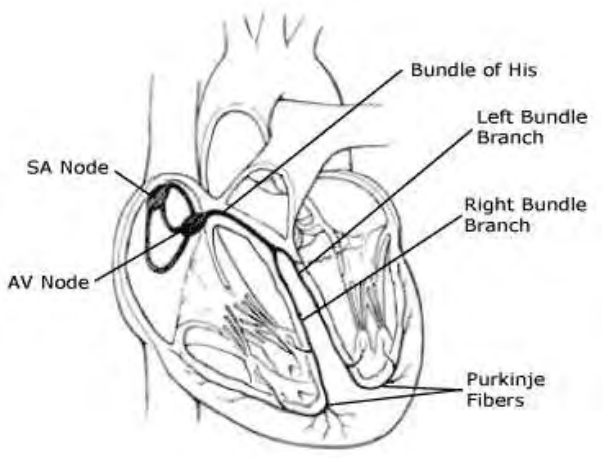


Figure 2.3. Electrical conduction system of the heart.

The reason the SA node controls the whole heart, is that its action potentials are released most often, triggering other cells to generate their own action potentials. In the muscle cells, action potentials will produce contraction. The action potential generated by the SA node passes down the cardiac conduction system and arrives before the other cells have had a chance to generate their own spontaneous action potential. This is the normal conduction of electrical activity within the heart.

All living cells have different concentrations of ions, particularly Sodium (Na+), Potassium (K+), Chlorine (Cl-), and Calcium (Ca++), across the cell membrane. There are also impermeable negatively charged proteins within the cell. This different concentration of ions produce bioelectric potentials [5]. Bioelectric potentials consist of a resting potential, and when appropriately stimulated, an action potential. An individual excitable cell maintains a steady electrical potential difference between its internal and external environments. This resting potential at the internal medium is in the range of 50mV to 100mV, relative to the external medium. The membrane potential (Vm) at which this steady state exists is called equilibrium potential. For potassium, E_k is measured in volts and calculated from the Nernst Equation (2.1).

$$E_k = \frac{RT}{nF}\, ln\frac{[K]_o}{[K]_i} = .0615\; log_{10}\frac{[K]_o}{[K]_i} \quad (V) \tag{2.1}$$

The active state is initiated when an adequate stimulus is used to excite the cells. An adequate stimulus is one that causes depolarization in a membrane and exceeds the threshold potential. Figure 2.4 shows the electrochemical process that occurs when an adequate external stimulation is provided to the cells. The stimulus should be applied for the right period of time, with enough strength and with the right polarity. Without stimulation, the cells are at the rest potential. Since ions can travel from and into the cells through the membrane, when stimulus is applied a diffusion process takes place causing a current flow. The magnitude of this current is in the order of the pico Amperes. After a finite period of time (typically 1 ms or so) the diffusion of ions decreases the membrane potential until it passes the threshold voltage. At that point the cells undergo total depolarization and a chain reaction of action potentials is developed.

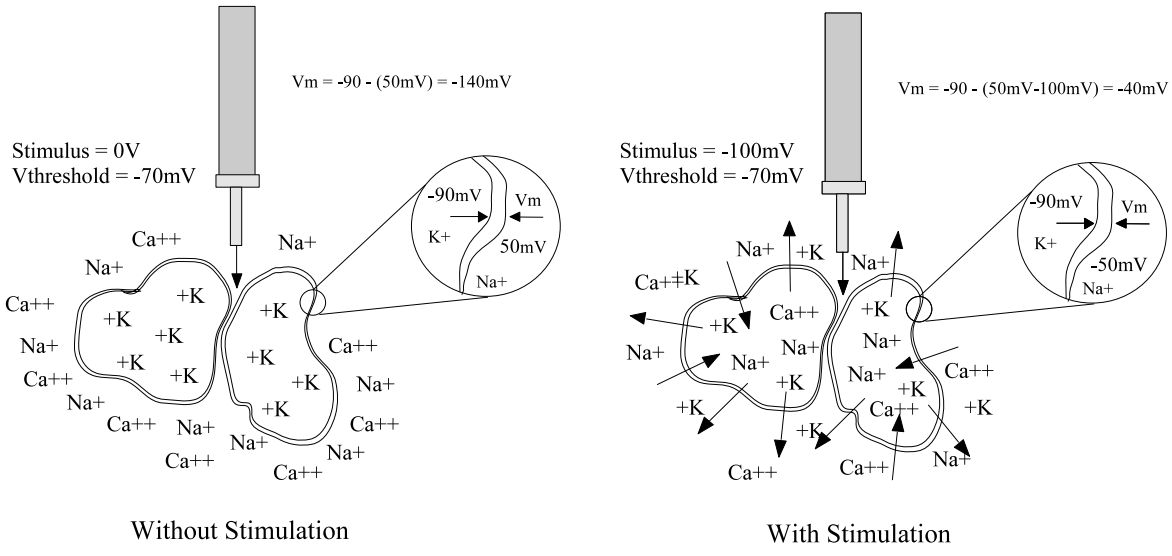


Figure 2.4. Artificial Stimulation of the heart.

There are two types of action potentials in the heart: The non-pacemaker potentials and pacemaker cells potentials. The non-pacemaker potentials are found throughout the heart except on the SA-node. The non-pacemaker potentials are generated by the more abundant cells in the heart, ventricular myocytes and Purkinje cells. Non-pacemaker cells have an actual resting potential, which means that the membrane equilibrium is achieved. Figure 2.5 presents the 4 phases that form the complete action potential. When undisturbed, the cell is in phase 4 known as the resting state or equilibrium state. When disturbed, the cells enter in a rapid depolarization known as phase 0. The name depolarization is due to the fact that at the equilibrium state the cell is actually polarized by the potassium, calcium and sodium ions creating the membrane potential (Vm). As Figure 2.5 shows, this depolarization has a steep

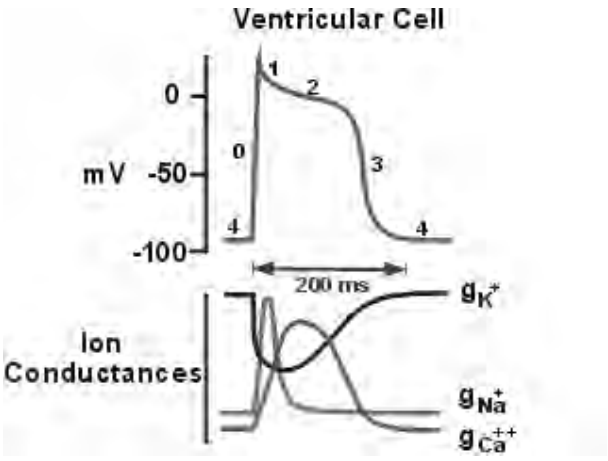


Figure 2.5. Ventricular Action Potential [4].

slope with a slew rate greater than 150V/s. This rapid depolarization is caused by Na+ diffusing rapidly into the cell. This influx causes the counter reaction of K+ ions which try to bring back the cell into its initial state. This is shown as phase 1. However, there is another component that causes depolarization. Although at a slower rate, Ca++ help in the depolarization process of the cell. Since Ca++ diffuses

slowly, it shapes the plateau in phase 2. Even though Na+ and Ca++ maintain the cell depolarized for some time, K+ gets to develop such a rapid diffusion that it establishes the rest potential again, closing the cycle. Consequently any stimulation during phase 0, 1, 2 and part of 3 will be unsuccessful. For this reason this period of time is called the Effective Refractory Period (ERP).

On the other hand, the pacemaker potentials are generated by SA cells in the SA node. These cells are characterized as having no true resting potential, generating regular and spontaneous action potentials. Figure 2.6 illustrates the behavior of the

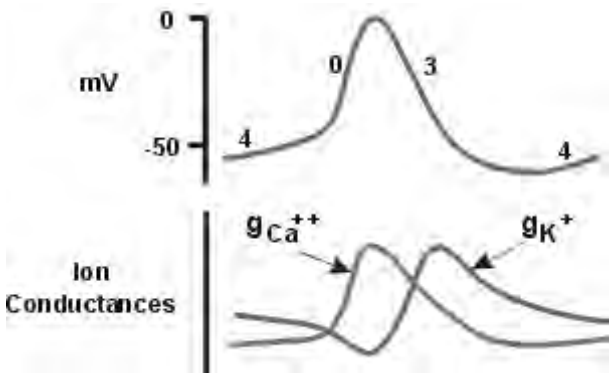


Figure 2.6. SA node Action Potential [4].

SA cells. It should be noticed that pacemaker cells do not have neither phase 1 (atrial contraction) or phase 2 (isovolumetric contraction). The electrochemical mechanism in this case is only comprised by K+ and Ca++. Due to the absence of Na+, these cells cannot achieve a rest potential. This is due to the fact that equilibrium between K+ and Ca++ is very difficult to attain.

The combination of all these action potentials produces the peculiar wave called ECG (electrocardiogram), shown in Figure 2.7. This is the ECG for the normal heart.

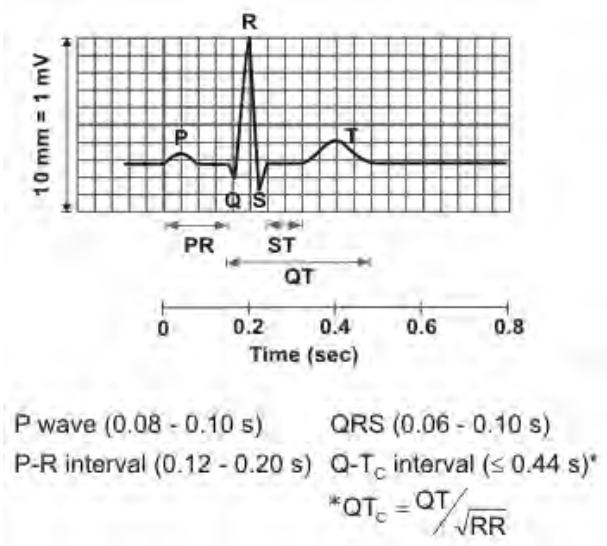


Figure 2.7. Typical electrocardiogram of a normal heart.

The ECG is due to wave interferences in the electrical path of the heart. P, Q, R, S, and T are used to identify the different electrical time intervals of the heart. The description for the time intervals is given below:

- P wave

  Identify the wave of depolarization that spreads from the SA node throughout the atria. It typically has 80-100 ms in duration.

- P - R interval

  It normally lasts from 120 to 200 ms. It represents the time between the onset of atrial depolarization and the onset of ventricular depolarization. If the P-R interval is >0.2 sec, there is an AV conduction block. It is also termed as a first-degree heart block if the impulse is still able to be conducted into the ventricles.

- QRS complex

  The QRS complex represents the ventricular depolarization. It is the most

prominent amplitude of the ECG. It can be used to diagnose bundle branch blocks or abnormal pacemaker site located in the ventricles. This can be detected when the QRS complex is prolonged above 100ms.

- ST segment

  The ST segment is measured from the onset of the S wave to the onset of the T wave. The T wave represents the repolarization of the ventricles. The ST segment is the time at which the entire ventricle is depolarized. The ST segment is important in the diagnosis of ventricular ischemia or hypoxia because under those conditions, the ST segment can become either depressed or elevated.

- QT interval

  The Q-T interval represents the time for both ventricular depolarization and repolarization to occur. Therefore, it roughly estimates the duration of an average ventricular action potential. This interval can range from 200 to 400 ms. In practice, the Q-T interval is expressed as a "corrected Q-T (QTc)" by taking the Q-T interval and dividing it by the square root of the R-R interval. The R-R is the time between two consecutive R waves. This nomenclature allows an assessment of the Q-T interval that is independent of heart rate.

The ECG is a fundamental tool in cardiology and electrophysiology. It is obtained by measuring the currents that flows through the body due to consecutive polarization/depolarization cycles. This current can be measured because the body acts as a volume conductor. Surface electrodes are used to measure the generated ECG, although sometimes intracardiac electrodes are used to monitor more severe conditions. The ECG is valuable to obtain information concerning heart diseases like Bradycardia, Arrhythmias, and Tachycardia.

2.1.3 Diseases of the heart

The study of diseases of the heart is known as cardiology. Important diseases of the heart include:

- Coronary heart disease - is the lack of oxygen supply to the heart muscle; it can cause severe pain and discomfort known as Angina.
- Heart attack - occurs when heart muscle cells die because blood circulation to a part of the heart is interrupted.
- Congestive heart failure - is the gradual loss of pumping power of the heart.
- Endocarditis and myocarditis - are inflammations of the heart.
- Congenital heart defects.
- Cardiac arrhythmia - is an irregularity in the heartbeat. It is sometimes treated by implanting an artificial pacemaker.

Cardiovascular diseases (CVD) accounted for 38.5 % of all deaths, or 1 of every 2.6 deaths, in the United States in 2001. CVD mortality was about 60 percent of "total mortality". This means that of over 2,400,000 deaths from all causes, CVD was listed as a primary or contributing cause on about 1,408,000 death certificates [6].

In 2003, Arrhythmias (disorders of heart rhythm) were the cause of 37,892 deaths. The total mentioned mortality was 484,000 of over 2,400,000 U.S. deaths. Arrhythmias are often treated using pacemakers, which means that there is a high demand for these devices.

2.2 History of implantable cardiac pacemakers

Since their commercial introduction in 1961, approximately 3.4 million pacemakers have been implanted worldwide, and approximately 600,000 units were implanted in 2000 alone [7]. Of the more than ten million people worldwide who suffer from cardiac arrhythmias, only a small percentage receives proper medical attention. Approximately 300,000 - or only 3% - are fortunate enough to be evaluated by electrophysiologists - cardiologists who specialize in cardiac rhythm disorders [8]. These statistics clearly reveal that there is a need of research in order to help in the development of more accessible treatment for all patients. This was the motivation behind the origins of the pacemaker. This section discusses the development of the electrical study of the heart focusing on the origins of the pacemaker as a therapeutic tool.

In 1902 Willem Einthoven applied the string galvanometer to the measurement of the electrical potentials generated by the beating heart. Einthoven demonstrated that these electrical potentials could be detected from electrodes placed on the surface of the body. This study opened the possibility of quantifying and displaying the typical signals of a normally-beating heart together with those produced by cardiac arrhythmias. Furthermore, it led to the development of instruments for recording the ECG and to the whole modern science of electrocardiography.

The first experimental heart pacemaker was designed by Hyman in New York. He developed a device in which a needle was passed through the intact chest wall into one of the top chambers of the heart. This produced an interrupted current, that is, a pulsed rather than 'galvanic' or continuous current, as it had been used in earlier works. This device allowed prolonging the lives of two patients for 24-48 hours in 1932.

In 1950 two Canadians, Bigelow and Callaghan, presented a paper describing their work on stimulation of dog hearts with one electrode in the esophagus and the other over the precordium. During the same year they stimulated the sinoatrial node of a patient endovenously during open-heart surgery.

Two years later Zoll, building on the work of Bigelow, Callaghan, and Hyman, published an article entitled "Resuscitation of the heart in ventricular standstill by external electric stimulation". Zoll's system used plates held on the chest wall by straps avoiding the dangers associated with methods involving surgery. The main drawback from this method is that it could not be used for long-term pacing since it produced many undesirable effects including skin burns, pain and contraction of skeletal muscles in the chest.

In 1958 Senning and Elmqvist, in Sweden, developed a pacemaker that was able to run from batteries and was small enough for implantation. This pacemaker is shown in Figure 2.8. The batteries of this unit were nickel-cadmium and could be recharged inductively. The first unit was implanted in a patient in October 1958 [9].

Figure 2.8. The first cardiac pacemaker, 1958 [10].

In America in 1959, Chardack and Greatbacth developed the first implantable pacemaker using mercury zinc oxide cells. It measured 6 cm in diameter by 1.5 cm thick.

It contained a blocking oscillator generating a very low power pulse which triggered a transistor switch. This drained a capacitor charge from the power supply to deliver a 1 ms biphasic pulse to the electrodes. The current drained from the unit was 11 μA, which gave the batteries a usable estimated life of 5 years. Warning of low battery state was provided by a slow rise in the pulse rate over a period of weeks. Figure 2.9 illustrates the circuit schematic of the Greatbatch-Chardack implantable pacemaker. From this first approach in the design of pacemakers, several improvements have taken place. The new developments in hardware implementations are discussed in the next section.

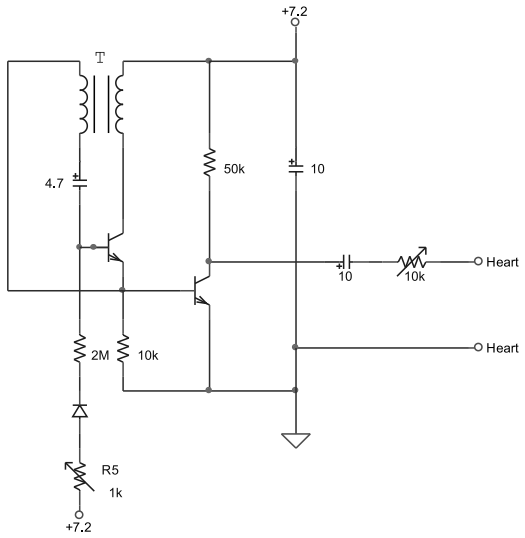


Figure 2.9. Circuit diagram of the Greatbatch - Chardack implantable pacemaker. (Redraw from Chardack et al. (1964). Ann. N.Y. Acad. Sci.111, 1075-1092.).

2.3 Definition of a cardiac pacemaker

A pacemaker or "artificial pacemaker", not to be confused with the heart's natural pacemaker, is a medical device designed to regulate the beating of the heart. The pur-

pose of an artificial pacemaker is to stimulate the heart when either the heart's native pacemaker is not fast enough or there are blocks in the heart's electrical conduction system, preventing the propagation of electrical impulses from the native pacemaker to the ventricles. In general, pacemakers are not used to treat fast rhythms of the heart.

In most cases, the indication for permanent pacemaker placement is a slow heart rate (bradycardia) or a defect in the electrical conduction system of the heart (heart block) with associated symptoms. Typical symptoms of a slow heart rate include light-headedness, poor exercise tolerance, and loss of consciousness. Asymptomatic individuals who have a slow heart rate but do not require a pacemaker, like athletes, typically have resting heart rates in the 40 bpm without any hurtful effects.

If the slow heart rate is due to complete heart block, a pacemaker is indicated, since the heart rate can dramatically decrease without notice. Pacemakers can also be placed in patients at high risk for complete heart block.

Rarely, in people prone to ventricular fibrillation, a slow rhythm in the heart can lead to a ventricular fibrillation. For them, preventing the slow rhythm can prevent ventricular fibrillation.

Modern pacemakers have two basic functions, monitoring and stimulation. A pacemaker "listens" to the heart's native electrical rhythm, and if it does not sense any electrical activity within a certain period of time, it stimulates the heart with a preset amount of energy, typically measured in Joules.

There are three modes of pacemaker's operation [11]:

- *freerunning* (fixed rate or asynchronous) - is insensitive to any rhythm that may develop in the paced chamber.

- *inhibited* - senses cardiac activity and does nothing if this is present, but deliver a stimulus after an elapsed time if no further cardiac activity occurs to inhibit operation.

- *triggered* - senses activity and delivers a stimulus in a desired way.

The North American Society of Pacing and Electrophysiology (NASPE) and the British Pacing and Electrophysiology Group (BPEG) generic code is a pacemaker naming convention originally developed in 1974 that uses a 3 - 5 letter code to describe the main features of an artificial pacemaker. Table 2.1 presents the nomenclature used by NASPE to classify pacemakers [12].

Table 2.1. The Revised NASPE/BPEG Generic Code for Antibradycardia Pacing.

Position	Category	Code
I	Chamber(s) Paced	O = None
		A = Atrium
		V = Ventricle
		D = Dual (A+V)
II	Chamber(s) Sensed	O = None
		A = Atrium
		V = Ventricle
		D = Dual (A+V)
III	Response to Sensing	O = None
		T = Triggered
		I = Inhibited
		D = Dual (T+I)
IV	Rate Responsive	O = None
		R = Rate Modulation
V	Multisite Pacing	O = None
		A = Atrium
		V = Ventricle
		D = Dual (A+V)

Each of the 5 positions implies a particular aspect of the pacemaker's functionality.

Using this scheme, a designation of VATOO would describe, for example, a pacemaker that sensed the atria and paced the ventricles in a triggered mode with no rate response or multi-site pacing.

Patients with sick sinus syndrome and chronotropic incompetence, some of those in whom the atrioventricular (AV) node has been ablated because of intractable supraventricular arrhythmias, and patients with chronic atrial fibrillation and complete AV block, cannot increase their heart rate. These patients can only increase their cardiac output through an increase in stroke volume. A rate responsive pacemaker system, which detects the need for a rise in heart rate, will increase the exercise capability and the quality of life of these patients.

A rate responsive pacemaker is designed to adjust the lower rate of the pulse generator based on the output signal of the sensor. Recently, several other uses of the sensor have been proposed such as capture detection, detection of tachycardias, rejection of interference, and upper rate behavior adaptation.

2.4 Previous pertinent research in cardiac pacemakers

Since the conception of the first pacemaker there has been a great amount of research to improve it. Improvements can be divided in three main areas: (1) Hardware Implementation, (2) Control Algorithms and (3) Pacemaker's Battery Efficiency. This section revises previous reseach that has been the foundation for this thesis and discusses it briefly.

2.4.1 Hardware implementations

At the beginning, the design of a pacemaker was only at the hardware level. The first generation of pacemakers was a composition of discrete electronic devices that served as therapeutic tools. The first steps toward miniaturization were taken in 1970 by Gerhard Weil, Walter L. Engl, and Albrecnt Renz [13].

They proposed a design based on a thyristor, which is a four-layer structure, to control the output of a relaxation oscillator that served as the pacemaker. Their circuit is shown in Figure 2.10. It should be mentioned that this design used a power supply of 7V and generated an asynchronous 1.5ms width pulse. The total current consumption of this system was 18μA without load. Their design is important for its contribution toward miniaturization and implementation of pacemakers using integrated circuits. It also includes safety features that helped in the development of reliable pacemaker designs.

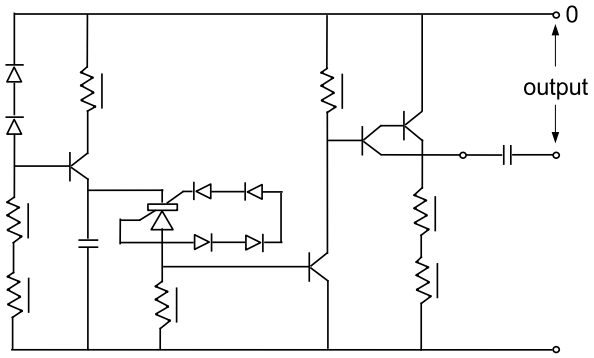


Figure 2.10. Circuit schematic of the design proposed by Gerhard Weil, Walter L. Engl, and Albrecnt Renz in 1970 [13].

In 1979 Robert A. Walters and Gary W. Bivins, at Arco Medical Products Company, developed a cardiac pacemaker that included digital elements to achieve a programmable pacemaker [14]. Their main concern was about the physiological change

that suffers the patient's heart. Further, the impedance built up in the heart-lead interface changed with time causing loss of capture to the pacemaker. To solve these problems they designed a circuit that could be programmed externally by means of electromagnetic waves. Figure 2.11 illustrates how the programmability of the design was done.

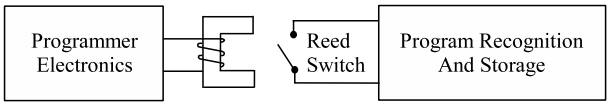


Figure 2.11. Block diagram of the programming of the pacemaker [14].

The first approach to design a pacemaker using a microcontroller was completed in 1989 by Larry J. Stotts, K. Ross Infinger, Janet Babka, and David Genzer [15]. Their paper presents a custom design of an 8-bit microcontroller for pacemaker applications. It also includes a detailed description of the design of the sense amplifiers. The ROM and RAM memories are used as transition registers for programmability purpose. Figure 2.12 presents the block diagram of the pacemaker.

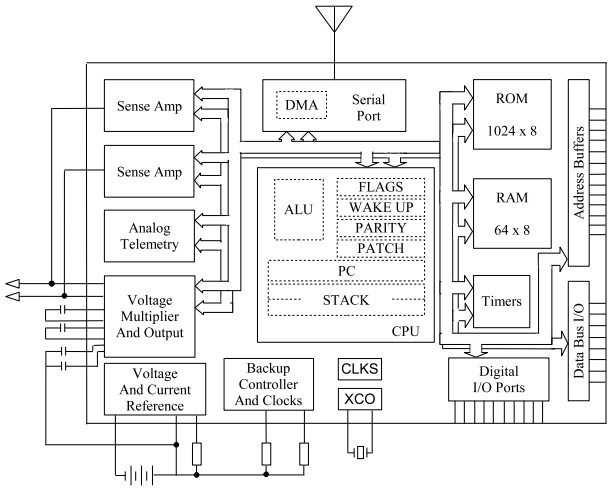


Figure 2.12. Block diagram of the circuit design by by Larry J. Stotts, K. Ross Infinger, Janet Babka, and David Genzer [15].

The last approach used for hardware implementation was taken by Wen-Yaw Chung, Heh-Sen Lin, Chung-Huang Yang, Tai-Ping Sun, Guo-Ching Chen, and Chang-Horng Hsieh in 1995 [16].

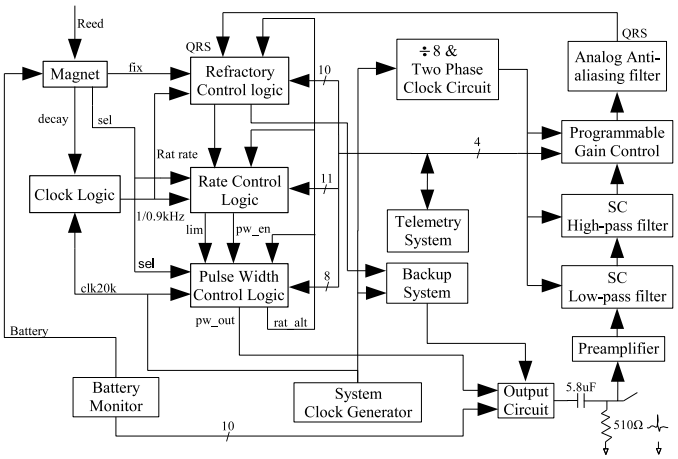


Figure 2.13. Modular-based block diagram of the overall VVI pacemaker [16].

They proposed a demand-type VVI mode research prototype pacemaker partitioned in three modules: digital processing, analog processing and master clocking module. Figure 2.13 shows the block diagram of the modular design proposed in their paper.

It is useful to complement the hardware with software to have a more flexible control of the parameters of the pacemaker. The software enhances the capabilities of the pacemaker, making it a diagnostic tool besides the typical therapeutic characteristics. The next section will discuss this area more in detail.

2.4.2 Control algorithms

The first generation of pacemakers did not have any kind of control algorithm. Gradually, technology improved the way to approach the design of a pacemaker. Memories and registers were included to implement logical operations. These pro-

grams constitute the control algorithms that give functionality and adaptability to a pacemaker in order to satisfy the needs of each individual patient. In the following paragraphs a brief description of the control algorithms that have been previously studied will be presented in order of relevance.

In 1998, Jungkuk Kim and Paul Haefner proposed an algorithm for an automatic gain control (AGC) that improved sensing performance and minimized human intervention [17]. The basic idea of the algorithm is relatively simple as shown in Figure 2.14. When three out of four of the previous peak filtered (10-100 Hz bandpass) electrogram amplitudes are equal to or higher than an upper limit, the sense amplifier gain decreases by 1.25 times.

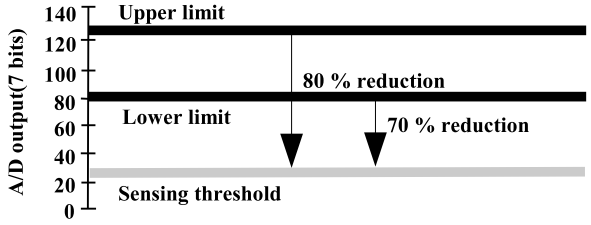


Figure 2.14. The AGC upper and lower limits with sensing threshold [17].

In contrast, when three out of four previous peak amplitudes are lower than a lower limit, the gain increases by 1.25 times. The results of the sensing performance of both methods (optimal threshold vs. AGC) are shown in Figure 2.15.

The percentage of malsensing for the automatic sensing algorithm is 0.33% (0.19% over-sensing and 0.14% under-sensing) for 3585 beats, compared to .45% (0% over-sensing and .45% under-sensing) using the conventional method with an optimal sensing threshold.

In 1994, R. Frohlich, A. Bolz, R. Hardt, M. Hubmann, and M. Schaldach proposed an

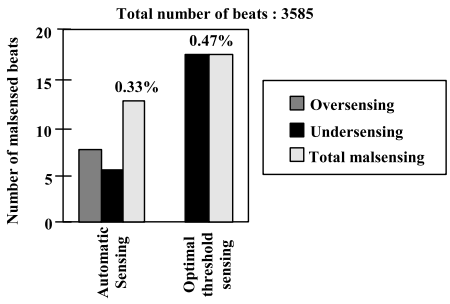


Figure 2.15. Performance of the automatic sensing method and the optimal threshold sensing method [17].

algorithm to automatically adjust the amplitude of the output stimulus in pacemakers [18]. They call their algorithm the Automatic Amplitude Adjustment (AAA). The basic idea of the AAA algorithm was that the effectiveness of each pulse is checked immediately after it has been released. If the stimulus is effective, the pacemaker works normally and nothing is changed. Otherwise, the pacemaker increases the output voltage to guarantee safe pacing above threshold.

In 1993, D.B. Shaw and M. Horwood, explained the significance of having recording pacemakers as a diagnostic tool for physicians [19]. They outlined some useful applications for recording pacemakers. Theoretically, ventricular pressure and cardiac output would seem to be most useful, but mixed venous oxygen saturation or central venous temperature would provide additional information as might indirect measurements, such as impedance for cardiac stroke volume or respiration and acceleration sensing for body activity.

There is a special area of programmed pacemakers, known as rate responsive pacing. Rate responsive pacing is an effort of pacemaker's designers to create intelligent pacemakers that can adapt to the changeable ambient of the human body. In the following subsection the major contributions is presented.

Rate responsive pacemakers

In 1989, M. Schaldach, proposed a parameter for rate responsive pacing [20]. He stated that the Preejection Period (PEP) get shorter under physical and emotional stress. This shortage reflects the associated sympathetic response that caused the heart rate to increase when sinus function is normal. Figure 2.16 defines the Preejection Period as the interval beginning at the start of ventricular depolarization to the onset of ventricular ejection. The period ends with the opening of the pulmonary and aortic valves (see Figure 2.2).

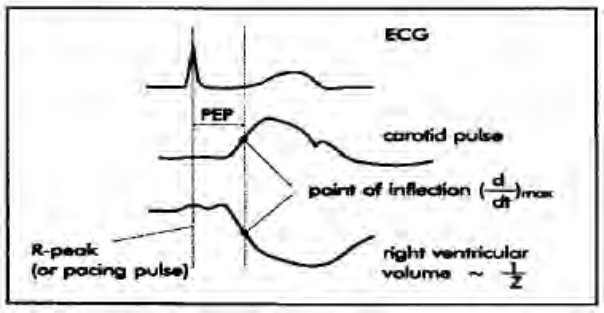


Figure 2.16. Definition of the Preejection Period [20].

As shown in Figure 2.17, systolic time intervals (STI), particularly PEP, serve as physiological parameters which can be used for the control of pacing rate [20]. From the two intervals comprising PEP, the isovolumetric contraction time is physiologically more important since it is a direct reflection of the speed of ventricular contraction and, hence, sympathetic tone.

The block diagram of the design proposed by Schaldach to implement the PEP algorithm is shown in Figure 2.18. The design included sensing and pacing channels, a backup timer and a telemetry module. A special circuitry to measure and determine PEP was also included. A microcontroller circuit with CPU, ROM, RAM, clock generation, watchdog timer, and interrupt capability was used as the central processing

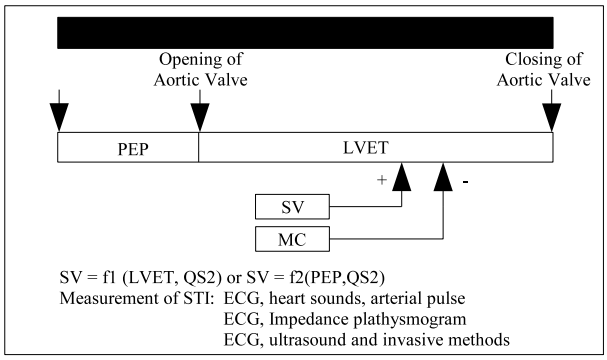


Figure 2.17. Systolic Time Interval (STI).

unit of the system.

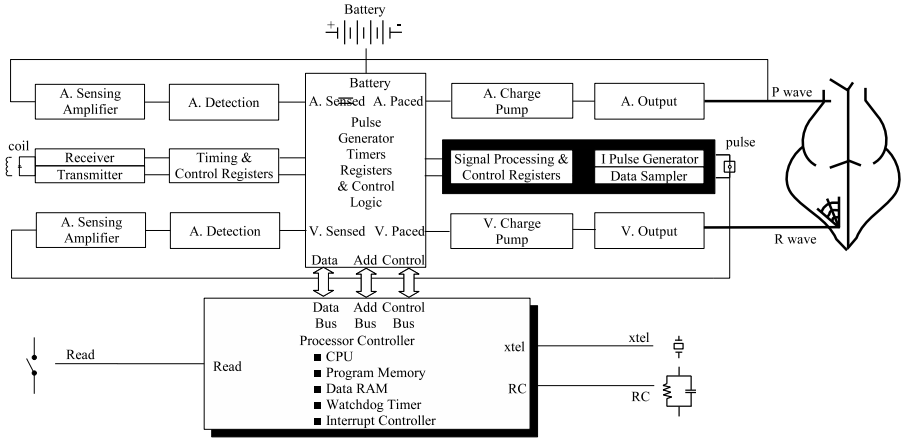


Figure 2.18. Block diagram of a Rate-Adaptive Multiprogrammable Microprocessor Controlled Dual-Chamber Pacemaker (DDD-R) Based on pre-ejection period (PEP) [20].

Finally in 2002, M.P.R. Hexamer, M. Meine, A. Kloppe, E. Werner, proposed a control sensor for rate responsive pacemakers [21]. This sensor consists of the atrio-ventricular conductive time (AVCT). The AVCT corresponds to a well defined interval in the intra electrocardiogram. Moreover, the AVCT is coupled to the sympathetic and parasympathetic activity of the autonomous nervous system (dromotropic effect),

leading to a shortening of the AVCT during exercise.

A pacemaker contains its hardware inside a sealed case with its software stored as an array of binary instructions. However it cannot accomplish its objective of pacing the heart if it does not have a suitable power supply providing the energy necessary to generate the stimuli. The next section discusses the most important achievements in pacemaker's power supply.

2.4.3 Pacemaker's power supply (Battery)

Irrespectively of the type of pacemaker, that is, external or implanted, the pulse generator requires a source of electric energy. With implanted pacemakers, some internal source of electric energy is required. Chemical batteries are by far most popular; however, nuclear powered cells have been used occasionally.

A chemical cell is the functional unit that produces electrical energy. A battery is merely a group of cells arranged in series or parallel. Each cell contains two electrodes (anode and cathode) and an electrolyte. Electrons are produced by a chemical reaction within the cell. The chemical reaction alters the composition of the electrodes and the electrolyte. Often gas is evolved as the cell is used. With the passage of time, in some cells, the open circuit voltage is relatively constant and the internal resistance rises. The term shelf life is often used to identify the self-discharging propensity of chemical cells. The concept of a long shelf life for a cell is of obvious importance in pacemaker technology, since the cells are sealed in the pacemaker at the time of manufacture, and implantation may not occur for a considerable time. Another important characteristic of the cells used in pacemakers relates to the by-products of the chemical reactions that produce the current. Cells that liberate substantial amounts of gas are not used for pacemaker construction.

There are two main chemical cells batteries for pacemakers, the Mercury-Zinc battery and the Lithium-Iodide battery.

Mercury-Zinc batteries

Mercury-Zinc batteries contain a porous zinc cathode and an anode composed of a compressed mixture of mercuric oxide, graphite, and silver oxide. The electrolyte is largely potassium hydroxide, and the chemical reaction that takes place to yield electrons is:

$$Zn + HgO + H_2O \rightarrow Zn(OH)_2 + Hg$$

The open-circuit voltage is 1.35V and typical cells provide 1 ampere-hour of charge when discharged at 40 A. The power density is on the order of 500 mW-hr/cm3 of cell. Although this battery was quite common in the first implanted pacemakers, they are not currently used at all for implantable pacemakers.

Lithium-Iodine batteries

In Lithium-Iodine cells the anode is lithium and the cathode is a proprietary iodide (MI2). Instead of a liquid electrolyte, a pasty salt of lithium is used. The chemical reaction, which releases electrons, was given by Greatbatch [11] as

$$2Li + MI_2 \rightarrow M + 2LiI$$

In this reaction no gas is liberated. The open circuit voltage of the cell is 2.8V, and has ratings of 2 ampere-hours. The power density is similar to or slightly higher than that of the mercury-zinc cell. The lithium-iodine has several unique features that make it an ideal candidate for pacemakers. For example, since no gas is liberated, the cell can be completely sealed. This feature allows the whole pacemaker to be enclosed in a welded metal container, thereby rendering it fluid tight and reducing

susceptibility to electromagnetic interference.

Lithium-Iodine batteries are the standard in modern pacemakers. Figure 2.19 depicts a diagram of a typical lithium-iodine battery. Notice that the case is the anode (+) and the output pin is the cathode (-).

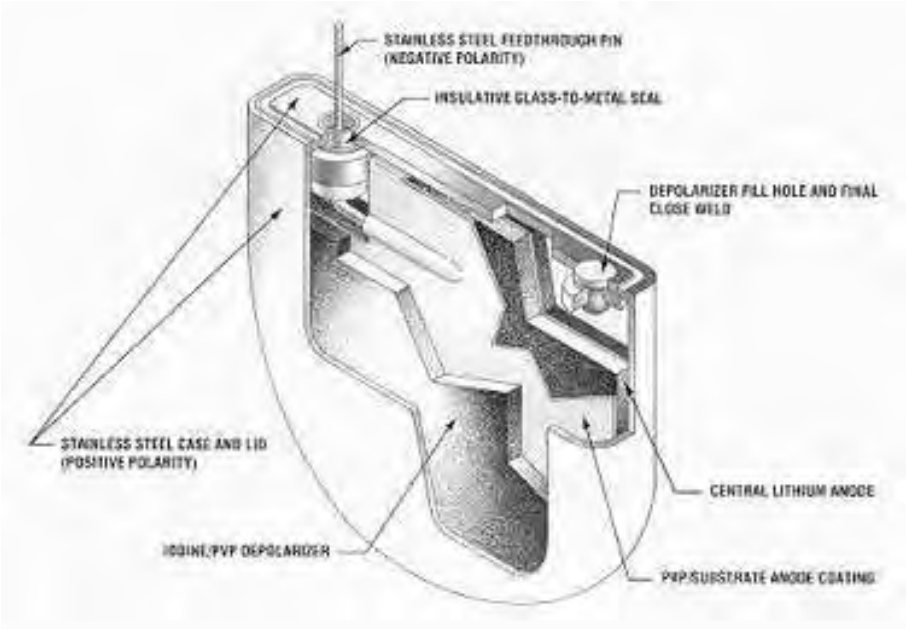


Figure 2.19. Anatomy of a Lithium-Iodine Battery [22].

These are the typical characteristics for a standard lithium-iodine cell [23]:

- Open Circuit Voltage : 2.8 Volt
- Control Circuit minimal voltage : 2.2 Volt
- Control Circuit current drain : 10 μA
- EOL battery resistance : 10 k Ohms
- Chold : 10 μF
- Discharge times : 1 ms, 5 ms
- Oscillator frequency : 167 Hz
- Duty Cycle : 16.7 %
- Ah rating : **2 Ah (typical rating)**
- Reliability : 99.6% probability of survival beyond 8 years
- Failure Rate : 0.005% failures/month.

Modern batteries are smaller and more efficient. A typical battery weights around 12.5g, although cutting edge technologies are reducing the size significantly. Figure 2.20 shows a picture of the internal circuitry of a modern VVI pacemaker.

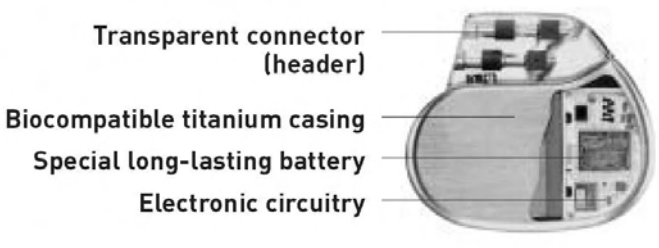


Figure 2.20. Layout of a modern pacemaker [24].

The major companies in the United States that produce pacemakers are Medtronic

Inc, Guidant Corporation and St. Jude Medical Inc (SJM). Some prestigious international pacemaker industries are Biotronik GmbH & Co. and CCC medical devices from Germany and Uruguay, respectively. Table 2.2 compares commercially available VVI pacemakers in terms of longevity. The current consumption for each pacemaker is calculated using equation 2.2. A typical Ah rating for a lithium-iodine battery (2Ah) is assumed. As can be seem in Table 2.2 the current consumed by the stimulus itself is a very small portion, about 10% of the total current consumption. This implies that almost all the current, about 90%, is consumed to generate the stimulus. This 90% of the current is shown in Table 2.2 as I_{System} and is composed of the pacemaker's software and hardware consumption.

$$I_{consumed} = \frac{Ah\ rating}{Hours\ of\ Service\ Life} \quad (A) \tag{2.2}$$

Table 2.2. Longevity Comparison Table.

Company	Product	Characteristics	$I_{Stimulus}$	I_{System}	Service Life	Longevity Testbench
Medtronic	Sigma VVI 100	Single Chamber Rate Program I_{Total} = 22.8 μA	1.7 μA	21.1 μA	10 yrs	2.5V, 0.4ms, 60bpm, 100% paced at 600Ω
Guidant	Insignia 1198	Single Chamber Onset EGMs I_{Total} = 22.4 μA	2 μA	20.4 μA	10.2 yrs	2.5V, 0.4ms, 60bpm, 100% paced at 500Ω
SJM	Regency 2402L	Single Chamber AutoCapture I_{Total} = 11.1 μA	1.1 μA	10 μA	20.5 yrs	1.5 V, 0.3 ms, 60bpm, 100% paced at 400Ω

2.5 VVI cardiac pacemakers

In 1965, on-demand pacemakers were developed. These types of pacemakers used the pacing electrodes to apply the stimulus, but also to detect natural occurrences

of ventricular beats. There are two types of on-demand pacemakers: (1) demand-inhibited and (2) demand-triggered pacemakers. Demand-triggered pacemakers consume larger quantities of energy, although they are useful to the physician for diagnostic purposes. On the other hand, the demand-inhibited consume less battery, which aligns with the objectives of this work. Furthermore, it is known that right ventricular inhibited pacing is the most used therapy for maintaining a suitable heart rate in the presence of chronic or paroxysmal bradycardia [9] [25].

The ventricular inhibited pacemaker senses the R wave of the ventricles and produces no stimulus until a preset time (e.g., 800-1000ms) has elapsed. At the end of this time a pace pulse is delivered; then the pacemaker waits for another R wave. Figure 2.21 illustrates schematically the manner in which this type of pacemaker operates.

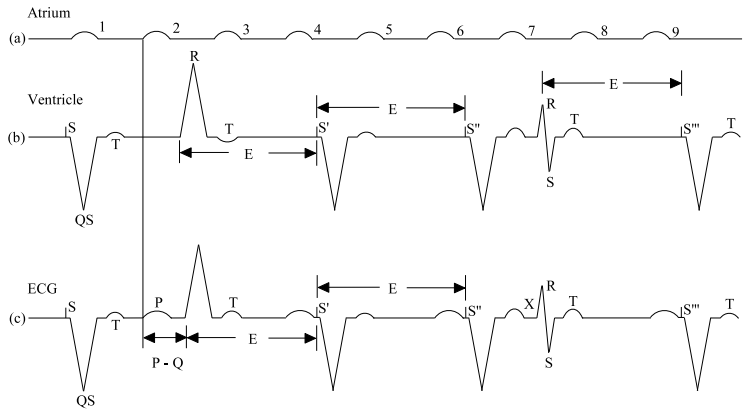


Figure 2.21. Diagrammatic representation of the operation of the R-wave-inhibited pacemaker. (a) illustrates atrial, (b) illustrates ventricular activity, and (c) identifies the ECG [11].

The first beat in Figure 2.21 was initiated by the pacemaker stimulus (S). Following this beat, A-V conduction (P-Q) was restored and a normal beat (R) occurred, inhibiting the pacemaker. However, after an A-V block occured, the pacemaker waited the elapsed time E and then delivered a pacemaker stimulus S', followed by a second

stimulus S". After the fourth ventricular beat, a ventricular ectopic beat X (R-S in Figure 2.21) takes place, and the pacemaker is inhibited again. The pacemaker waited for its preset elapsed time (E) and then delivered a stimulus (S"") to evoke a beat.

Although VVI pacing is the most widely-used, recent articles [26] [27] have shown that Dual-Chamber pacing improves exercise capacity and quality of life. However, these papers are based on theoretical data and lack of hard evidence from large-scale randomized trials. Other factors that may encourage the usage of VVI pacemakers is that Dual-Chamber pacing is more expensive and that pacemaker implanters have a limited experience with atrial-based pacing and atrial leads.

The pacemaker leads are one of the most important parts of any pacemaker. They provide the interface between the pacemaker and the heart. These can be implemented using a bipolar or a unipolar configuration. The difference between these two is that bipolar electrodes carry the signal differentially while unipolar carry the signal with reference to the can of the device. The design of the pacing electrode also affects the capacity of the battery [28]. In order to stimulate the heart, a sufficient amount of energy must be concentrated near excitable tissue. For this reason, an important consideration is the density of current at the tip of the electrode. This current density is affected by several factors, including the surface area of the electrode, amount of fibrotic encapsulation, electrical material, pulse width and pulse amplitude among others. Manufacturers have focused on these factors and have achieved a dramatic reduction in energy requirements for maintaining capture, which results in consistent depolarization of the heart.

The objective of this research is to reduce the power consumption of a cardiac pacemaker. With this purpose in mind, the VVI pacemaker is chosen as the design in which the research is conducted. VVI offers advantages in terms of power reduction

(e.g. use one sensing channel and one pulse generator) and less complicate algorithms. The research is focused in the pacemaker as a whole, from the control algorithm design to the hardware implementation. For the actual implementation a general purpose microcontroller is used. This approach permits to evaluate the performance of the control algorithm in an actual low cost platform. The next chapter presents a review of microcontrollers making emphasis on their relevant characteristics for cardiac pacing.

CHAPTER 3

Microcontrollers and Embedded Systems

Microcontrollers (μCs) are everywhere. From cars to washing machines, there is an unimaginable number of day to day equipments with a μC inside. In simple terms, μC is a computer-on-a-chip, so it is sometimes also known as microcomputers. Typically, a μC contains all the memory, peripherals and Input/Output interfaces necessary for embedded applications. Unlike general-purpose microprocessors, the kind used in personal computers, a μC emphasizes self-sufficiency and cost-effectiveness. Around 3 billion embedded μCs are sold every year, with the smaller CPUs (4, 8, and 16 bit) dominating the market [29].

Pacemakers were one of the first embedded systems in the biomedical field to include a customized μC as a central processing unit. However, customized microcontrollers are very expensive while general purpose μCs are available at moderate prices. This work considers the option of using a general purpose μC as the central processing unit of a pacemaker. In order to do so, this chapter reviews the different CPU architectures available today in general-purpose μC and discusses the implications of those architectures in a pacemaker system. Moreover, the definition of an embedded system

with a detailed explanation of the pacemaker as an embedded system is presented.

3.1 Description of general-purpose microcontrollers

A general purpose microcontroller is an application-specific instruction set processor (ASIP). There are three categories of ASIPs: 1) Processors, 2) Microcontrollers, and 3) Digital Signal Processors (DSPs). Table 3.1 shows most popular ASIPs. Processors and DSPs normally have great processing power, but are power-hungry devices. Since microcontrollers lower power consumption, which is a very appealing feature for embedded systems. μC's are are used in quite different applications, they are customized for such different environments by software programming using a specific instructions set [30]. A typical μC will have a built in clock generator and a small amount of RAM and ROM, which may be an EPROM or a EEPROM, although more recently Flash memory has been also used. μCs also have a variety of peripheral devices, such as analog-to-digital converters, timers, UARTs or specialized serial communications interfaces like I^2C, Serial Peripheral Interface and Controller Area Network.

In the beginning, μCs were only programmed in assembly language and later in C code. The debugging options were limited, but recently μCs are designed with a Joint Test Action Group (JTAG) interface, an IEEE standard, included. The IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture was created to regulate the test access ports used for testing printed circuit boards with boundary scan. This standard enables a programmer to debug the software of an embedded system within the actual μC. JTAG brings enormous advantages to the programmer since the program run in the actual hardware with the actual constraints

Table 3.1. Most popular Application-Specific Instruction Set Processors.

Processor	Clock Speed	Periferals	Bus Width	MIPS	Power	Tran-sistors	Price
General Purpose Processors							
Intel Pentium 4	3GHz	2x16KB L1, 2M L2, HTT FSB 800MHz	32	11356	86 W	55M	$475
AMD64 3800+	2.4GHz	128KB L1, 512KB L2, FSB 2000MHz	64	11176	89W	68.5M	$329
IBM PowerPC 750X	550MHz	2 x 32KB L1, 256KB L2	32/64	1300	5W	7M	$100
Microcontrollers							
Intel 8051	12MHz	4KB ROM, 128KB RAM,32 I/O Timer, UART	8	.67	.2W	10k	$3
Motorola 68HC811	3MHz	4KB ROM, 192KB, RAM,32 I/O Timer, WDT	8	.57	.135W	10k	$10
Digital Signal Processors							
TI C2812	150MHz	40KB RAM, 256KB ROM, CAN, SPI, Timer, UART	32	150	.63W	NA	$34
AD ADSP21992	150MHz	WDT,3 Timers DMA controller POR generator	32	160	NA	NA	$50

of the system. μCs trade away speed and flexibility to gain ease of equipment design and low cost. There is a limited die area to include functionality. This limitation cause the wide variety of μCs since there is a diversified way of deciding which peripherals should be included.

3.2 Microcontrollers' CPU Architectures

The Central Processing Unit (CPU) of any computer is vital to the overall system performance. This is also true to μCs since they need to be very reliable, fast and

with a wide functionality. A general CPU consists of a datapath and a control unit linked together by a memory.

Figure 3.1 illustrates the arrangement of a basic CPU.

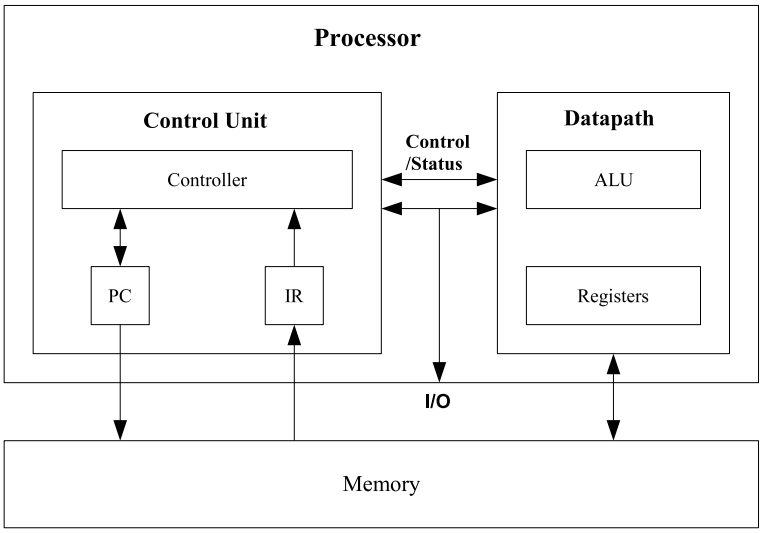


Figure 3.1. Basic block diagram of a typical microcontroller.

3.2.1 Datapaths

The datapath consists of the circuitry for transforming data and for storing temporary data. It contains an arithmetic and logic unit (ALU) which manipulates data, monitors the status of the data and generates signals that are stored in a status register. Typically, processors are described by the size of datapath components; thus, in an 8-bit processor, data is transported in packets of 8-bits. Common sizes are 4-bit, 8-bit, 16-bit, and 32-bit. More recently, 64-bit processors have gained considerable popularity. However, sometimes processors are designed containing multiple buses sizes. An example of multiple bus sizes is IBM's PowerPC (see Table 3.1).

3.2.2 Control Unit

The control unit consists of circuitry for retrieving program instructions and for moving data through the datapath according to those instructions. The control unit has a program counter (PC) that holds the address of the next program instruction, and an instruction register to hold the fetched instruction. The controller inside the control unit can be described as a finite state machine that sequences through the different states, generating the signals necessary to control the datapath. The controller also determines the value of the next instruction. For a branch instruction, the controller looks at the datapath to determine the appropriate address. For every other instruction, it just increments the value of the PC.

The Program Counter's size determines the processor's address size, that is, the number of directly accessible memory locations. This addressable memory is known as address memory. Each instruction may involve several fetches to the address memory, since the instruction but also the operands for that instruction. All this process occurs in a unit of time known as clock cycle. A clock cycle is usually the longest time required for data to travel from one register to another. The inverse of the clock cycle is the clock frequency which is used to compare speeds of different processors.

3.2.3 Memory

Registers serve as processor's short-term storage. However, since the processor execute many instructions in a typical program it is necessary to store results and the instructions themselves in a different space. Memories are the processor's medium and long term storage devices. The information stored in a memory is classified in two categories, data or program. Data information represents the values being input, output and manipulated by the program. Program information consists of

the sequence of instructions that cause the processor to carry out the desired system functionality. The way data and program information is stored in memory depends of the memory architecture. There are two basic configurations: 1) Harvard architecture and 2) Von Neumann or Princeton architecture.

Harvard architecture

The Harvard architecture uses physically separated storage and signal pathways for instructions and data. This architecture is depicted in Figure 3.2. In a computer with Harvard architecture, the processor can read both instructions and data from memory at the same time. In other words execution occurs in parallel. Harvard architecture can be faster because it is able to fetch the next instruction at the same time it completes the current instruction. However this faster execution is obtained at the cost of silicon complexity. This complexity traduces in larger silicon areas and complicated pathway networks that may increase the power consumption of the processing unit.

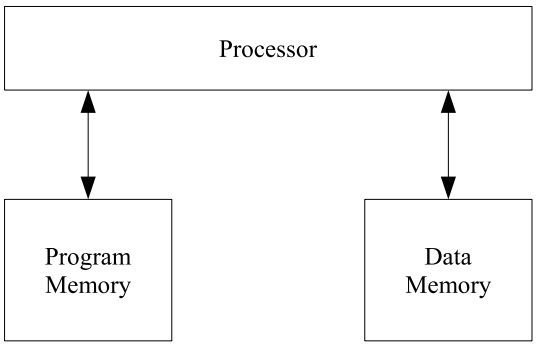


Figure 3.2. Basic block diagram of the Harvard Architecture.

Von Neumann architecture

The Von Neumann architecture, also known as a "stored-program computer architecture", refers to a memory design that uses a single storage structure to hold both instructions and data. Figure 3.3 shows the block diagram of Von Neumann architecture. By treating instructions in the same way as data, a stored-program machine can easily change the program, and can do so under program control. Von-Neumann architecture has a single "data" bus to fetch both instructions and data. When the controller addresses main memory, it first fetches an instruction, and then it fetches the data to support the instruction. Since it has a single bus to access memory, the instruction-fetching process takes longer than in Harvard architecture. On the other hand, Von Neumann architectures are typically simpler than Harvard architectures, resulting in less silicon area. Since the signals to be process by a pacemaker are very slow (500ms - 1s) the difference in fetching time between the two architectures is insignificant.

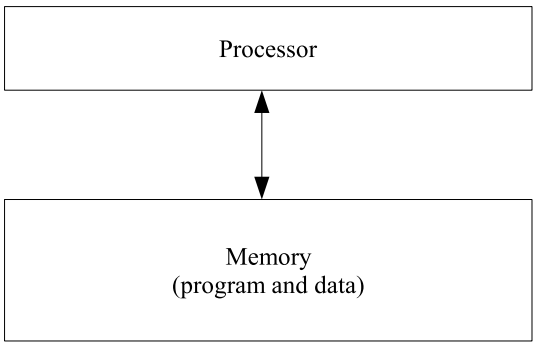


Figure 3.3. Basic block diagram of the Von-Neumann Architecture.

Memory may be read-only memory (ROM) or readable random access memory (RAM). ROM has the advantage of occupying less space than RAM. Therefore, many μCs have larger amount of ROM than RAM. In embedded systems ROM is used as a program memory, since program does not change quite often in an embedded system. Constant

data is stored in ROM too, and RAM is mostly used for real time data.

To increase the speed of fetching instructions and data, a local copy of a portion of memory, known as cache is put inside the CPU. This memory is optimized to be accessed at faster rates. Cache memory is usually implemented using static RAM since, although expensive, is faster than dynamic RAM. Figure 3.4 shows the difference between cache and normal memory. Cache memory uses the principle that if at a particular time a processor access a particular memory location, is highly probable that in a near feature it will also access the surrounding memory locations. For this reason, when the processor first access a memory location it copy the whole memory block in cache to have the information more accessible. This information might be either data or program information.

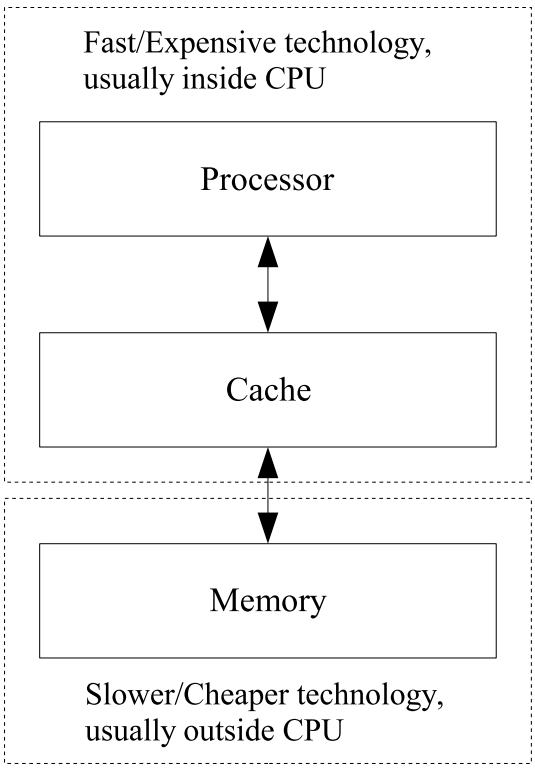


Figure 3.4. Cache Memory.

3.3 Microcontrollers' Instruction Set Architectures

The way the instruction set is designed in μCs greatly influences system performance. An instruction set architecture describes the aspects of computer architecture visible to a programmer. It includes the native datatypes, instructions, registers, addressing modes, memory architecture, interrupt and exception handlers, and external Input/Output of μCs. Since instruction set architectures are independent of the actual hardware of the μC they have become a very attractive design option. For example, the Intel Pentium and the AMD Athlon implement nearly identical versions of the x86 instruction set, but have radically different internal designs. The following sections will discuss the most widely used instruction architectures in terms of advantages and disadvantages, and also will compare side by side their performance.

3.3.1 CISC architecture

In the complex instruction set computer (CISC) architecture, which each instruction can execute several low-level operations, such as a load from memory, an arithmetic operation, and a memory store, all in a single instruction. The term was coined in contrast to another emerging architecture at the time, the reduced instruction set computer (RISC).

A typical CISC μC has well over 80 instructions, many of them very powerful and very specialized for specific control tasks. It is quite common for the instructions to behave quite differently. Some might only operate on certain address spaces or registers, and others might only recognize certain addressing modes. The disadvantage of CISC architecture is that although the high-level program can be expressed in fewer instructions, the actual performance of the μC is not always improved. Actually, the complexity of the instruction set is directly proportional to the overhead of decoding

any given instruction. This means that adding a large and complex instruction set to a μC, the simpler instructions slow down too. Some devices with CISC are the Motorola 68000 family and the Intel 80x86 CPUs. Modern CISC architectures divide the complex instructions in subsets of smaller instructions improving performance by reducing the overhead [31]. This is precisely the philosophy behind the RISC architecture.

3.3.2 RISC architecture

The reduced instruction set computer, or RISC, favors a smaller and simpler set of instructions that take about the same amount of time to execute. The key elements shared by most RISC architectures are [32].

- A limited and simple instruction set.
- The use of either a hardware or compiler strategy to maximize the use of registers and minimize references to main memory.
- An emphasis on optimizing the instruction execution pipeline.

The motivation behind RISC architecture philosophy is based on studies that demonstrate that 25% of the instructions in a CISC architecture make 95% of the executions. This implies that 75% of the hardware that support these instructions is not used at all, but consumes silicon area [31]. When using a High Level Language (HLL) such as Fortran, Basic, C or others, RISC architectures outpast their CISC contenders as illustrated in Figure 3.5.

HLL's are very popular since the programmer can practically forget about the hardware behind the software. Yet, when dealing with μCs the programmer needs to have fair understanding of the hardware because of the peripherals that are needed

Benchmark programs	RISC I (milliseconds)	MC68000	Z8002	VAX 11/780	PDP 11/70	BBN C/70
		(number of times slower than RISC I)				
S	0.46	2.8	1.6	1.3	0.9	2.2
B	0.06	4.8	7.2	4.8	6.2	9.2
L	0.10	1.6	2.4	1.2	1.9	2.5
M	0.43	4.0	5.2	3.0	4.0	9.3
Q	50.40	4.1	5.2	3.0	3.6	5.8
A	3,200	—	2.8	1.6	1.6	—
R	800	—	5.9	2.3	3.2	1.3
T	6,800	—	4.2	1.8	2.3	1.6

A = Ackermann's function
B = Testing, setting, and resetting 3 bits within a bit string
L = Inserting five new entries into a doubly linked list
M = Transposing a square-bit matrix
Q = Performing a nonrecursive quicksort algorithm on a large vector of fixed-length records
R = Performing a recursive quicksort algorithm on 2,600 fixed-length character strings
S = Examining a character string for the first occurrence of a substring
T = Towers of Hanoi

Figure 3.5. Execution times for C-benchmarks on the RISC I and CISCs [33].

to achieve control of an embedded system. Moreover, for applications where speed or power consumption are tight constrains, HLL may be impractical. More on this subject will be discussed in the section about embedded systems.

Significant effort has been invested in identifying common operations and optimizing them in terms of speed. Since the maximum clock rate of a CPU is limited by the speed of the slowest instruction, speeding up that instruction results in higher clock rates. Instructions can be speed up by reducing the number of addressing modes they support. At the end, it is desirable that each instruction in a RISC architecture can run in only one clock cycle.

Although at a first glance RISC architecture looks almost perfect, it has several drawbacks. Ironically, RISC shortcomings are a byproduct of its advantages. Since to execute a single operation RISC will need more instructions than CISC, a program developed on a RISC platform will occupy more memory [31]. The problem with

larger programs is not a limitation in terms on memory space, but the traffic between memory and CPU. Any operation performed inside the CPU will be executed at least 100 times faster than when fetching memory. This behavior is shown in Figure 3.6. One idea that RISC encouraged to solve this problem was pipelining in the mid '80s.

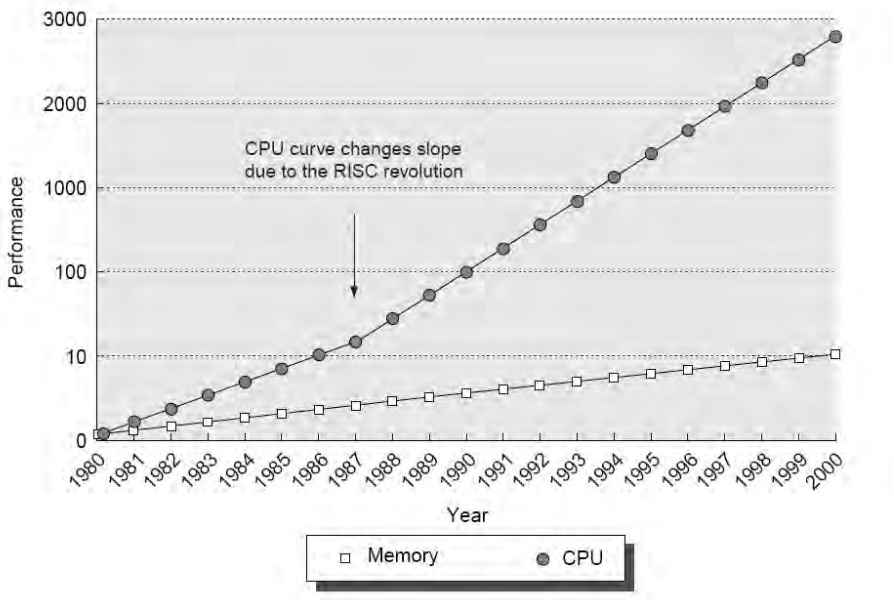


Figure 3.6. Performance time development for CPU and Memory [34].

RISC architecture is highly recommended over CISC architecture for low power applications. Moreover, as mentioned before, the RIC configurations optimize a small set of instructions to be executed in one clock cycle, thereby allowing the inclusion of pipelining and minimizing wake-up time.

3.3.3 Pipeline architecture

Pipelining greatly improves throughput, is the average number of instructions performed in one second. Instructions consist of a number of steps, which constitute the so called instruction cycle. These steps can be resumed in three tasks:

1. *Load or Fetch*: Reading the instruction from memory

2. *Execute*: Interpreting and executing the instruction

3. *Store*: Storing result in memory or register

To do this sequentially is a wasteful approach, since each task uses only part of the hardware; this will remain idle while the other tasks are performed. For example, while the processor is using the adder, the loader of data is idle. Pipelining improves the performance of the CPU by reducing the idle time of each piece of hardware. Pipelined CPUs subdivide various functional units within a processor into different stages, or relatively independent components, which can each be working simultaneously on a different task. Stages are ordered in sequence with the output of each stage feeding the input of the stage after it. This way, the overall clock speed can be increased tremendously. Let us illustrate with a 3-stage pipeline work

Consider the following pseudo-assembly code running on a 3-stage pipeline:

LOAD # 10,A ; load 10 in A
MOVE A,B ; copy A in B
ADD #15,B ; add 15 to B
STORE B, 0x100 ; store B into memory cell 0x100

Assuming one clock for each task in the instruction cycle, the four instructions would require 12 clocks. On the other hand, with pipelining the complete set is execut.ed in six clocks, as illustrated in Figure 3.7. Here, in the first clock, LOAD instruction is fetched; in the second clock, while LOAD is executed, the MOVE instruction is fetched, etc.

	Load step	Execute step	Store step
clock1:	LOAD		
clock2:	MOVE	LOAD	
clock3:	ADD	MOVE	LOAD
clock4:	STORE	ADD	MOVE
clock5:		STORE	ADD
clock6:			STORE

Figure 3.7. Sequence of instructions with pipelining.

Although pipeline architectures have promising advantages, they also suffer from drawbacks. The larger problem when using pipelining is that every time that a branch is taken in software the pipeline must be flushed. Branch predicting is used to alleviate this effect of branching in pipelines architectures. It also helps when the programmer reduce the number of branch instructions.

3.4 Definition of an embedded system

An embedded system is a special-purpose computer system completely encapsulated by the device it controls, and does not require any human interaction to work. It is a computer-controlled system, with pre-defined tasks and specific requirements. For this reason an embedded system usually has a μC as its core controller. Since embedded systems are designed for mass production, they need to have reliable software and hardware parts and, in most cases, should accomplish this with the lower possible cost.

The first recognizably modern embedded system was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory. Since this first implementation the application field has growth exponentially. Today em-

bedded μCs systems can be found in Automatic Teller Machines, cellular telephones, computer printers, network routers, handheld calculators and medical devices, to name just a few.

3.4.1 The pacemaker as an embedded system

The cardiac pacemaker can be considered an embedded system since it monitors the heart and delivers a pulse when needed without human intervention. All the circuitry of the pacemaker is enclosed inside a hermetically sealed encapsulation. This encapsulation has two purposes: 1) To help filter out electromagnetic disturbance, and 2) prevent body fluids to get inside the electronic circuitry, causing premature oxidation of electronic components. The only interface a pacemaker has is via a magnetic programmer that is connected to a personal computer (PC). All these characteristics define a cardiac pacemaker as an embedded system.

The most basic pacemaker configuration is an oscillator. However, this one will operate as an asynchronous timer, with the problem that if the heart generates its natural response there will be a competitive rhythm. If the stimulus generated by the pacemaker coincides with the vulnerable period of the heart it will cause ventricular fibrillation. Figure 3.8 shows that the localization of the vulnerable period is when the T-wave is getting to its summit. This behavior should be expected since the T-wave represents the moment when the ventricles are in the repolarization state. Ventricular fibrillation should be avoided since it is too dangerous to the quality of life of a patient. Fibrillation in the ventricles results in a loss of pumping and the fall in blood pressure to a near-zero level. Figure 3.9 presents the behavior of the heart during a ventricular fibrillation.

As the picture suggests, since the ventricles are contracting at such a rapid rate the

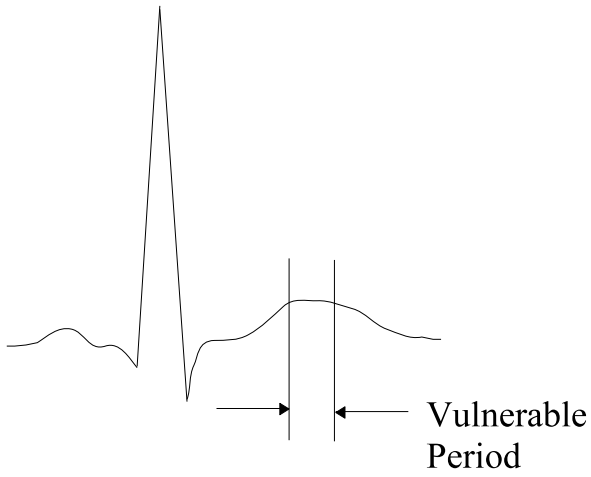


Figure 3.8. Vulnerable period of the heart.

Figure 3.9. Typical ECG of a Ventricular Fibrillation.

ventricle chambers are not able to accumulate enough blood to pump to the body, causing a zero cardiac output. Therefore the tissues start to die because they do not receive the oxygenated blood coming from the lungs. Although asynchronous pacemakers are currently non existent, modern pacemakers include a programmability option to put the pacemaker in asynchronous mode. This is because, although dangerous, a cardiologist may use it to analyze the behavior of the heart and check the status of the pacemaker. Almost all currently used pacemakers are controlled via customized μCs using system-on-chip configurations.

There is not doubt that by using a μC, the reliability of a pacemaker can be improved.

Furthermore, programmable features can be included which make the system more flexible. The μC architecture should include some of the features presented above. The Von Neumann architecture satisfies the memory mapping requirements of a pacemaker system, since it involves less silicon complexity which probably consumes less power. Moreover, RISC architecture is also preferred because of power requirements and speed.

Another important part of the architecture selection is the size of the data that will be processed. Since a pacemaker system needs to count for large periods of time (up to 1s, which can take many bits to represent depending on the counter clock) it is more functional to have 16-bit buses. 32-bit are not considered because CPUs with this structure consumes more power. By using 16-bit buses, larger quantities of data can be accessed and processed in a single clock cycle reducing memory paging and code size. These reductions will in turn maximize the sleep mode time of the μC getting a lower power consumption.

As mentioned in Chapter 2, the total current consumption of pacemakers should be around 20 μA. To satisfy this constraint HLLs should be avoided in the development of the control software for the pacemaker. The problem when using HLLs for low power applications is that programmers do not control how the program is encoded into the machine language. The compiler constructs the machine code. Although today compilers are sophisticated and have very good optimizations schemes, they will nevertheless encode unnecessary instructions into machine codes. These unnecessary codes will take execution time thereby wasting power, a luxury that cannot be taken in pacemakers. Therefore, the preferred programming language for the control algorithm is ASSEMBLY since it allows the programmer to have greater hardware control and a one to one instruction relationship with machine code. Chapter 5 dis-

cusses the trends in the development of low power algorithms as well as the research done in recent years in the minimization of power consumption in control algorithms for embedded system.

To summarize, based on the previous observations, the microcontrollers to consider in our selection process should have these 4 architectural features:

- Von Neumann CPU architecture.
- RISC Instruction set architecture.
- 3-stage pipeline system (3-stage as a minimum size)
- 16-bit data processing

In the next chapter a methodology for the development of this research will be presented. One part of that methodology is the microcontroller selection process. For this effect, all the inputs given in this chapter will be considered.

CHAPTER 4

Design Methodology and Objectives

4.1 Pacemaker Design Methodology

The intention of this research is to develop a prototype of a VVI cardiac pacemaker. For that purpose, a methodology for the design of a pacemaker is proposed as follows:

1. Device requirement definition
2. Microcontroller selection
3. Software / Hardware specifications
4. Software Design
5. Hardware Design
6. Integration
7. Verification (Testing)

This chapter discusses the first two steps.. Chapter 5 will explain in detail the software specifications and software design. Chapter 6 will include the hardware specifications, hardware design, integration and verification of the whole system. Finally, Chapter 7 will present the results, conclusions and future work.

4.2 Device requirements definition

The device requirements describe what the device must do in order to be considered a pacemaker. The specifications used in this research were developed by taking the reference of a commercially available VVI pacemaker, the TEROS SSI 503. This is a product offered by the "Centro de Construcción de Cardioestimuladores del Uruguay S.A." [35]. It will serve as a benchmark for the prototype developed in this research. From now on, to avoid confusion, the pacemaker developed in this research will be called PROV910.

4.2.1 System Description

The PROV910 will be a VVI cardiac pacemaker used to treat bradycardia, with the following charactristics:

- It should stimulate and sense the heart with unipolar or bipolar polarities.
- It operates using a lithium battery of 3V, and works for supply voltages from 2V to 3V.
- It should offer a wide range of programmability, including basic pace rates, pulse widths, pulse amplitudes, hysteresis intervals, sensitivities and polarities.
- Although described as VVI, it also includes the following modes: VVT, VOO, AAI, AAT, and AOO.

- The whole system is intended to have low power consumption and a life of 10 years or more when using a 2Ah rate

This program will be develop using IAR free development environment and debugger for the msp430. A prototype will be implemented using surface mount (smt) discrete components, and it should be a low cost system.

4.2.2 Definition of Pacemaker's Parameter

The pacemaker has several programmable parameters that must be defined. These are divided in two categories: 1) Timing parameters, and 2) Operational parameters. The timing parameters are those that define the timing behavior of the pacemaker; the operational parameters include all others that have nothing to do with time intervals, but are necessary for proper operation of the pacemaker. The four timing parameters are: basic pace interval, escape interval, ventricular refractory period (VRP) and pulse width (PW). The operational parameters are: pulse amplitude (A), sensibility, sensing and pacing polarities, and mode of operation. Figure 4.1 shows the timing parameters and the pulse amplitude.

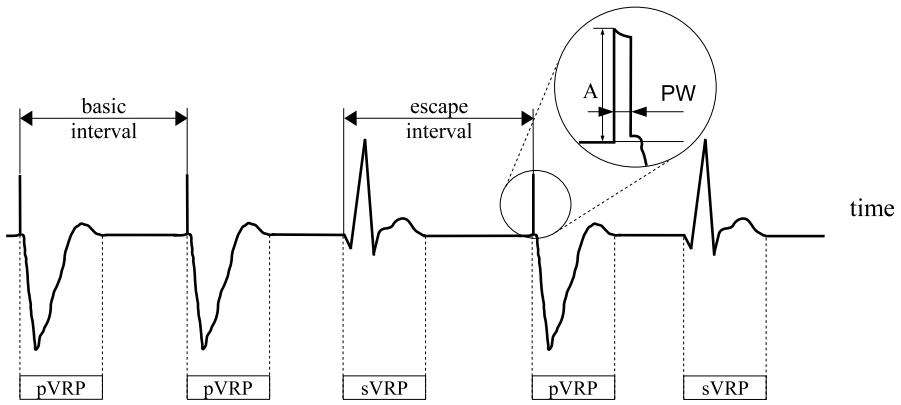


Figure 4.1. Pacemaker's programmable paramaters.

Pacemaker's timing parameters

- *Basic Pace Interval*

  The basic pace interval is the period of time that the pacemaker awaits to apply a stimulus to the heart. It is measured in beats per minutes (bpm) or min^{-1}.

- *Escape Interval*

  The escape interval is the period of time that the pacemaker awaits after a spontaneous QRS complex has been generated. If the hysteresis is off, this interval is the same as the basic pace interval. However, if hysteresis is turn on this interval becomes longer that the basic pace interval by the amount specified by the hysteresis parameter. Hysteresis is used to reduce the number of stimuli generated by the pacemaker and hence to reduce power consumption. The reduction is achieved because if a spontaneous QRS occurs, it is highly probable that more of them are generated by the heart. The escape interval has the same units of the basic pace, which are bpm.

- *Ventricular Refractory Period (VRP)*

  The VRP is the amount of time that the sense circuit is turned off. This is done to avoid sensing the pacemaker own stimulus, the paced QRS complex, the T wave and afterpotentials. If the sensing circuit is not turn off then it will generate stimuli to all these events causing ventricular fibrillation. Figure 4.1 shows two kinds of VRPs: sensing VRP and pacing VRP. These two intervals usually are programmed to the same amount of time which is measure in ms.

- *Pulse Width*

  The Pulse Width is the amount of time the pulse generator will supply the stimulus to the heart. This parameter is of immense importance to capture the heart. Capture is the action of generating a potential that develop a chain reaction through out the ventricle. The pulse width is measure in ms.

Pacemaker's Operational parameters

- *Pulse Amplitude*

  The pulse amplitude and the pulse width are essential for correct capture of the heart. The amplitude and witdh are related to the energy consumption curve shown in Figure 4.2, for which it is seen that there exists a minimum. The dashed line (b) in the figure is the rheobase or minimum-current asymptote. The unit for the pulse amplitude is Volt (V).

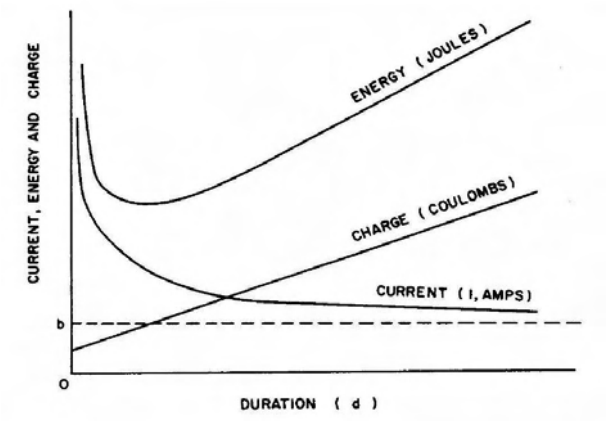


Figure 4.2. Strength-Duration duration curve for cardiac stimulation.

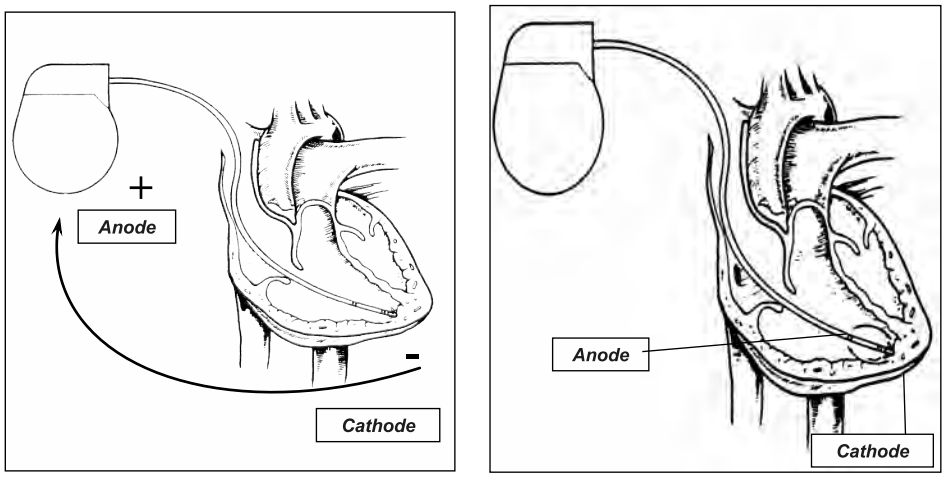
- *Sensitivity*

  The sensitivity refers to the minimum threshold voltage of an input signal that must generate a flag in the microcontroller. This flag resets the basic timer and marks the beginning of the escape interval. This parameter is measured in mV.

- *Pacing and Sensing polarities*

  There are two polarities available for either pacing or sensing the heart, unipolar and bipolar polarities. Unipolar polarity uses an electrode with the cathode (-)

located in its tip and the anode (+) located on the encapsulated plate that surrounds the pacemaker. The unipolar concept is shown in Figure 4.3(a). On the other hand, bipolar polarity uses an electrode that consists of a ring and a tip, both located in the electrode. In this case the ring is the anode (+) and the tip is the cathode (-). More on the subject of polarities will be discussed in Chapter 6 where the pulse generator along with the voltage multiplier will be explained. Figure 4.3(b) illustrates the bipolar polarity.



(a) Unipolar polarization (b) Bipolar polarization

Figure 4.3. Sensing and pacing polarization modes.

- *Modes of Operation*

  The modes of operation specify which part of the heart is paced and which part is sensed. This parameter also involves the type of response to sensing. The different types of modes are: VVI, VVT, VOO, AAI, AAT, and AOO (see Table 2.1).

As a matter of fact, the specified sensitivity for the PROV910 will have a wide range so that either a p-wave or a QRS complex can be sensed. This means that the mode

will be finally decided by the placement of the electrode either in the atrium or in the ventricle. This decision is left to the cardiologist when implanting the device into a patient.

4.2.3 User Interface

The interface for the PROV910 will be done by a serial connection. This connection will go from a PC to an ES449 SoftBaught evaluation board. This board contains the driver to decode the serial message and also a LCD to display useful information. This evaluation board also contains a MSP430F449 connected to the PROV910 through wires. For future work the communication should be done through magnetically coupled loops.

4.2.4 PROV910 programmability and safety goals

Programmable goals:

- Basic Pacing Rates : From 32 to 120 bpm in steps of 2
- Pulse Widths : 20 values from .07 to 1.5 ms
- Pulse Amplitudes : 10 values from 3 to 7.5 V
- Refractory Period : 21 values from 200 to 500 ms
- Hysteresis : 20 values from 2 to 40 bpm
- Sensitivities : 16 values from .4 to 6.4 mV
- Pacing polarity : Unipolar / Bipolar
- Sensing polarity : Unipolar / Bipolar
- Upper rate in Trigger mode : 51 values from 80 to 180 bpm

Battery Monitoring goals:

- Magnetic Response: BOL (Begin of life) = 96 bpm
  ERI (Elective replacement) = 84 bpm
  EOL (End of life) = 74 bpm

- Elective Replacement : When the remaining capacity of the battery is 15% (Vbat = 2.3V)
- End of Life : When the remaining capacity of the battery is 5% (Vbat = 2.1V)

Protection goals:

- Run away limit : The pacemaker will not deliver stimuli at a rate larger than 200 bpm.
- No DC current path : All path to the heart will be coupled with caps to avoid dc currents.

Service Life Goals:

It is expected that PROV910 can achieve at least 10 years of service life when pacing a 100 % at 60 bpm, with a 3 V amplitude, a pulse width of 400 ms and a load of 500Ω.

4.3 Microcontroller selection process

The main attribute of the microcontroller to be chosen is that it must have very low power consumption. This means that it must be able to work down to at least 2V and consume supply current in the μAs order when in standby mode. In this section a comparison of different μCs will be performed. The μCs were chosen from

a broad sample that passed through a preliminary elimination process. This first elimination was based purely on minimum operational supply voltage. Those that remained to be compared in more detail were: a) MAXQ2000, b) PIC18LF242, and c) MSP430F1611.

The comparison of these μCs is made in terms of 1) average current consumption, 2) available power-down modes, 3) clocking system, and 4) pin leakage. A short explanation describing the importance of each feature for the design of PROV910 is also presented. Ranks are used in the decision process to determine which μC fits better the needs of the research; the rank functions as a weight in the final evaluation. The microcontroller with highest evaluation is the one selected.

4.4 Low Power Consumption Features

The following features are essential to obtain low power consumption. The information for comparison of each microcontroller was obtained from the respective datasheets provided by Maxim [36], Microchip [37], and Texas Instruments [38]. This comparison of low power microcontrollers is a significant contribution in the field of low power embedded systems. Designers does not have to go through all the comparison process since the following analysis serves as reference for future work using low power microcontrollers. The methodology followed for the comparison that will be presented can be easily used to compared new available low power microcontrollers.

4.4.1 Average current consumption

The average current consumption is essential to determine battery life. The average current has two components: the standby current and the active current. Figure 4.4 shows the typical behavior of a microcontroller changing operation modes.

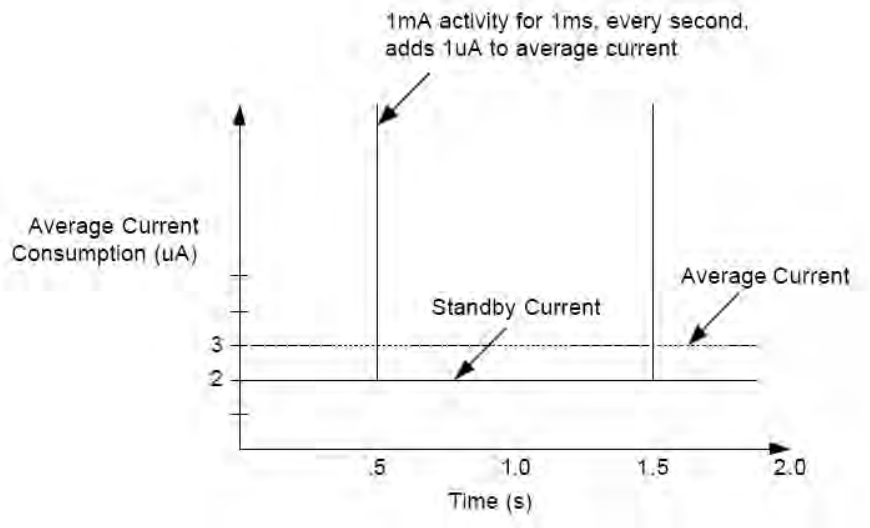


Figure 4.4. Average Current Consumption [39].

A μC should have a wake up time small enough to take care of interrupts in real time. This wake up time should be at least in the μs range to be insignificant when compared with the active time. If this assumption is accomplished, then the average current may be calculated using the following equation:

$$I_{average} = I_{standby} + \frac{I_{active}}{t_{base}} \sum_{i=1}^{n} t_{int}(i) + \frac{I_{peripheral}}{t_{base}} \sum_{i=1}^{n} t_{peripheral}(i) \qquad (4.1)$$

where $I_{standby}$ is the constant standby current and $I_{interrupts}$ is the current consumption due to handling interrupts. $I_{interrupts}$ is composed of the active current (I_{active}) of the μC, the base time (t_{base})which is the time in which a cycle is completed and the time consumed taking care of a particular interrupt (t_{int}).

It is important to notice that the active current depends on the μC's master clock frequency and the supply voltage. It is therefore imperative to define the master clock with the minimum frequency in which a proper real time operation can be achieved.

A real time operation is one that accomplishes a specific task before the dateline is done. For example, as discussed in section 4.2.4 the range of time for the *basic pace interval* is from 500 ms to 1.9 s ($Interval(ms) = \frac{60000}{Rate}$). This means that in order to achieve real time operation, the task performed by *basic pace interval's* interrupt must be done at least at half the minimum dateline time, that is 250 ms. A detailed analysis of real time operations will be presented in Chapter 5.

Since the standby current is constant throughout all cycles, this is the most critical characteristic and should be minimized. Therefore it has the highest rank. Table 4.1 presents the characteristics along with the ranks for each of thecurrents. Notice that the μC with better performance in a given characteristic is underlined.

Table 4.1. Current consumption characteristics.

Character-istics	Conditions	maxq2000	pic18lf242	msp430f1611	Rank
Sleep Current	All oscillators off RAM retention only	.7 μA	<u>.1 μA</u>	.2 μA	5
Standby Current	Only 32kHz Crystal is on, interrupt enable	4.8 μA	14 μA	<u>1.1 μA</u>	10
Active Current	Master Clock = 1MHz (PIC, f_{master}=4MHz) V_{supply}=2V	5000 μA	300 μA (1 mA)	<u>290 μA (400 μA)</u>	7

The decision is done as follows:

Microcontroller	Characteristics	Ranks	Score
maxq2000	0	0	0
pic18lf242	1	5	Score = 5
msp430f1611	2	7 + 10	<u>Score = 17</u>

As shown in the table above, msp430 offers better performance in terms of average current consumption.

4.4.2 Power down modes

Power down modes enable the microcontroller to meet low current consumption. A μC has several power down modes providing different levels of functionality. For a pacemaker, we need a power down mode that consumes the least amount of current but at the same time monitors ports and timers to take care of interrupts generated. This is not possible in sleep mode, since this mode turns off all the μC's peripherals and only can be woke up with a reset signal which cannot be delivered by the pacemaker itself.

Table 4.2. Maxq2000 power down modes.

Power Down Modes	Description	Current Consumption
PMM1	Allows one system clock is 256 oscillator cycles	190 μA
PMM2	Runs from a 32kHz crystal oscillator.	4.8 mA
Stop Mode	The external oscillator, system clock and all processing activity is halted. No interrupts available.	.7 μA

Table 4.2 shows the low power modes of maxim'm microcontroller maxq2000. This one has only 3 low power modes which consume too much current to be considered for the design. Table 4.3 shows the eight different modes in which pic18lf242 can operate. These modes are obtained by manipulating the clock frequency source. Although these μCs offer great flexibility and customization, the current consumption

is still too high. Finally Table 4.4 shows the low power modes for the msp430. The

Table 4.3. pic18lf242 power down modes.

Power Down Modes	Description	Current Consumption
LP	Low Power Crystal	14 μA
XT	Crystal/Resonator	500 μA
HS	High Speed Crystal/Resonator	600 μA
HS + PLL	High Speed Crystal/Resonator with PLL enabled	15 mA
RC	External Resistor/Capacitor	300 μA
RCIO	External Resistor/Capacitor with I/O pin enabled	300 μA
EC	External Clock	10 mA
ECIO	External Clock with I/O pin enabled	10 mA

msp430 offers a variety of modes which truly consume very low current. The LPM3 is particularly appealing for the design of the pacemaker since it offers a very low current consumption while maintaining interrupts enabled, a feature which satisfies the real time demands of the pacemaker. These features put the msp430 as the stronger candidate at this point, for the design at hand. However, although low current consumption is important, the way clocks enters or exit the low power modes is important too. Next section will compare the clock system of the microcontrollers to examine how fast clocks respond to a given interrupt.

4.4.3 Clock systems

The clocking system of any microcontroller is critical for low power consumption. Applications may enter and exit various low power modes several times a second. The ability to get into and out of the low power modes, and process data quickly, is crucial because current is wasted when the CPU waits for the clock to become

Table 4.4. msp430f1611 power down modes.

Power Down Modes	Description	Current Consumption
AM	All clocks are active	300 μA
LPM0	CPU and MCLK are disabled AUXCLK and SCLK are active	50 μA
LPM2	CPU, MCLK, SCLK are disabled DCO and AUXCLK are active	11 μA
LPM3	CPU, MCLK, SCLK, and DCO are disable. AUXCLK enabled	1.1 μA
LPM4	All clocks and CPU disabled	.1 μA

stable. Some μCs have a two-stage clock wake-up providing a low frequency (usually 32.768kHz) clock to the CPU while a high frequency clock is being stabilized, which can take up to a millisecond or longer. This is the case of maxq2000 which needs 65,536 oscillations to achieve stability while a ring oscillator provides the clock for the CPU.

Table 4.5. Comparison of microcontroller's wake up time.

Character-istic	Conditions	maxq2000	pic18lf242	msp430f1611
Wake-up time	Time from standby to active (f = 1MHz)	33 ms	2 ms	6 μs

As it can be seen in Table 4.5, the msp430 has a superior performance in terms of the wake up time, taking only 6 μs to be stable. This is about 300 times faster than the pic18lf242 and 5500 times faster than the maxq2000.

4.4.4 Pin leakage

Pin leakage is often overlooked when designing low power systems. However, in a demanding system as the PROV910 design, pin leakage is crucial. Typical microcontrollers have pin leakage currents around 1 μA. In a 20 pin package, this traduces into 20 μA. This is the case of pic18lf242 which has a maximum leakage current of 1μA and 28 pins (if a SO-28 package is used). Therefore the total leakage current is around 28uA which is more than the total current budget of the system. Table 4.6 compares the leakage current for the different microcontrollers. Again the msp430 satisfies the low power constrains of the system.

Table 4.6. Comparison of microcontroller's leakage current.

Character-istic	Conditions	maxq2000	pic18lf242	msp430f1611
Leakage current	Pin at high impedance Vcc or Vss applied to pin	100 nA	1 μA	50 nA

4.4.5 Final Microcontroller Selection

The above analysis demonstrated that in all the aspects considered for low power consumption, the MSP430 was superior to its counterparts. It is also important to point out that the MSP430 has peripherals that are optimized for low power consumption. The peripherals were designed so they can be individually enabled or disabled when needed. Furthermore, some peripherals have the ability to enable or disable themselves automatically. An example of an intelligent peripheral is the ADC12. If the ADC is not actively converting data, it turns off automatically. But it also has the capacity of turning on automatically if a conversion is triggered. Other useful peripherals are the Direct Memory Access modules (DMA) that can transfer large blocks of

data without CPU intervention. Using the DMA with the ADC is a winning combination, since long signals can be converted and stored without CPU intervention. This information can be processed by the CPU later in a single wake up event minimizing power consumption. The MSP430 complies with all the specifications lay down in chapter 3, it contains a 16-bit RISC CPU core with a 3-stage pipeline system and Von Neumann architecture. The next chapter will present the software specifications and design.

CHAPTER 5

Pacemaker's Software Design

Pacemakers, as any embedded system, are composed of two parts: software and hardware. The software assures the correct operation of the hardware. Although there are several papers exploring the hardware implementation of the front end, which is composed of an amplifier, filter and level detector [40] [41] [42] [43], and few for the output stage implementation,which is composed of a voltage multiplier and switches [44] [45], this researcher could not find any reference of a control software source used for pacemaker's hardware. Moreover, at present, there is not any design implementing a pacemaker using a general purpose microcontroller. This chapter presents the control software implemented in an msp430 microcontroller to control a prototype VVI pacemaker. The development and methodology to design low power pacemaker software are discussed in details. This software is intended for educational purposes and is accessible to anyone interested in the field of biomedical equipment. This chapter is basically composed of two parts: the software specifications and the software design.

5.1 Software Specifications

The software specifications resume the requirements needed to accomplish a functional software. Well defined software specifications lead to robust and reliable software. For this reason this section is entirely dedicated to define and explain the software specifications for PROV910.

5.1.1 Hardware partition

Microcontrollers have several internal hardware peripherals (see Chapter 3). Thus the need to classify hardware as internal or external. This is the first step in the software specifications.

Internal Hardware

The most basic function of a pacemaker is to generate stimuli at specific time intervals. Therefore, the internal hardware should include at least 4 timers to keep track of the basic pace interval, the escape interval (hysteresis), the refractory period, and the pulse width. Although these intervals are the time requirements previously presented in Chapter 4, there exist others time intervals that should be generated by internal timers as well. These are the EGM interval, the battery and electrode impedance interval, and a timer for the digital to analog converter. These intervals are described next.

- EGM interval

  This interval defines the rate at which the electrogram (EGM) is sampled and converted to digital data. The electrogram is different from the electrocardiogram since the former is the signal sensed by the intracardiac electrode and the latter is the signal sensed by external leads using an electrocardiograph. Figure

5.1 shows a typical ventricular electrogram compared to a typical electrocardiogram. The electrogram has valuable information that cardiologists can use to diagnostic heart diseases. Therefore it is important to include this feature in the PROV910.

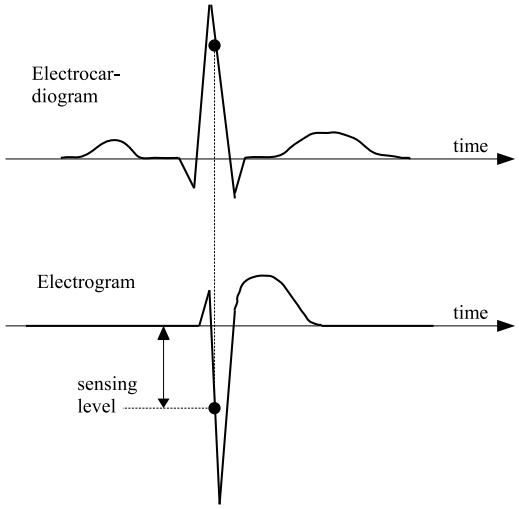


Figure 5.1. Typical ventricular electrogram [46].

- Battery and electrode impedance interval

  It is important to monitor the voltage of the battery energizing the pacemaker since a power failure is critical for the well-being of a patient. Another important parameter is the electrode impedance since a broken or not well insulated electrode will lead to failure to capture the heart and hence in an ineffective pacemaker. However, although these parameters are very important, they do not change rapidly. This means that it would be a wasteful approach to monitor these parameters at the end of every basic pace interval (typically between 60 - 80 bpm). It is more efficient to have a longer time interval to monitor them, between 1 min to 10 min.

- Digital to analog converters

The front end of the pacemaker needs programmable voltage references to set the sensitivity and the pulse amplitude. These voltage references are implemented using a couple of timers and PWM techniques to generate a DC voltage level.

The above specifications identify the need of internal timers with the capability of generating at least 8 different time intervals. Besides the timers, it is important that the microcontroller include internally an Analog-to-Digital Converter (ADC) to obtain the electrogram of the heart. As it was mentioned in Chapter 4, the msp430f1611 has an ADC optimized for low power consumption with a 12 bit resolution.

To monitor the battery voltage, the internal Supply Voltage Supervisor (SVS) of the msp430f1611 is used. The SVS can be programmed to check if the supply voltage gets below certain voltage value. If a low voltage condition occurs the SVS sets a flag indicating the event. It also has the capability of restarting the microcontroller.

Additionally, an internal comparator is required to monitor the electrode impedance. This is because, to check the impedance, a discharge time interval should be measured at a specific voltage level. This level can be set by an internal reference and a comparator. Further discussion of the electrode impedance will be presented later in this chapter.

Although highly unlikely, sometimes internal timers can fail. In this case PROV910 needs a backup timer that delivers a stimulus at a constant rate. RC circuits are suitable for low cost timers. Since this backup timer is only a temporary timer (the pacemaker need to be changed in case of failure), it only needs to deliver pulses at a rate of about 60 bpm. To implement this RC timer an I/O port with interrupt capability is needed.

Finally an internal UART and a DMA are necessary to communicate externally. The

msp430f1611's universal asynchronous receive/transmit (UART) has interrupt capabilities and can be set at several baud rates. Moreover, the hand shaking protocol can be configured easily in the UART control registers. The Direct memory Access (DMA) is essential since it can transfer large amount of data, like large EGM samples, from memory to external peripheral without CPU intervention. Since CPU consumes large amounts of current, the longer is in sleep mode the best.

External Hardware

The external peripherals are defined by the specialized requirements of the pacemaker. There are only two external stages: the front end and the output stage. Figure 5.2 shows the block diagram of the front end of a pacemaker. The front end senses small signals with many noise components. Since the sensing of the signal can be performed using unipolar or bipolar sensing, the first stage of the front end is a difference amplifier. The difference amplifier converts a differential signal into a single ended signal. After this first stage, a filtering stage and an amplification stage follow. The last block of the front end is an external comparator that compares the output of the filter/amplifier and a programmed referenced voltage.

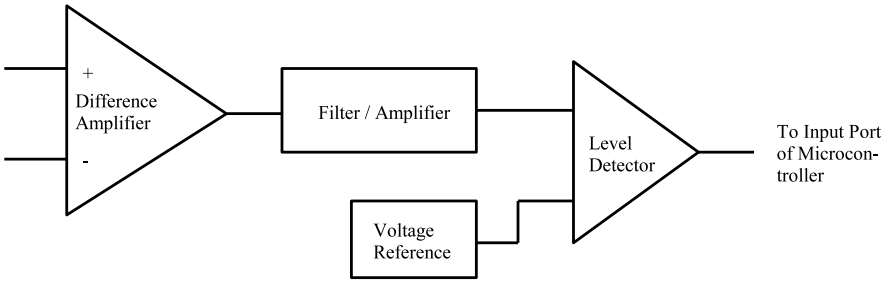


Figure 5.2. Block diagram of the pacemaker's front end.

The output stage consists of a charge pump, a complementary clock generator, a comparator and several switches. This stage should be able to triple the battery

voltage to generate a high voltage pulse that captures the heart. Figure 5.3 shows the block diagram of the output stage. The output stage should multiply the battery voltage to a voltage defined by the DC generator timer inside the msp430f1611. Note that a voltage divider should be used to step down the high voltage and be able to input it to the comparator without permanent damage. When the output voltage stored in the capacitor is larger than the reference voltage, the clock generator is disabled causing the charge pumping function to stop until the voltage stored in the capacitor is smaller than the voltage reference.

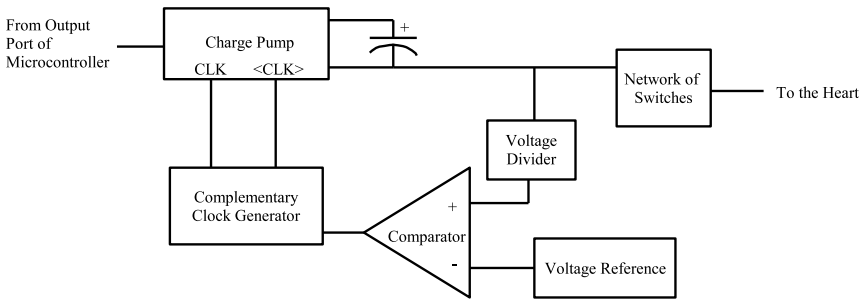


Figure 5.3. Block diagram of the pacemaker's output stage.

5.1.2 Internal hardware considerations

The external hardware can be controlled via I/O ports, while internal hardware has more control capabilities since every piece of the msp430f1611's hardware can be controlled by modifying certain registers. Since microcontrollers also include internal RAM and ROM, it is necessary to verify that the available memory in the msp430f1611 is sufficient for the intended application. It is also important to verify that there are sufficient available ports to control external hardware.

This section will outline the specifications for the microcontroller in terms of: (a) Peripherals, (b) I/O ports, (c) RAM requirement and (d) ROM requirement.

MSP430F1611 hardware peripherals

The following internal peripherals are needed for the design:

- Timers with capacity for eight different time intervals
- 12 bit resolution low power ADC
- Supply voltage supervisor with programmable voltage thresholds (2.1V to 3.0V)
- Comparator
- Ports with Schmitt trigger inputs and interrupt capability
- Universal asynchronous receive/transmit (UART)
- Direct memory Access (DMA)

MSP430F1611 I/O ports

The following I/O ports are indispensable to control properly all the external hardware. There are two port categories: timing ports and operational ports.

The timing ports are used to output the external signals that control the pulse, the refractory period and the RC timer. The operational ports are the ones that control all the functional parts of the pacemaker.

The timing ports required by the control software are:

1. Pulse signal port.
2. Refractory signal port.
3. Blank signal port.

4. RC oscillator port. Need interrupt capability.

The operational ports required by the control software are:

1. Sensitivity voltage reference (DAC1).
2. Pulse amplitude voltage reference (DAC2).
3. Sensing polarity.
4. Pacing polarity.
5. Input from external comparator. Need interrupt capability.
6. Input to ADC.
7. Comparator On/Off
8. Rosc input. Used to stabilize the Digitally Controlled Oscillator (DCO) over temperature to 800kHz.
9. Input to internal comparator.
10. I/O port charge/discharge Electrode impedance.
11. RC timer output port.
12. TurnOn RC signal port.
13. UART receiver port. Need interrupt capability.
14. UART transmitter port.
15. XOUT port. Watch crystal is connected here. A 5.1Mω resistor should be connected to this port for stability since the supply voltage goes down to 2.1 V

16. XIN port. Watch crystal is connected here.

17. Reed switch. Need interrupt capability.

18. External reference positive (+) port.

19. External reference negative (-) port.

As it can be seen from the I/O ports outlined above, the msp4301611 satisfy the port requirement since it contains 48 I/O general purpose ports. Furthermore, of those 48 ports, 16 have interrupt capability activated by either the raising or the falling edge of the input signal. Moreover, Schmitt triggered inputs avoid false interrupts.

RAM memory requirements

The data that will be stored in RAM are the EGMs, status variables, histograms, memory buffers, and Stack. RAM is preferred over Flash, since the latter consume 7 mA to write. [38]. Stack memory is vital since every interrupt uses the stack as a temporary storage for the status register and program counter. If the Stack is not properly selected the software will crash, which would be catastrophic for the application at hand. Figure 5.4 shows the RAM requirement for the different data that should be stored. Using this RAM distribution, an EGM of 12 seconds can be stored when using a sampling rate of 400 samples per second (400 Hz), which is a typical sampling rate for cardiac signals [19].

The total RAM required should be at least 5,023 words of memory space which translates into 10,046 bytes. The msp430f1611 attain this requirement given that it has a 10 kBytes (10240 bytes) of RAM memory space.

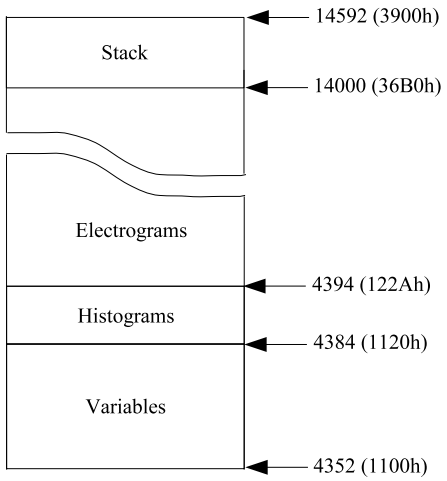


Figure 5.4. RAM memory distribution.

ROM memory requirements

The ROM memory is where the program resides. It typically is programmed one time only, although the msp430 has the ability to rewrite it at any moment if the proper sequence is used. However, as already mentioned this is not an option for the current design due to the high current consumption. The ROM should be distributed between software code, and parameters' tables. These tables are stored in ROM since if for any reason the pacemaker is reset, it will be able to fetch the default parameters from ROM and continue with its operation. This makes the software more reliable due to the fact that ROM memory can store data for a minimum of 100 years without damage [38].

The code should not occupy too much space given the fact that the program is developed using ASSEMBLY. A good estimate for the program size is 3 kBytes. However, the size occupied by the PROV910's parameters table can be accurately calculated.

Table 5.1. Pacemaker's parameters memory consumption.

Parameter	number of steps	Total Bytes in ROM memory
Basic Pace	45	90
Pulse Width	20	40
Refractory Period	21	42
Hysteresis	20	40
Sensitivities	16	32
Upper Rate in trigger mode	32	64
Magnetic Response	3	6
Unknown	50	100
Total	207	414

Table 5.1 presents each parameter with its corresponding size in memory. From Table 5.1 it can be seen that at least 4 kBytes of ROM memory are necessary for the software design. The msp430f1611 satisfies by far this specification since it has a flash memory of 48 kBytes.

Figure 5.5 illustrates where the I/O port are located in the msp430F1611 along with some external circuitry. A more detailed schematic including all the external hardware will be presented in Chapter 6.

From this analysis, we may conclude that the msp430f1611 has the required internal hardware to implement PROV910's design. The specifications presented above are a good foundation to proceed to the software design.

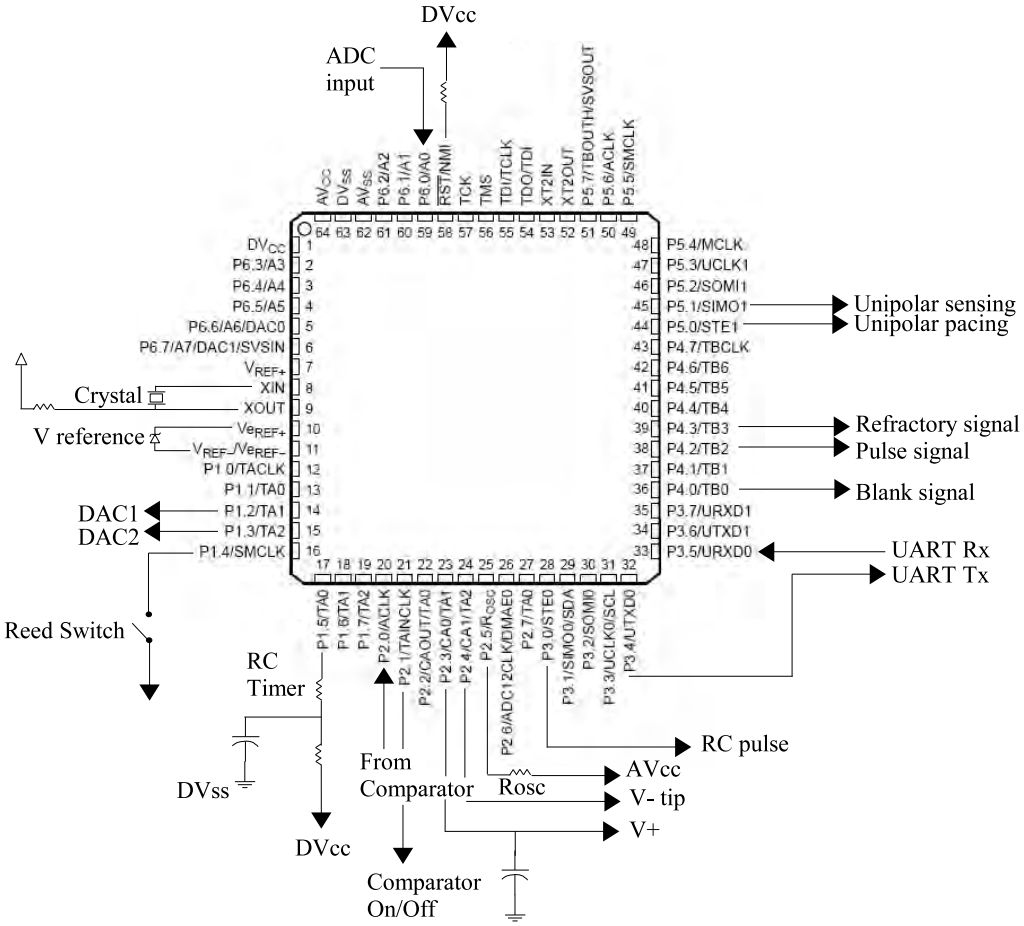


Figure 5.5. I-O port map for the pacemaker design.

5.2 Software Design

The goal of the software design is to achieve functionality with the minimum amount of current consumption. With this goal in mind the following design methodology is followed:

1. Research in low power software for embedded systems.

2. Investigate current consumption of the msp430f1611 in the available modes.

3. Generate timing diagrams for all tasks.

4. Create flowcharts for all tasks.

5. Code analysis.

6. Functionality verification

5.2.1 Low power embedded software

The increasing relevance of software's power consumption in embedded systems has promoted several research efforts recent years. The approaches for software power analysis reported in literature fall in two categories: architectural analysis and instruction level analysis. The first approach is discarded for the current research since architectural analysis is only useful if one is designing a microprocessor or at least has the internal structure of a processor core. However, the instruction level approach is suitable for the current research since it gives insight information of how programming instruction behaves in terms of power consumption.

The idea behind instruction level power analysis is to measure the current consumed by the processor when executing a particular instruction. This current can be the average current [47] or the instantaneous current [48] consumed by the μC.

The average current power analysis has been widely used in literature [49, 50]. Any processor consumes certain amount of average power while executing a program. This average power is expressed as:

$$P_{average} = I \times V_{cc} \tag{5.1}$$

where $P_{average}$ is the average power, I is the average current and V_{cc} is the supply voltage. Since power is the rate at which energy is consumed, the energy consumed by a program is given by: E = P x T, where T is the execution time of the program. This execution time in turn is given by: T = N x τ, where N is the number of clock cycles taken by the program and τ is the processor's clock period [47].

There are several methods to measure the average current. The more simple method is to open the circuit and use a standard off the shelf digital ammeter. Execution time of programs is measured though detection of specific bus states using a logic analyzer. However, the logic analyzer is only useful to calculate energy, to measure current the ammeter is enough. Since typically a program completes execution in a short period of time, it is necessary to use programs executing in an infinite loop to be able to measure the average current. Although, other more sophisticated methods exits to measure current [51, 52], using oscilloscopes or real time data acquisition systems, the trends found in all of them are similar. Given the fact that the interest of this research is focus in the trends rather than in the precise current consumption values, the analysis proposed in [47] is used to characterize the instruction set of the msp430F1611. Nevertheless, the results presented by these other techniques are used to complement experiment's data.

The instruction level power analysis is a valuable tool for the software design. It provides the means to estimate the current consumption when the msp430 enters active mode. Furthermore, an efficient program can be generated by characterizing the msp430 instruction set. This is true since the programmer can minimize the current consumption of the software by proper selection of the instructions and addressing modes used.

The instruction level power analysis models the power consumption in terms of energy

costs. The total energy of a particular program is composed of a base energy cost and an Inter-instruction cost.

Base energy cost

The base cost is the cost of a single instruction run in a loop. This loop should be large enough to converge the average current but small enough to avoid cache misses. These should be avoided to conserve the integrity of the base cost. A good estimate for the loop size is 120 steps. Interrupts should be disabled to obtain reasonable results.

Consider now the instruction processing technique used by msp430's CPU. The msp430 has a 3 stage pipeline. This means that at any particular point in time it will be processing three instructions at the same time. But, as shown in [47] this is not a big concern since the energy consumed in a pipeline cycle, E_{cycle}, is equal to the energy consumed by the instruction, E_{ins}. Since the total energy consumed in a cycle is given by $E_{cycle} = E1_{I_1} + E2_{I_2} + E3_{I_3}$ and the energy of an instruction is expressed as $E_{ins} = E1_{I_1} + E2_{I_1} + E3_{I_1}$, when using an adequate loop, I_1 becomes $I_1 = I_2 = I_3$ resulting in $E_{cycle} = E_{ins}$. In this case, the average current is $\sum_j Ej_{I_1}/(V_{cc} \times \tau)$ which is equal to the reading obtained by the ammeter.

A multimeter was used to measure the average current. The Agilent 34401A is a $6_{1/2}$ digit, high-performance digital multimeter. The multimeter used a continuously integrating, multi-slope III A/D converter to measure the average current [53]. The multimeter's integrating time is 10 power-line cycles, that is 167ms $(1/60 \times 10)$. Since the execution time of the programmed loop is much smaller than the integrating time, a stable reading is obtained. Figure 5.6 shows the test-bench used to measure the

average current. The resolution for this test-bench was set up to 10 nA. The msp430 flash emulation tool was used, since it has a jumper (J7) to measure current directly.

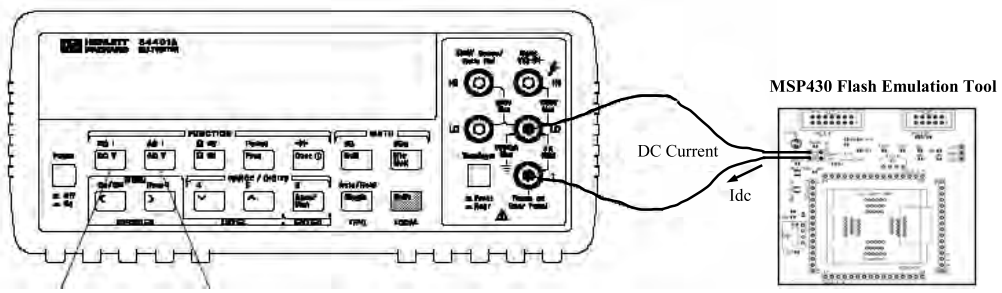


Figure 5.6. Test-bench to measure the average current.

The msp430 has 27 core instructions. The base energy cost of these 27 instructions was measured. The msp430's clock cycles are defined by the instruction itself. They are actually defined by the addressing mode used for the source and destination. Because of this, two addressing modes were used to characterize the current consumption of the instruction set. The register mode is chosen as the lower limit since it consumes the minimum clock cycles, and the absolute mode gives the maximum value due to its large clock cycle consumption. Table 5.2 was generated using this approach.

Note that instructions "call" and "reti" were not included in the analysis. The reason for excluding them is that it was not possible to generate a loop sequence including only this instructions. The minimum and maximum currents were measured using addressing modes with the shortest and longest clock cycles, respectively. The register mode has the shortest clock cycle, while absolute mode has the longest. Although absolute mode has long clock cycles it is widely used when programming the msp430 since it increases portability of the software. The clock cycles are based on an operating frequency of 800kHz, which will be the operating frequency for PROV910's software design.

As it can be seen in Table 5.2, the current consumption oscillates from 296 μA to 361 μA. As expected, instructions with longer execution cycles consumed larger quantities of average current.

The number of existing 1's in a particular instruction can increase significantly its current consumption. This has been widely discussed in the EASY project [54]. They demonstrated that there is a linear relationship between the number of ones in an operand and the amount of current consumed. Another experiment was conducted to verify this influence of 1's in the current consumption. Table 5.3 shows the result of using FFFFh (1111111111111111b)as input for the instruction's operands. The absolute mode was used to explore the worst case. The results of the experiment support the findings of the EASY project. The operand's content does influence the power consumption of the instruction set. Note that a reduced instruction set was tested for this experiment. The reason is that these instructions are more commonly used. For example the extended sign instruction (sxt) is only useful if the program works with negative numbers, which is not the case in this research.

Inter-instruction energy cost

The base cost presumes that instructions of the same type run continuously. However, in reality, programs are composed of multiple instructions performing quite different tasks. This task's difference causes switching activity in hardware, which in turn consumes power. This activity depends on the present state of the inputs and the previous state of the circuit. Since a program has several states, it is difficult to predict the state changes during program execution. However, an experiment can give some insight about the existence of this additional inter-instruction cost in the msp430.

Table 5.4 shows the result for the inter-instruction experiment. There is clearly an

Table 5.2. Msp430f1611 instruction set power characterization.

Instruction	Current μA		Cycles	
	Minimum	Maximum	Minimum	Maximum
Double - Operand Instruction (Format I)				
mov.w	302	330	1	6
add.w	314	334	1	6
addc.w	315	338	1	6
sub.w	332	335	1	6
subc.w	334	340	1	6
cmp.w	334	338	1	6
dadd.w	318	334	1	6
bit.w	311	336	1	6
bic.w	317	336	1	6
bic.w	317	336	1	6
bis.w	313	337	1	6
xor.w	317	337	1	6
and.w	313	339	1	6
Single - Operand Instruction (Format II)				
rrc.w	316	316	1	4
rra.w	313	317	1	4
push	342	342	1	5
swpb	310	316	1	4
sxt.w	313	318	1	4
Jump Instruction (Format III)				
jeq / jz	297	339	2	2
jne / jnz	296	339	2	2
jc	298	334	2	2
jnc	297	340	2	2
jn	297	339	2	2
jge	298	336	2	2
jl	298	336	2	2
jmp	n/a	361	2	2

Table 5.3. Influence of operand's content in current consumption.

Instruction	Current μA		Cycles
	0000h	FFFFh	
mov.w	332	359	6
add.w	334	360	6
cmp.w	338	354	6
bit.w	336	357	6
bic.w	336	351	6
bis.w	337	362	6
xor.w	337	376	6
and.w	339	361	6
rra.w	317	347	6
sub.w	335	361	6
swpb	316	347	6

Table 5.4. Evaluation of Inter-instruction cost.

	Instruction	I_{base} (μA)	I_{meas} (μA)	I_{calc} (μA)	Cycles
Set 1	mov.w	330	342	336.6	6
	add.w	334			6
	cmp.w	338			6
	bit.w	336			6
Set 2	bis.w	337	340	336.8	6
	bic.w	336			6
	bit.w	336			6
	mov.w	330			6
Set 3	bis.w	336	338	331	6
	jc	298			2
	swpb	316			4
	xor.w	337			6
Set 4	bit.w	336	340	331.7	6
	jn	297			2
	and.w	339			6
	rra.w	317			4
Set 5	xor.w	337	342	337	6
	sub.w	335			6
	mov.w	330			6
	cmp.w	338			6

overhead cost for the sequence. The measured sequence current (I_{meas}) is larger than the calculated average current (I_{calc}). For example, for set 1, the following equation is used

$$I_{calc} = \frac{361 \times 2 + 330 \times 6 + 338 \times 6 + 336 \times 6}{26} = 336.6\mu A \tag{5.2}$$

This difference between measured and calculated values proof that there is an inter-instruction cost as specified in [47, 48]. Note that the cost of a jump instruction was added for each measurement (361 x 2). This is because each set must be repeated in an infinite loop to be able to measure the average current.

The experiments presented above validate the results presented in [47, 48, 49, 50, 51, 52, 54] for the msp430. It is worth to mention that this power characterization of the msp430 instruction set is the first instruction set profile generated for the msp430. This profile can be used by software designers to choose the appropriate instructions for their applications while considering the power impact in the software performance. An embedded software design guideline is developed using the information obtained by the experiments run above and results of previous research.

Embedded software design guideline

- *Reduce as much as possible memory access.* Read and write operations to memory consume more current and typically need multiple clocks cycles. The memory's current consumption depends on the distance between the current and previous address. This is further encouraged by [55], where a functionality approach is used to estimate power consumption of a processor. This paper defines 5 functionalities that can be defined for any processor: 1) Fetch and decode, 2) Branch, 3) Write to register, 3) Arithmetic and logic, and 5) Load, store and

stack. The load store (memory access) functionality consumed nearly double the current when compared to others functionalities.

- *Choose instructions that consume less current.* For the msp430, bit manipulation instructions consumed less current. To take advantage of this, software should pass information by bit manipulation whenever possible.

- *Use addressing modes with few clock cycles.* The longest an instruction takes to execute the more power it consumes. Typically the Register mode is the faster addressing mode available. This implies that efficient use of the registers is a must.

- *Use operands with the least amount of 1's when possible.* As it was shown previously there is a dependence between the number of 1's in an operand and current consumption. In the msp430 1's can add up to 10% more current consumption. When using bit manipulation to set reset flags, choose the reset state for the longest state.

The above analysis minimizes current consumption while in active mode. However, this alone is not sufficient to achieve the low current consumption required for PROV910's design. To achieve ultra low power operation it is necessary to understand the current behavior in the different available modes. The following section discusses this current consumption for the msp430.

5.2.2 Msp430f1611 current consumption

To obtain the full capability of the msp430 in terms of power consumption, the low power modes should be used. Figure 5.7 shows the typical current consumption of the msp430 in each mode.

As it can be seen from Figure 5.7, LPM4 consumes the lowest current. However, for the application at hand, this mode is not useful since all clock signals are turned off, which cannot be since then there is no possible way of managing the time intervals to generate the required pulses. On the other hand, mode LPM3 consumes approximately 2 μAs and activates the low power watch crystal as the auxiliary clock. This attribute satisfies the needs of the design. Due to the slow rate of the heart signal the 32 kHz watch crystal can be used.

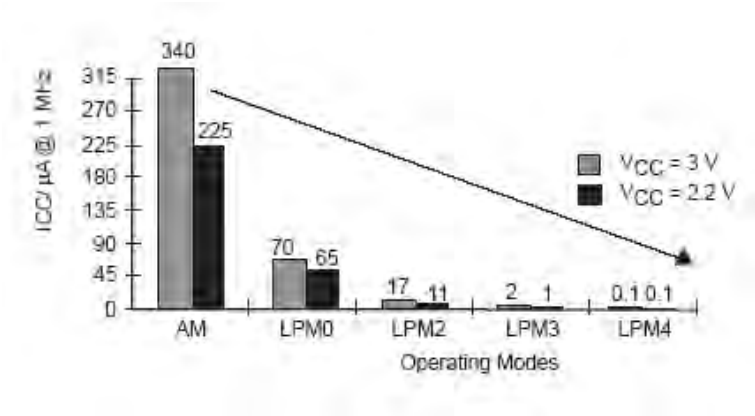


Figure 5.7. Typical current consumption for each operating mode.

When the msp430 wakes up from LMP3 it activates its Digitally Controlled Oscillator (DCO) at a frequency setup by the programmer. The DCO frequency which is also the master clock frequency was chosen to be 800 kHz. Such frequency is quick enough to execute several instructions while in active mode and then get back to LPM3. Moreover, 800 kHz is the default frequency of the microcontroller, which means that there is no need of inverting code in setting the DCO frequency [56].

The msp430x161x datasheet [38] define active mode current in terms of Vcc and Master clock frequency. Equation 5.3 define the active current in terms of the supply voltage. PROV910 needs to work from 3V or 2.8V down to 2V. In this case the

worst case current consumption occurs while Vcc = 3V. For 3V, the nominal current consumption is 500 μA with a maximum of 600 μA.

$$I_{AM} = I_{AM} \text{ @ } 3V + 210\ \mu A/V \times (Vcc - 3V) \tag{5.3}$$

The active mode current also depends on the system frequency. Given that the system frequency was set to 800 kHz. equation 5.4 is used to calculate the actual current consumption of the system. For 3V and 800 kHz the active current nominal value is 400 μA with a maximum of 480 μA

$$I_{AM} = I_{AM} \text{ @ } 1Mhz \times f_{system} \tag{5.4}$$

The LPM3 current is specified in the msp430x161x datasheet. Again, the worst case current occurs at 3V with a nominal value of 2 μA and a maximum of 2.8 μA.

The battery used for the design is a Panasonic poly-carbonmonofluoride lthium coin. This battery is used only for prototyping purposes. The actual battery used in a pacemaker is a lithium-iodine battery, discussed in Chapter 2. Panasonic's battery has half the capacity of an actual pacemaker's battery, that is 1Ah. The self-discharge rate is 1 μA per year. So the actual total capacity is 900 mAh. The goal is to consume 20 μA or less giving a service life of 900mAh 20 μA = 45000 hrs or 5.13 yrs. If a lithium battery is used then the service life would double to 10.26 yrs.

To effectively profit from the ultra low power consumption, it is necessary to develop a well defined time scheme. The following section discusses the time diagrams for each pacemaker's task.

5.2.3 Time diagrams

Time diagrams are useful to organize interrupt requests. Precaution should be taken when organizing interrupts since run conditions, stack overflow, and boundary conditions should be avoided [[57]]. These are defined as follows:

- Run conditions - refers to interrupts events that occur very close to each other, and both access the same global variable.
- Stack overflow - occurs when a string of interrupts happen consecutively, producing multiple push operations which could fill the stack.
- Boundary conditions - similar to run conditions. A boundary condition occurs when an interrupt is generated exactly at the moment a variable it needs is been updated.

A task is defined as an action accomplished in a predefined period of time. The PROV910's tasks are defined as follows:

1. Sensing / Pacing the heart
2. Adquire heart Electrogram
3. Monitor Battery / Electrode status
4. External communication

Sensing / Pacing the heart

Figure 5.8 shows the distribution of interrupts for the Sensing / Pacing of the heart. Port 2 generated interrupts when a spontaneous QRS is sensed. This interrupt is represented with dashed lines since it is generated randomly by the diseased heart.

Hence, port 2 interrupt's interval can not be defined. As Figure 5.8 shows, there are four interrupts controlled by software. To minimize current consumption, the active time t_int_xxx, should be as short as possible for every interrupt. Also, special care is needed to avoid waking up the msp430 unnecessarily.

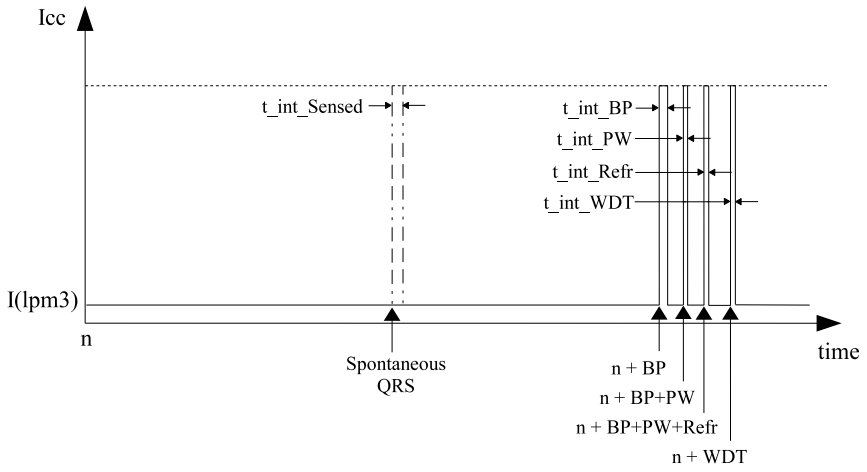


Figure 5.8. Timing diagram for the sensing / pacing task.

The four interrupts presented above have the highest priorities since they execute continuously. There are other interrupts that occur less frequently. This is the case of the battery and electrode status check. Figure 5.9 shows the timing diagram for these tasks. Note that in this example 60 Watchdog interrupts must occur before one Status interrupt is performed. Since the watchdog timer can be set to generate interrupts every second, for this example the status check is done every minute.

There are other instances in which certain tasks are not executed unless they are requested by the external programmer. An example of this type of task is the EGM interrupt. Figure 5.10 presents the timing diagram for an EGM request. As was mentioned previously, the EGM has a finite RAM memory allocation that can store up to 20s of data at a rate of 400 samples per second.

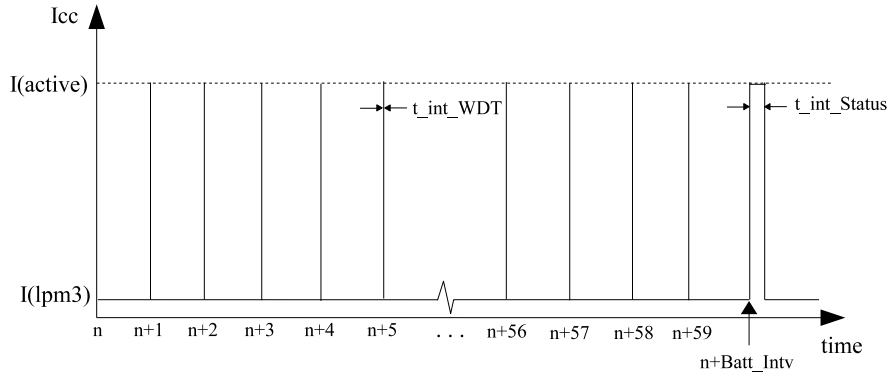


Figure 5.9. Timing diagram for the Battery and Electrode check task.

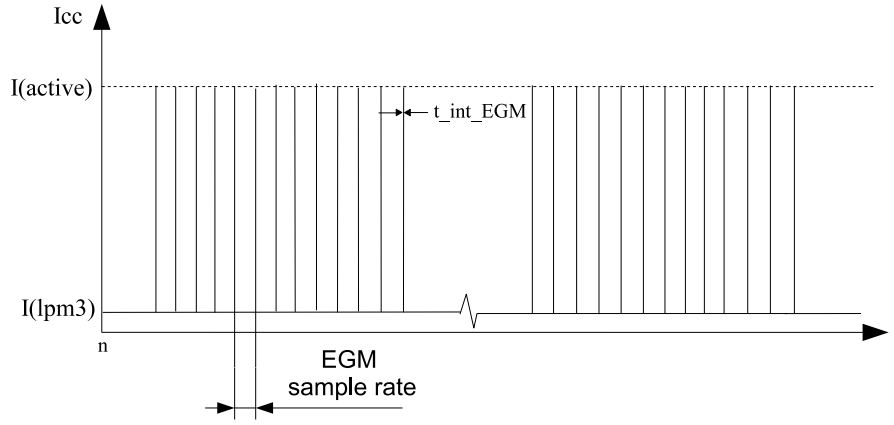


Figure 5.10. Timing diagram for the electrogram acquisition request.

In terms of power consumption, one should be careful of the methodology used to implement EGM acquisition since the internal ADC consumes a considerable amount of supply current that can go up to 1.6 mA.

The last timing diagram, Figure 5.11, presents the interrupts for the external programmer. The external programmer can either send or request data. This interrupt basically consists in moving data from memory locations. There will always be two interrupts, the incoming and the outgoing interrupt, since every instruction received will be echo for verification purposes.

To avoid run conditions, flags will be used as much as possible. Variables will be

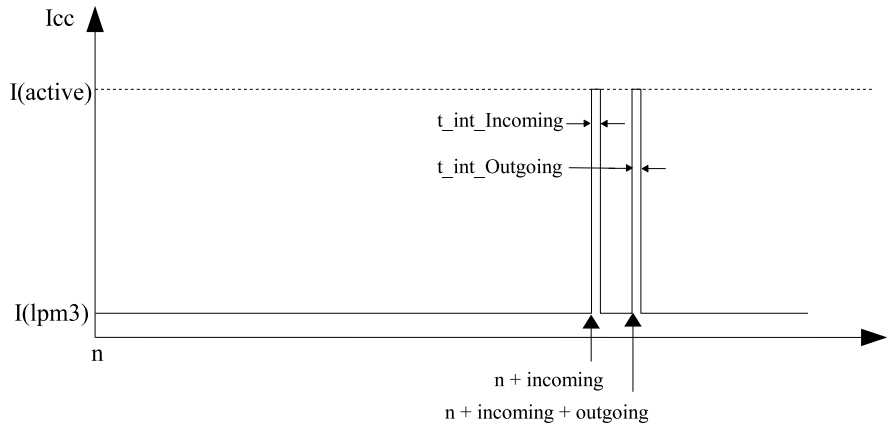


Figure 5.11. Timing diagram for the external programmer request.

defined within an interrupt. This means that a variable can only be updated within its respective interrupt routine. This approach avoids problems with boundary conditions. Now that the distribution of interrupts has been defined, it is necessary to discusse how the msp430 handles interrupt requests.

Msp430 interrupt

The msp430 has three types of interrupts: Reset, Non-Maskable and Maskable [56]. Non-maskable (NMI) interrupts are caused by oscillator faults, flash access error or a rising edge at the NMI pint. Maskable interrupts are those generated by msp430's internal peripherals. Each peripheral interrupt can be individually disabled, thereby offering great flexibility.

There is a 6 cycles latency, starting when the interrupt is accepted and ending when the first instruction in the interrupt handler is executed. Each time an interrupt is accepted the following logic is applied.

1. Any currently executing instruction is completed.

2. The Program Counter (PC), which points to the next instruction, is pushed onto the stack.

3. The Status Register (SR) is pushed onto the stack.

4. The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.

5. The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.

6. The SR is cleared with the exception of SCG0, which is left unchanged. This terminates any low-power mode. Because the Global Interrupt Enable (GIE) bit is cleared, further interrupts are disabled.

7. The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

After this logic is performed, the interrupt handler program is executed. At the end of execution, the Return from interrupt (RETI) instruction must be used to terminate the handler properly. It is important to mention that the interrupt's priorities are designated by hardware. Figure 5.12 shows the priorities for the msp430f1611.
Since TimerB has the higher maskable interrupt priority, timer B is used to perform the timing operations for the pacemaker critical parameters like basic pace, pulse width, etc. The following subsection presents the software's flow charts with a thorough discussion of the algorithm. The flowcharts presented in the following seciton are a significant contribution to the open academic literature since they illustrate pacemaker's software functionality. This flowcharts are the first presented in open literature and are suitable to explain the behavior of pacemakers to students at university level.

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up External Reset Watchdog Flash memory	WDTIFG KEYV (see Note 1)	Reset	0FFFEh	15, highest
NMI Oscillator Fault Flash memory access violation	NMIIFG (see Notes 1 & 3) OFIFG (see Notes 1 & 3) ACCVIFG (see Notes 1 & 3)	(Non)maskable (Non)maskable (Non)maskable	0FFFCh	14
Timer_B7 (see Note 5)	TBCCR0 CCIFG (see Note 2)	Maskable	0FFFAh	13
Timer_B7 (see Note 5)	TBCCR1 to TBCCR6 CCIFGs, TBIFG (see Notes 1 & 2)	Maskable	0FFF8h	12
Comparator_A	CAIFG	Maskable	0FFF6h	11
Watchdog timer	WDTIFG	Maskable	0FFF4h	10
USART0 receive	URXIFG0	Maskable	0FFF2h	9
USART0 transmit I²C transmit/receive/others	UTXIFG0 I2CIFG (see Note 4)	Maskable	0FFF0h	8
ADC12	ADC12IFG (see Notes 1 & 2)	Maskable	0FFEEh	7
Timer_A3	TACCR0 CCIFG (see Note 2)	Maskable	0FFECh	6
Timer_A3	TACCR1 and TACCR2 CCIFGs, TAIFG (see Notes 1 & 2)	Maskable	0FFEAh	5
I/O port P1 (eight flags)	P1IFG.0 to P1IFG.7 (see Notes 1 & 2)	Maskable	0FFE8h	4
USART1 receive	URXIFG1	Maskable	0FFE6h	3
USART1 transmit	UTXIFG1	Maskable	0FFE4h	2
I/O port P2 (eight flags)	P2IFG.0 to P2IFG.7 (see Notes 1 & 2)	Maskable	0FFE2h	1
DAC12 DMA	DAC12_0IFG, DAC12_1IFG DMA0IFG, DMA1IFG, DMA2IFG (see Notes 1 & 2)	Maskable	0FFE0h	0, lowest

Figure 5.12. Interrupt priorities for the msp430f1611.

5.2.4 PROV910's control algorithm flowcharts

The control algorithm of a pacemaker in essence is a time manager, counted by Timer B in this research. Since the msp430 should be in the standby mode as long as possible, the ACLK driven by a watch crystal should be used to keep track of time.

A watch crystal has a very stable oscillation frequency of 32.768kHz. By using this frequency, each count cycle becomes T = 1 / 32.768kHz = .03052ms. This means that the timer counts in steps of .03052ms. Equation 5.5 is used to calculate the

programmed value (tb_x) for a particular time period t_x:

$$tb_x = \frac{t_x}{.03052ms} \tag{5.5}$$

All interval parameters to program PROV910 are calculated using equation (5.5). It is important to evaluate the robustness of the watch crystal as a time reference. The watch crystal frequency drift over temperature is minimal. This can be proved by using equation 5.6 to calculate the frequency drift due to temperature changes [58]. The body temperature is 37 C and if fever occurs it can go up to 40C [59]. For 40 C (worst case), the frequency drift would be -.524288 Hz, which yields a .000002 ms variation. Hence, the watch crystal is a robust time reference for the control program.

$$\Delta frequency = frequency \cdot k \cdot (T_0 - T)^2 \tag{5.6}$$

Figure 5.13 shows the register organization. The allocation of PROV910 into registers allow reduction in power consumption (see section 5.2.1). When registers are used inside interrupts, the registers are pushed into stack so that the pacing parameter is not lost.

Registers 0 through 3 are special purpose registers so they can not be used to store pacing parameters. Registers 4 to 6 store the pointers to the programmed parameters. Also these registers contain some operational flags to communicate between interrupt's handlers, thus preventing the occurrence of run conditions. Registers 7 to 13 store the most frequently used pacing variables. Finally register 14 is used for logic manipulation (and, or, xor, etc.) and register 15 as a general counter. Figure 5.13 also presents the default values for flags and pacing parameters. Note than when possible flags were defined with reset as the default value. This approach is due the

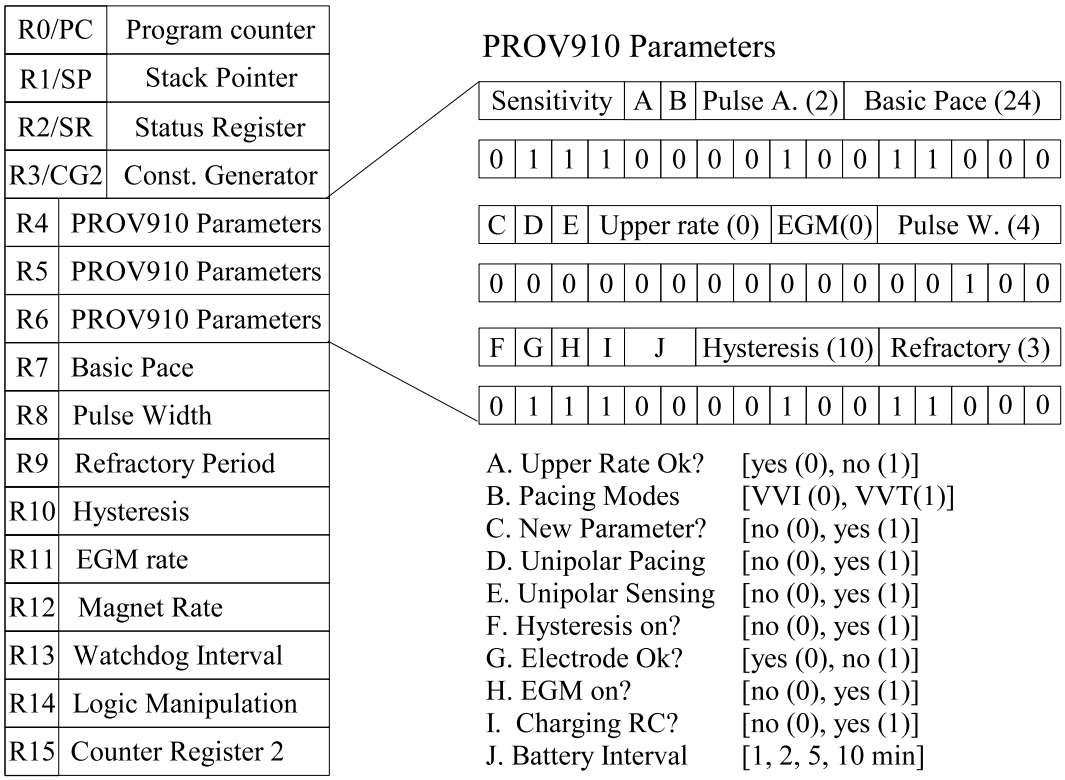


Figure 5.13. PROV910 Registers' Organization.

fact that zeros consume less power that ones.

The control program is composed of two functional groups: Modules and Interrupt handlers. Modules are substructures that aid in the software design of the main program.

The functional modules are:

- Time manager
- SVS
- Electrode

These modules are discussed first to establish the logical functionality of the mainloop. The mainloop has an nfinite loop structure. This structure has been widely used in

embedded systems due to its reliable response and low current consumption since all operations are performed within interrupts [57]. Figure 5.14 shows the mainloop flowchart.

The reset vector points to the first step. When working with the msp430, the first instruction should turn off the watchdog timer since the controller starts in Reset mode as default. If not reset in a certain period of time, the part will reset automatically. However, the watchdog timer is later used in interval mode. In interval mode the WDT works as a regular timer generating automatic interrupts every second.

To avoid the having random data in the registers all registers are cleared out and the default pace parameters are loaded in their respective registers.

The Battery and the electrode status are checked as a precaution. If the unit is reset due to a low supply voltage event, the battery status will be executed a short time after the reset. This provides useful information for diagnosis of the system and development of possible solutions

The final step in the mainloop is to enter into the Time Manager module.

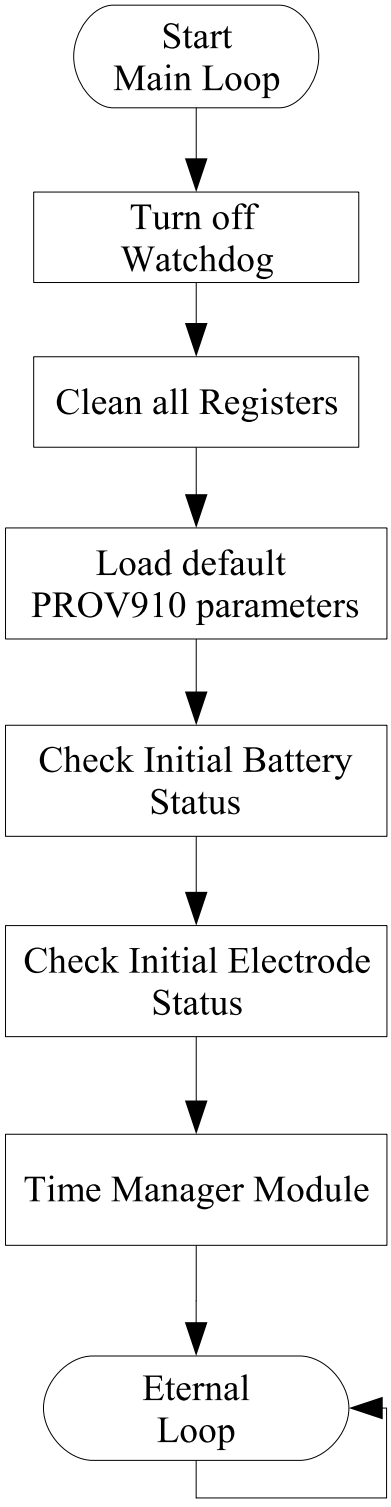


Figure 5.14. Flowchart of the Mainloop.

In the time manager module is where all the peripherals are setup to their respective programmed values. Also it setups the I/O p ports. Note that at the end of the module timers A and B are turned on. Timer B is used to count all the time intervals. Timer B can generate up to seven time interrupts, each of them individually maskable.

Timer A is used as a DAC. Timer A has two registers that can generate interrupts or that can generate pulse-width modulated (PWM) signals. If the duty cycle of a PWM signal is varied with time, and then filtered, the output of the filter is an analog signal [60]. Timer A is used to generate the voltage references for the sensitivity and the output voltage. The resolution of the Timer A is 5 bits, sufficient for the required references.

When the new parameter flag is set, the time manager module is invoked to setup all the pacing parameters. This module includes all the interrupt handlers.

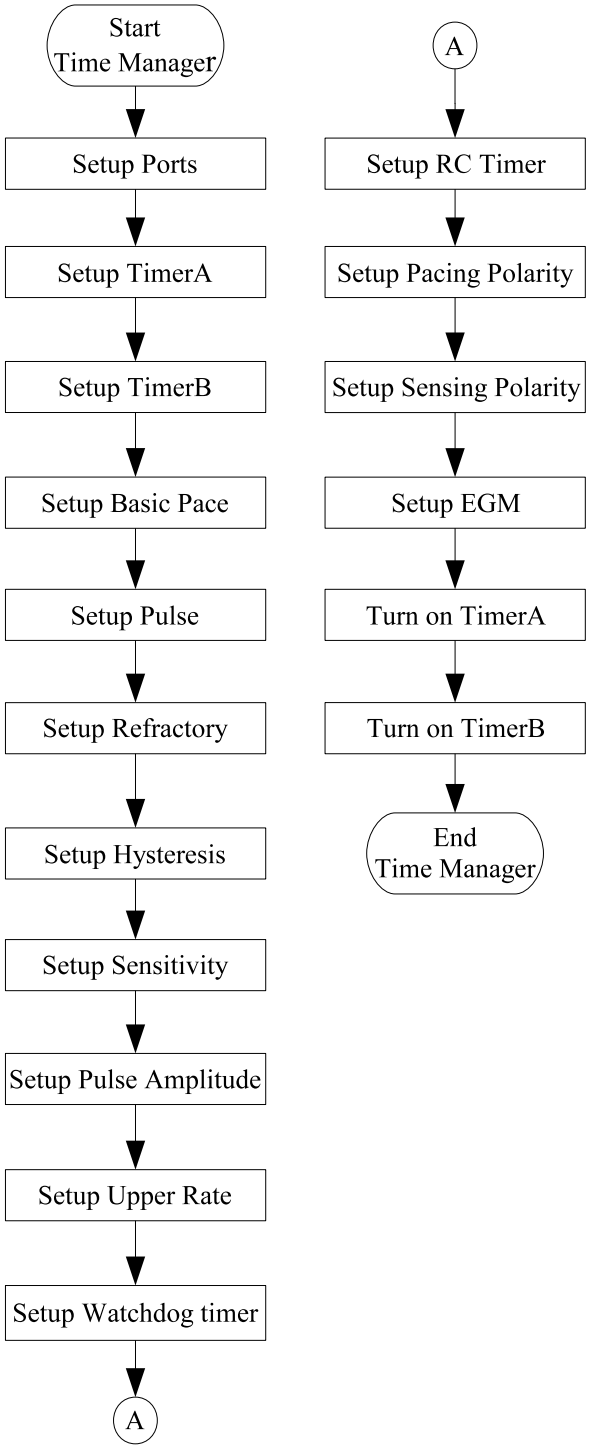


Figure 5.15. Flowchart of the Time Manager module.

The Supply Voltage Supervisor (SVS) module uses the internal msp430 peripheral with the same name. The peripheral has certain startup constrains as specified in [38]. These constrains do not allow the instantaneous usage of the SVS, hence certain software delay should be included to allow the SVS to stabilize. The larger delay occurs when the SVS is turned on, although an additional delay is required every time a new threshold voltage is programmed.

The SVS peripheral has 14 different internal generated voltage threshold. The maximum battery voltage is 3V, which are depleted to 2V throughout its service life. This means that thresholds within this range are required to check the battery status.

It is important to note that once the supply voltage gets below a certain threshold, this will not be used again for comparison. This is true because the battery used in the design at hand is not a rechargeable one.

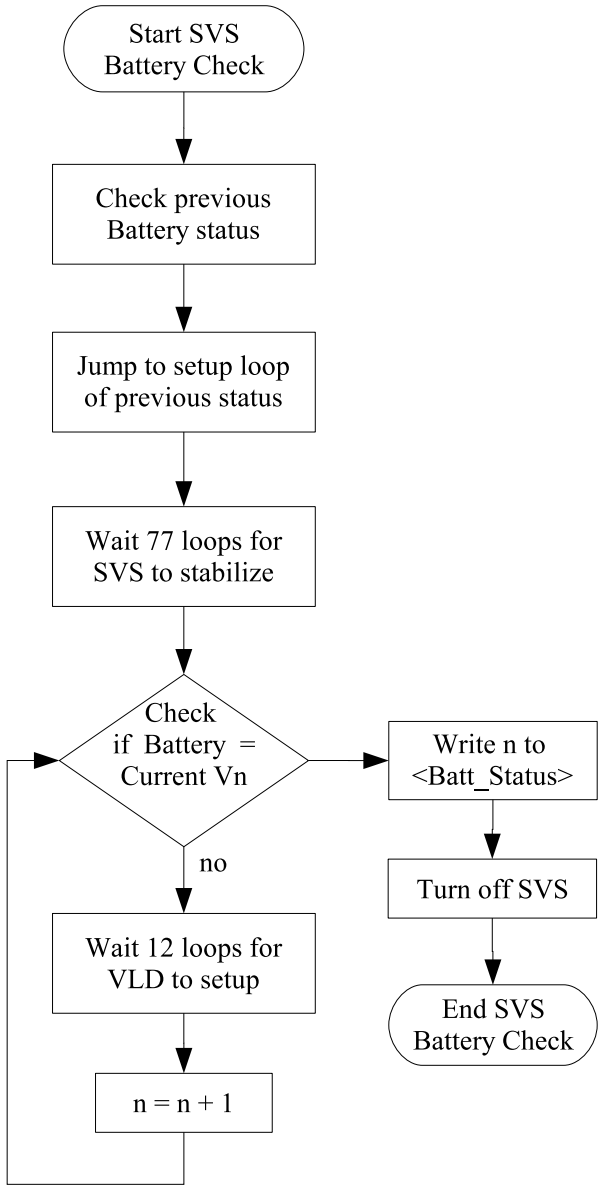


Figure 5.16. Flowchart of the Supply Voltage Supervisor module.

The electrode check uses the variation in discharge time of the RC circuit formed by the impedance of the heart and a capacitor. The typical impedance of the heart plus the electrode is around 500 Ω [46]. If this impedance goes above 1 kΩ, then the electrode is broken. In this case the module store a 4444 in memory to inform that the electrode was set due to high impedance. On the other hand, if the electrode presents an impedance below 250 Ω then there is an insulation defect. When this occurs a 2222 code is stored in RAM memory

The internal comparator is used to monitor when the capacitor discharges to .25Vcc. At this point the comparator sets its ouput to one. The capacitor is discharged through a 10 μF capacitors in series. These capacitors assure that all dc currents are isolated form the heart. Since the ACLK is too slow to measure this discharge time, the clock cycles of the DCO were used instead.

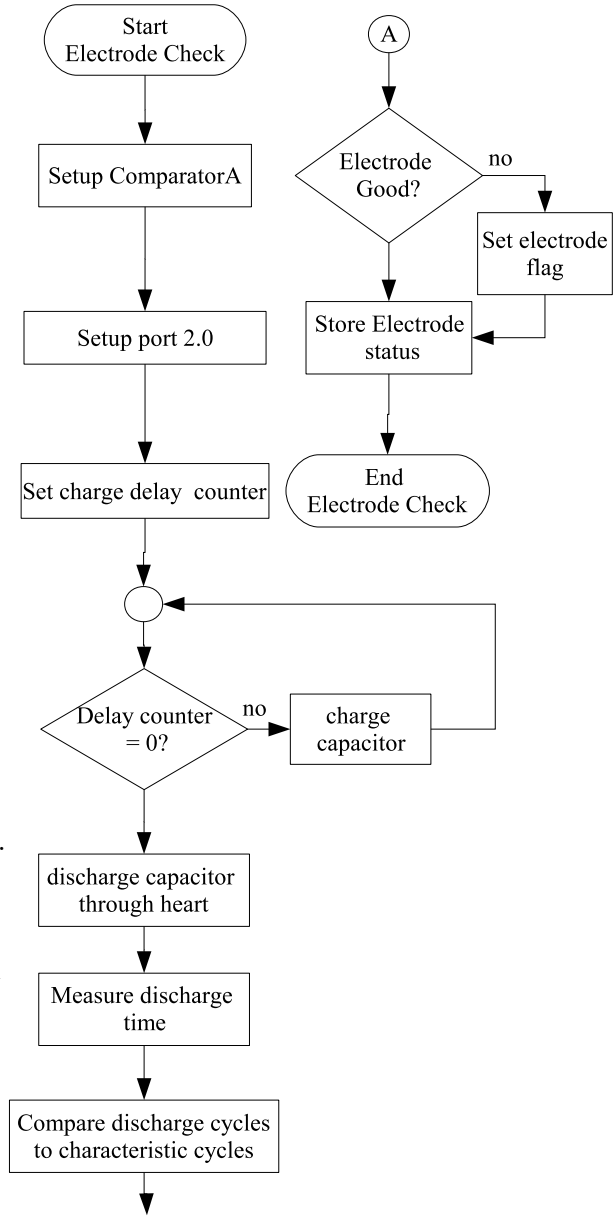


Figure 5.17. Flowchart of the Electrode Check module.

The modules are performed anywhere inside the main loop or by interrupts when needed. However, interrupts only occur when either a predefined time period elapsed generating an interrupt, or an external signal requests an interrupt. The following interrupts are defined for the PROV910's software:

- Basic Pace interrupt
- Pulse Width interrupt
- Refractory interrupt
- Electrogram interrupt
- Watchdog Timer interrupt
- Port1 interrupt
- Port2 interrupt

Interrupts are mostly controlled by hardware. This is the main difference between the functional modules presented above and the interrupts. The interrupt vector is an essential part of interrupts. The interrupt vector is a pointer to the location where CPU needs to start execution of the respective handler. The msp430 series allocate their interrupts in the higher addresses of Flash memory, that is 0FFFFh - 0FFE0h. This feature eased portability since all vectors are in the same memory allocation.

In the following pages the interrupt's flowcharts are presented with a brief explanation of their functionalities.

The Basic Pace is the most important interrupt since it activates the switch network that delivers the stimulus to the heart. Furthermore, the basic pace contains the time value which is the base for the pulse width and the refractory period. If no basic pace occurs, neither of the other two time intervals can be performed.

This interrupts handles both the VVI mode and the VOO mode. It tests if the reed switch is closed. If this is the case, then the external comparator is turned off and the respective magnet pace is performed. The magnet rate will depend on the battery status. Also, this interrupt checks if the output rate is smaller than 200 bpm. If not, an RC circuit which delivers a 60 bpm is turned on. This RC circuit stays on until an electrode check occurs. At this point Timer B takes control again of the Basic Pace. If everything is correct, then the Pulse and Blanking signals are set, the external comparator is turned off, and the programmed time value is added to the register preparing next BP interrupt.

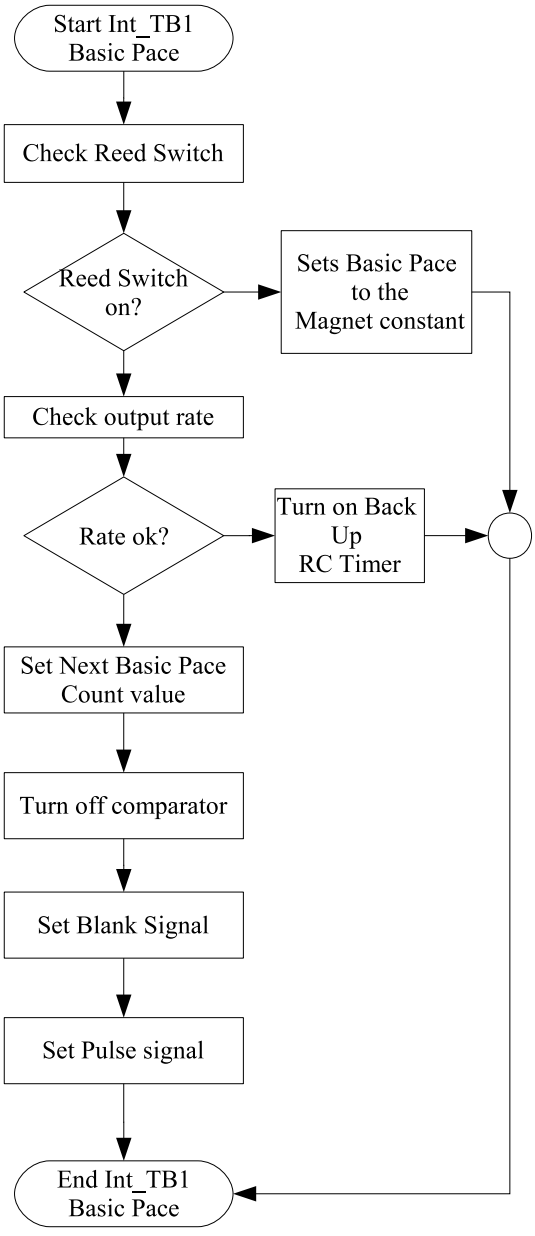


Figure 5.18. Flowchart of the Basic Pace interrupt handler.

The Pulse Width and Refractory interrupts are straight forward in their functionalities. The Pulse Width just resets the signal and updates its time value.

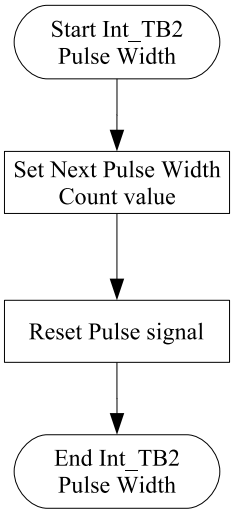


Figure 5.19. Flowchart of the Pulse Width interrupt handler.

The refractory interrupt resets the Blank signal and turns on the external comparator to permit sensing the heart again for any spontaneous QRS.

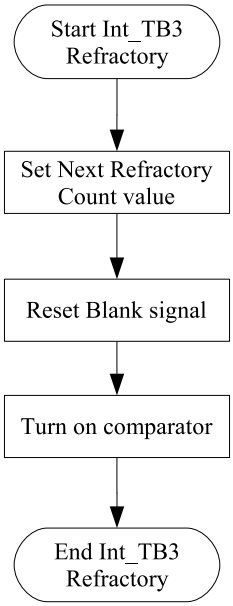


Figure 5.20. Flowchart of the Refractory Period interrupt handler.

This interrupt is enabled only when requested by the external programmer. The Electrogram use analog ports to acquire the analog signal coming from the output stage of the amplifier or the the filter. When not in use, the analog port should be put into high impedance to consume the lowest possible amount of current. The internal 12 bit ADC is used to convert the signal. The mode of conversion is single channel single conversion. This allows to control the sample rate by Timer B instead of using the internal oscillator of the ADC. The main concern while using ADC's internal oscillator is that it consumes power. By only taking one sample per interrupt, there is a significant reduction in power.

ADC's conversions are store sequentially in RAM using a circular buffer. If the last available address for EGMs storage is reach them the next address will start in EGM_start.

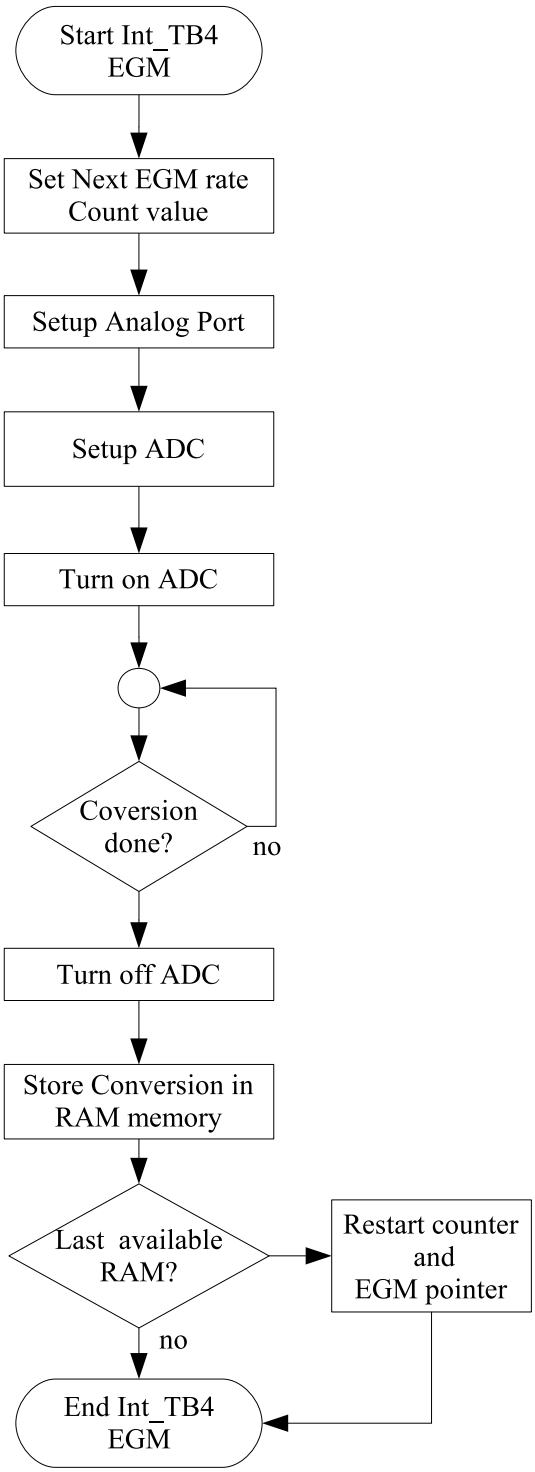


Figure 5.21. Flowchart of the Electrogram interrupt handler.

The main purpose of the Watchdog interrupt is to monitor if a new parameter has been sent. This interrupt is activated every second to ensure the prompt processing of any external programmer's request. If the new parameter flag is set, the interrupt handle pops the stack and jumps to the start of the time manager module. If the stack is not popped out, then an overflow condition will occur.

If the battery interval is done, the WD interrupt uses modules SVS and Electrode to check the status of both of them. In the case of the RC timer, this is always turned off. The reason for this is that the RC timer consumes a significant amount of power. However, there is a trade off between security and power consumption. If the RC timer was activated, it was because a run condition took place. If the RC timer is turn off to soon, the heart will receive a significant part of the run away. In the case the Run away continuous indefinitely, then the RC will respond reseting in every battery interval.

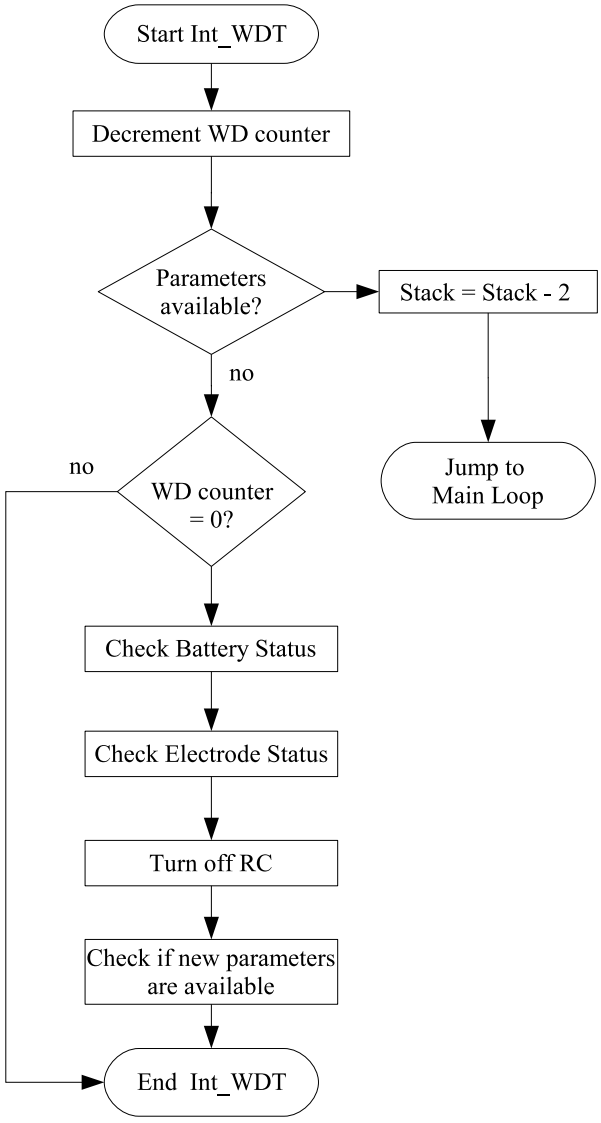


Figure 5.22. Flowchart of the Watchdog Timer interrupt handler.

Port 1 interrupt handler, shown in Figure 5.23, controls two tasks: The RC timer and the Reed Switch. Port 1 controls the charge/discharge cycle via its Schmitt trigger inputs. When the capacitor charges to a voltage above Schmitt threshold, the interrupt is accepted and the port is set to output low to start discharging the capacitor. Also, the falling edge is selected to generated interrupts. This complementary action is done to let capacitor charge and discharge cyclically. The pacing time is set by the RC values to a 60 bpm rate.

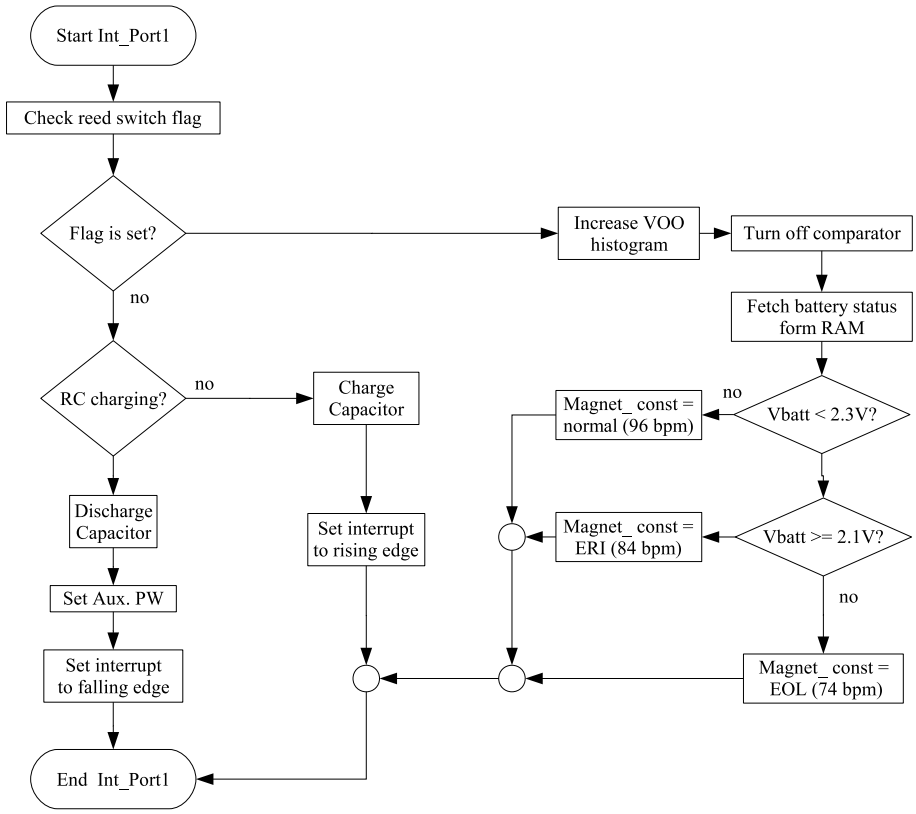


Figure 5.23. Flowchart of the Port1 interrupt handler.

When the reed switch is closed, the VOO mode is on. This switch is activated externally by a magnet. While the magnet is near the pacemaker it will continue in the VOO mode. The VOO rate depends on the battery status, as established in the requirements in Chapter 4.

The last flowchart presented in Figure 5.24 corresponds to the interrupt handler for port 2. The basic operation of this port is as follows: when an input high is received, it will inhibit or trigger an stimulus depending on the mode. There are other features related to patient safety. One of these is the evaluation the input frequency. If it is higher than 11 Hz, then the operation mode is changed to VOO. In trigger mode the stimulus is generated within this handler. This allows the monitoring of the trigger rate which is limited by the upper rate parameter.

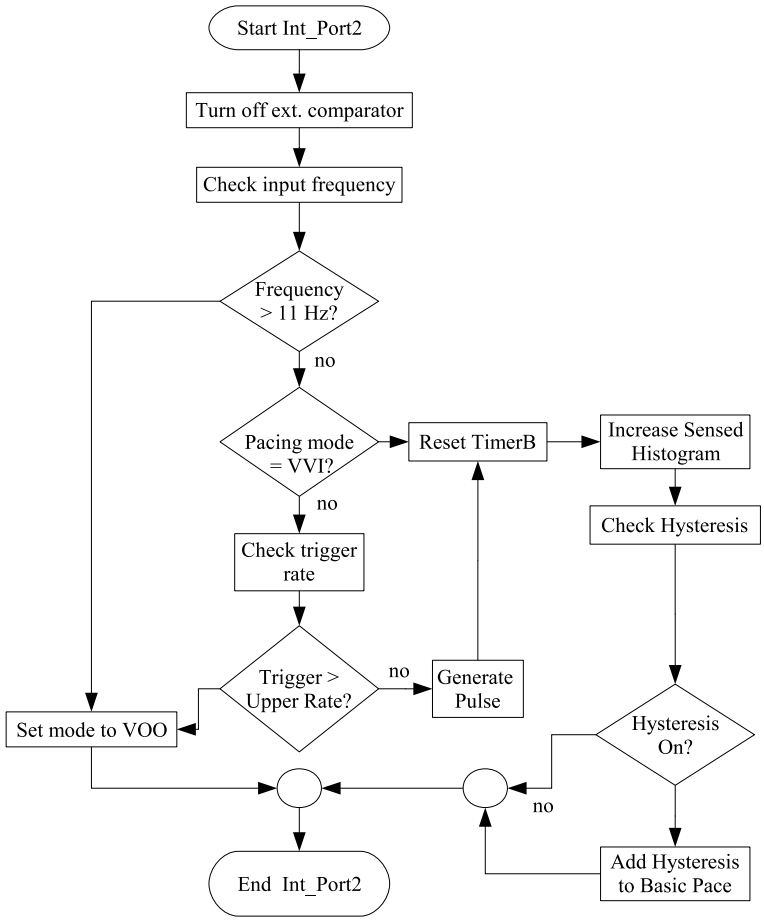


Figure 5.24. Flowchart of the Port2 interrupt handler.

The hysteresis is also set within this handler. If on, the additional delay for the sensed of spontaneous QRS is added to the basic pace.

The actual verification of the software parameters will be discussed in next chapter, along with the hardware verification. The source code of the control algorithm is available in Appendix A.

CHAPTER 6

Hardware Design and System Verification

The purpose of the hardware designed in this research is to validate the software in a simple platform. As was previously mention, there are cutting edge hardware designs for the front end and the output stage. However, a basic design and implementation of the front end and output stage is discussed. This implementation used discrete circuits and Printed Circuit Board (PCB) techniques to achieve the required hardware functionality. Finally a PROV910's system verification is presented.

6.1 PROV910's hardware requirements

As was stated in Chapter 5 the msp430f1611 offers a wide variety of internal hardware, reducing design's time and system's cost. However, the interface to the heart can not be implemented only using msp430's internal hardware. The reason is that the acquisition and stimulation of the heart have particular requirements that need special hardware. Basically two hardware's blocks are needed, the front end and the output stage.

The msp430f1611 cannot implement the front end since it does not offer internal filters that achieve the cutoff frequencies required to obtain a good reading of the heart signal. This is understandable since the cutoff frequencies of the front end are around 70 to 200 Hz [3], which are very difficult to achieved using integrated capacitors. Furthermore, the requirements for the output stage needs several capacitors in the 1 μF order to multiply battery's voltage. For these reasons, the design that is presented in the next sections is used to validate the msp430 software design. An explanatory note should be make here. The intention of the hardware designed in this research is not to achieve a cutting edge front end nor output stage design, but to have a prototype hardware that can demonstrate the validity of the software.

6.2 Front end design

As was stated in Chapter 5 the front end consists of a difference amplifier, a bandpass filter and level detector. the block diagram is presented in Figure 6.1. Each of this design blocks are discussed in the next sections.

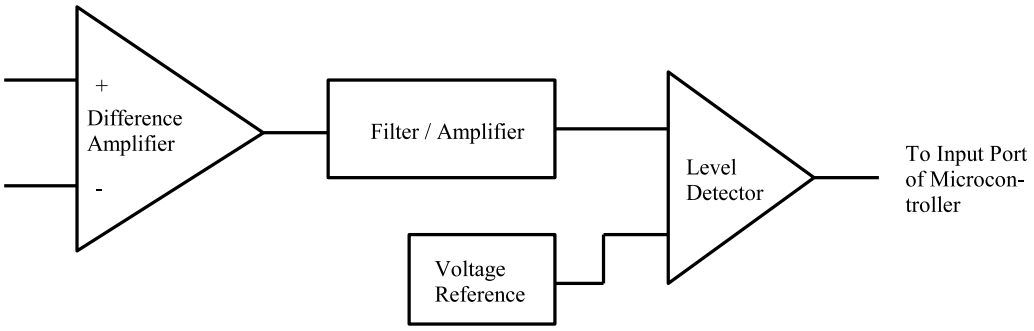


Figure 6.1. Block diagram of the pacemaker's front end.

6.2.1 Difference amplifier

The sensing of the heart is performed by a lead that connects the pacemaker unit to the heart. This lead can sensed the heart in two modes: unipolar and bipolar. The basic function of this first stage is to allow for both types of sensing. If a different structure is used, like an inverting amplifier, then only the unipolar mode can be achieve. There are different types of difference amplifiers, being the most popular the instrumentation amplifier. However, a typical instrumentation amplifier is composed of three operational amplifiers (OpAmps) which consumes three times the supply current of a single difference amplifier. Since the goal of this research is to obtain low power operation, the one-OpAmp difference amplifier is the one use to implement this first stage.

Figure 6.2 shows the schematic of the difference amplifier used. The difference amplifier have a single supply configuration since it will be supply by a battery. The common mode of the amplifier is set via a resistor network that set it to Vbat/2. This resistor network used resistor in the 1-MΩ order to limit the current consumption to μAs. AC-coupling capacitor are included to reject dc levels from the input signal. In this way the amplifier will only amplify the heart signal since it is an ac signal. Also a balanced design is used where $R_1 = R_3$ and $R_2 = R_4$. One concern over this structure as a difference amplifier is its low input resistance and second concern is the error caused by resistor's mismatches in the output signal. Let discussed these two separately.

The importance of the input resistance is better understood by illustrating the block diagram of an ideal voltage amplifier. Figure 6.3 shows the loading of an ideal amplifier. With the configuration shown the equation for the transfer function is:

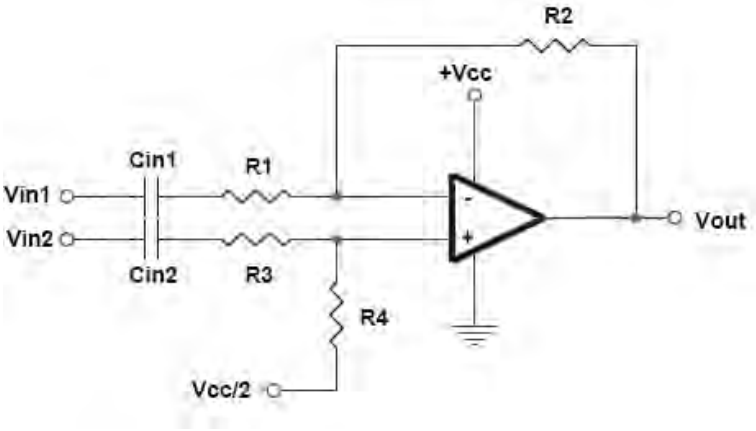


Figure 6.2. Schematic of the one-OpAmp difference amplifier [61].

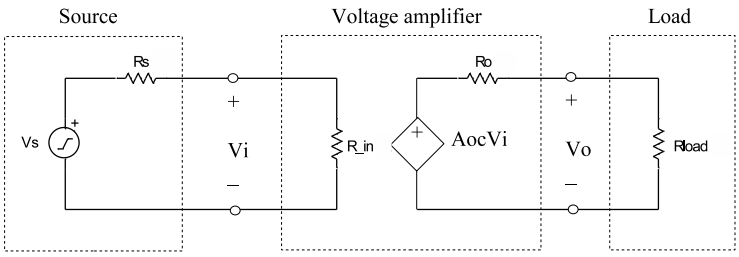


Figure 6.3. Schematic of an ideal amplifier [62].

$$\frac{V_o}{V_s} = \frac{R\_in}{R_s + R\_in} A_{oc} \frac{R_{load}}{R_o + R_{load}} \tag{6.1}$$

Equation 6.1 shows that a voltage divider is formed between the source and the input of the amplifier. The higher the input resistance the lower the voltage drop in R_s improving the precision of the transfer function. The load create by the electrode-heart network can be represent by the RC circuit shown in Figure 6.4. Typical values for Ch, Rf and Rs are 1.9 μF, 34.1 Ω, and 450 Ω, respectively, giving a typical impedance of approximately 500 Ω [**?**]. This means that a difference amplifier with an input resistance of 200 kΩ should acquire 99.75% of the heart signal ($\frac{200k}{200k+500}$) which is sufficient for the design at hand.

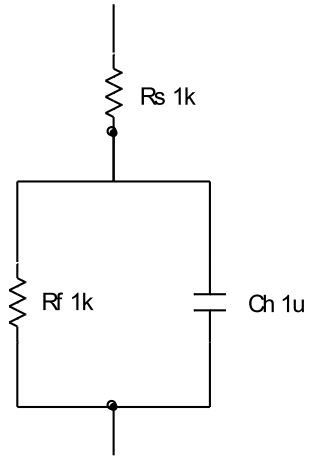


Figure 6.4. Equivalent electrode tissue impedance network [**?**].

The second concern is the resistor mismatches. This is especially important since the difference amplifier is used to reject common mode signals when sensing the heart in bipolar mode. The higher the rejection of common mode signals the higher the rejection of noise since noise is common to both inputs. Equation 6.2 express the effect of a resistor mismatch in the Common Mode Rejection Ratio (CMRR) of the difference amplifier.

$$CMRR_{dB} = 20log_{10}\left[\frac{1 + R_2/R_1}{\epsilon}\right] \tag{6.2}$$

where ϵ represents the imbalance factor of the resistors. To reduced the mismatch effect a buffer configuration, 1 v/v difference amplifier is used, with input resistors R_1 and R_2 set to 1 MΩ. As was stated above, this stage is only used to convert a double-ended signal to a single-ended, so this gain is sufficient. If resistors with a 1% tolerance and minimum width pcb traces are used the CMRR will be around 37.5 dB in the the worst case (ϵ = 4 x .01) and in the best case around 49.5 dB (ϵ = .01).

The Integrated Circuit (IC) OpAmp used to implement the difference amplifier is the OPA379 from TI. Table 6.1 presents the most important parameters for this OpAmp.

Table 6.1. OPA379 important parameters.

Parameter	Condition	OPA379			Units
		Min	Typ	Max	
Offset Voltage	Vs =5V		.4	1.5	mV
CMRR	(V_-) < Vcm <(V_-) - 1	90	100		dB
Input Bias Current	$V_s = 5V, V_{cm} <= V_s/2$		±5	±50	pA
Input Offset Current	$V_s = 5V$		±5	±50	pA
Input V_noise	f= .1 Hz to 10 Hz		2.8		μV_pp
Input V_noise Density	f = 1 kHz		80		nV / $\sqrt{Hz}$

With this OpAmp the error of the input offset on the output, which is given by equation 6.3, is .8mV.

$$E_O = \left(1 + \frac{R2}{R1}\right) V_O S \tag{6.3}$$

This offset error is a dc noise gain that moves up or down the ac signal coming from the heart. This is not critical since the thresholds are set by the cardiologist at the moment of implantation. Since the 1/f noise dominates at low frequencies , while white noise higher frequencies, and the frequency bandwidth is limited to 70Hz (imposed by the filter discussed in next subsection) noise should not be a concern. Also, is important to clarify that the actual input to the microcontroller is a pulse generated by the level detector, which can filter out additional noise by using an schmitt trigger configuration.

The schematic of the difference amplifier used is shown in Figure 6.5. This pcb was generated using EAGLE free version. The test signal used for the evaluation of the performance of the difference amplifier is a triangular waveform since the as will it represents more precisely the heart response as state by the tokyo standard [**?**].

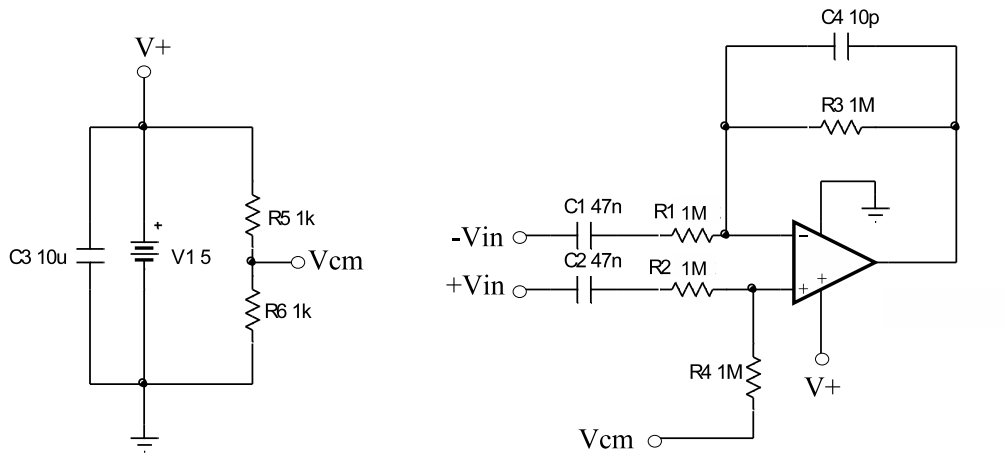
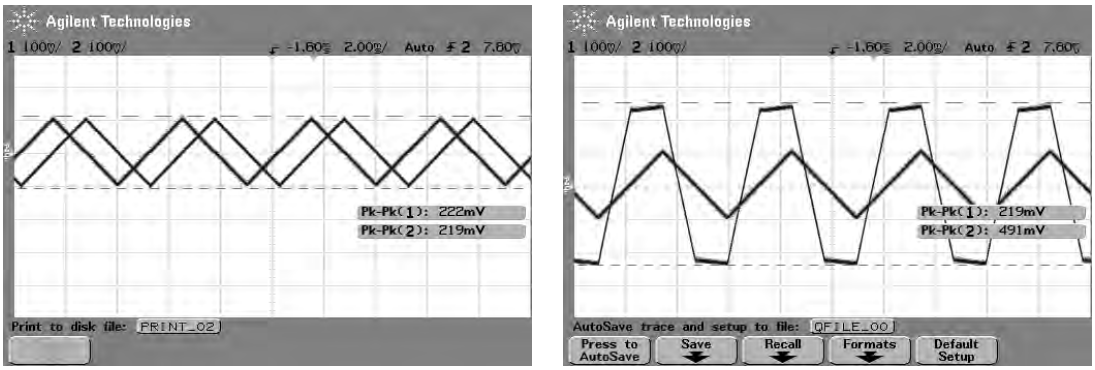


Figure 6.5. Schematic of the difference amplifier implemented.

the results for this stage where obtained using an Agilent 54622D Mixed-Signal Oscilloscope. Figure 6.6(a) presents the inputs to the difference amplifier, while Figure 6.6(b) presents is output response and Figure 6.7 presents the simulation for an ideal difference amplifier in Matlab. As can be seen from comparing Figure 6.6 and Figure 6.7, the difference amplifier is working properly.



(a) Measured Input (b) Measured Output

Figure 6.6. Measured response for the difference amplifier implemented.

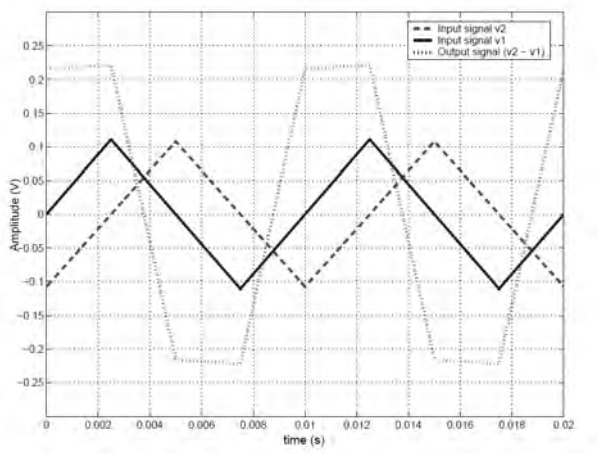


Figure 6.7. Simulate inputs and output response using MATLAB.

6.2.2 2 poles BandPass Filter

The second stage of the front end is a bandpass filter. Figure 6.8 shows the power spectra of a typical ECG. This figure shows that the main components of the heart signal are located at low frequencies ranging from .1 Hz to 80 Hz. The problem is that this range include the 50 and 60 Hz networks which produce a large amount of noise. However, it has been demonstrated that a bandpass using a range of frequencies from 80 to 200 Hz can be used to acquire the heart signal, obtaining good results [3]. This type of filter has been used in several generations of commercial pacemakers implemented by the CCC in Uruguay.

The bandpass specified removes two important sources of noise, the 60Hz noise and the intra muscle noise which has components near the 300Hz frequency and above. Although the filter can be implemented using a single OpAmp bandpass configuration, this was rejected. The reason is shown in Figure 6.9. When cascading bandpass filters the cutoff frequencies are move from 3 dB reducing the effective bandwidth of the signal.

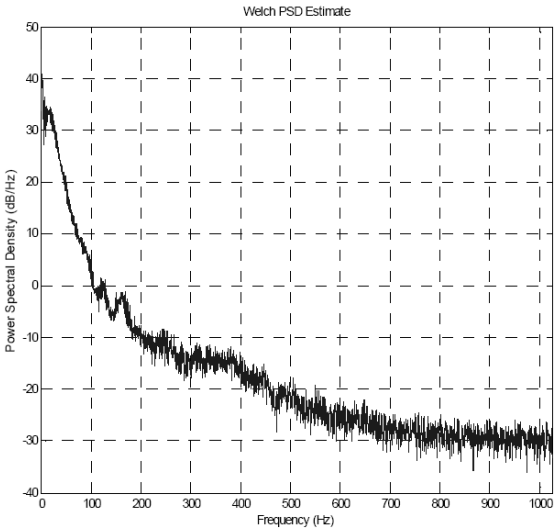


Figure 6.8. Typical Power Spectral density of an ECG.

A high pass and low pass filters were cascaded to implement the design. This

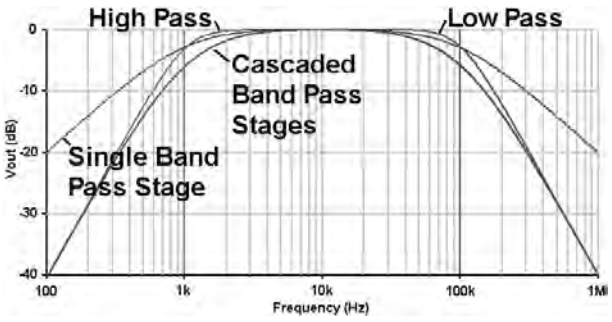


Figure 6.9. Bode diagrams for cascade bandpass vs. cascade low / high pass stages.

filter have a better performance in rejecting input noise from frequencies outside the specified frequency band.

Another important aspect to consider is the type of response require for the design. There are three popular filter responses, these are:

- Butterworth

This response, also know as a normally flat approximation, achieve a response of order n, without any ripple.

- Chebyshev

  This response offers a superior attenuation in the stop band at expense of a gain ripple in the passband. Generally a ripple of .1 dB to .3 dB is used.

- Bessel

  This response offers a flat group delay which is useful in digital design applications since it pass square signals with minimum harmonic distortion.

Figure 6.10 shows the graphical behavior of the three responses discussed above. The bandpass filter is designed using a butterworth response since it have a smooth transition from the pass to the stop band.

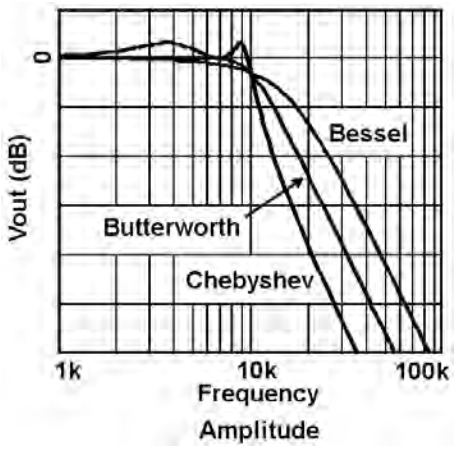


Figure 6.10. Bode diagrams Butterworth, Chebyshev, and Bessel responses.

The last consideration was to implement the circuit using a multiple feedback (MFB) configuration. MFB filters have lower sensitivities to external components when compare to their counterpart the Sallen-Key filter. This is useful since the variation of the cutoff frequency is minimized due the fact that they do not depend of external

components too much.

The schematic of the filter implement is shown in Figure 6.11.

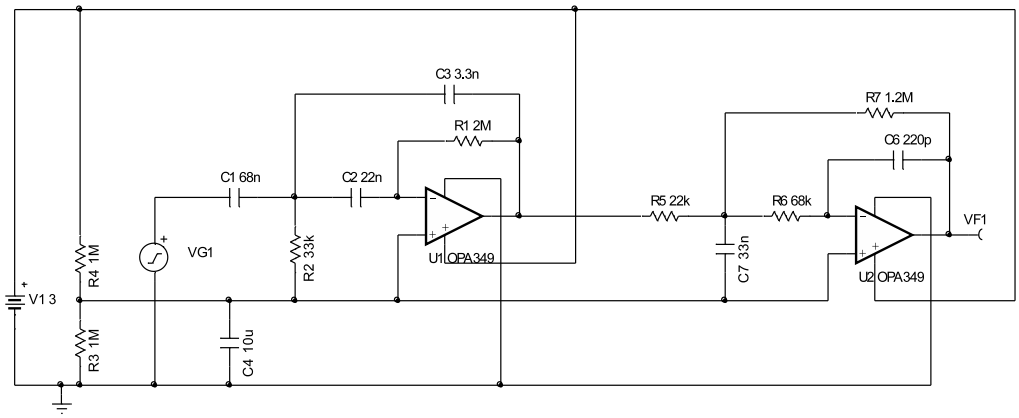
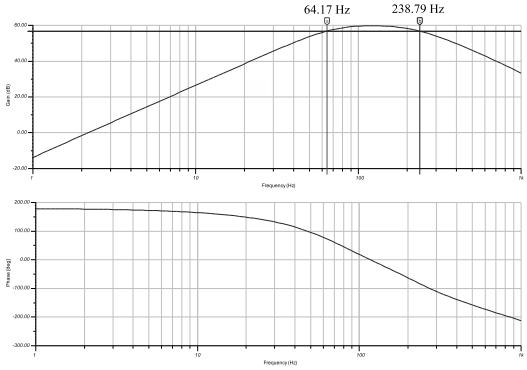


Figure 6.11. Schematic of the Multiple Feedback filter.

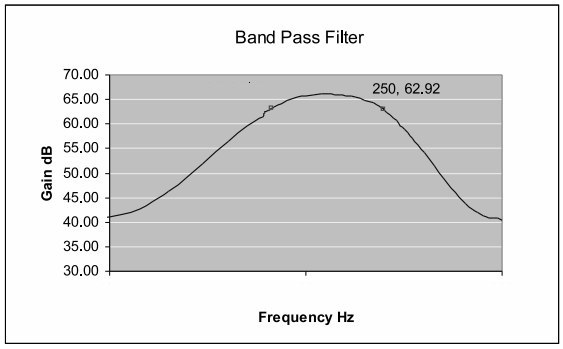
For the filter the OPA349 was used since it consumes only 1 μA of Icc, which is the OpAmp with lowest quiescent current consumption available in the market at present time. The first stage of the schematic show the MFB high pass filter. This first stage have a 20 V/V gain and a cutoff frequency near 70 Hz. The second stage is a MFB low pass filter with a gain of 50 V/V and a cutoff frequency of 200 Hz. the total gain will be A_{total} = 20 x 50 = 1000. The simulated output is shown in Figure 6.12(a) and the measured response is shown in Figure 6.12(b). The measured response closely resemble the simulated AC sweep in terms of Gain and cutoff frequencies. The measured low frequency was 67 Hz and the high one was 250 Hz.

6.2.3 Level Detector

Two configurations can be used to implement the level detector, the first one use a variable gain amplifier in the filter or difference amplifier and the second use a



(a) Simulated AC sweep for the MFB butterworth filter



(b) Measured AC sweep for the MFB butterworth filter

Figure 6.12. AC sweep for the MFB butterworth filter.

variable threshold for the level detector. The second approach is more suitable for this research since the voltage reference can be generated using Digital to Analog Converters (DACs) that can be implemented by the msp430. Figure 6.13 shows the DAC operation unsign TimerA to generate pulse-width modulated signals. The loading circuit for the DAC to be able to produce a DC voltage is a RC circuit with R = 1MΩ and C = 1 μF.

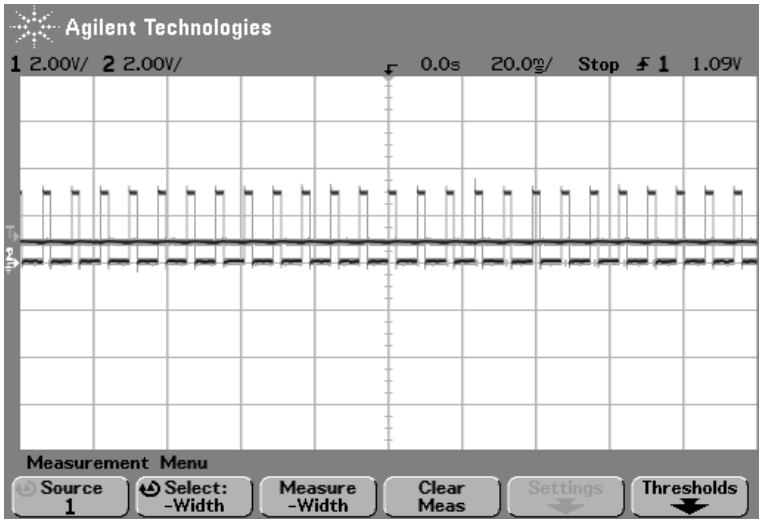


Figure 6.13. Measurement of the Internal DAC implemented by TimerA.

These DACs were validated in three different units of the msp430f1611 measuring the output voltage for a set of programmable bits. Table 6.2 shows a sample of one of the runs used to validated the programmability of the DAC. As shown the programmable range start from 33 mV going up to 951 mV.

Table 6.2. DAC's programmable voltage levels.

Run	Voltages	bit number		Voltages	bit number
Run 1	33 mV	3		558 mV	51
	98 mV	9		623 mV	57
	164 mV	15		689 mV	63
	230 mV	21		755 mV	69
	295 mV	27		820 mV	75
	361 mV	33		886 mV	81
	426 mV	39		951 mV	87
	492 mV	45			

This DAC is used in conjunction with the TLV349 micropower comparator to construct the level detector. The comparator requirements are not that difficult to achieve in terms of propagation delay and slew rate. However, the supply current should be as small as possible. TLV349 consumes an ultra low current of 1.2 μA which is an exceptional device to performed the comparison task.

6.3 Output stage design

The basic function of the output stage is to stimulate the heart with an adequate pulse. Since the adequate pulse varies in different patients as well with time, it is necessary that the pulse generator have the ability to produced programmable stimuli. The block diagram for the output stage is presented in Figure 6.14.

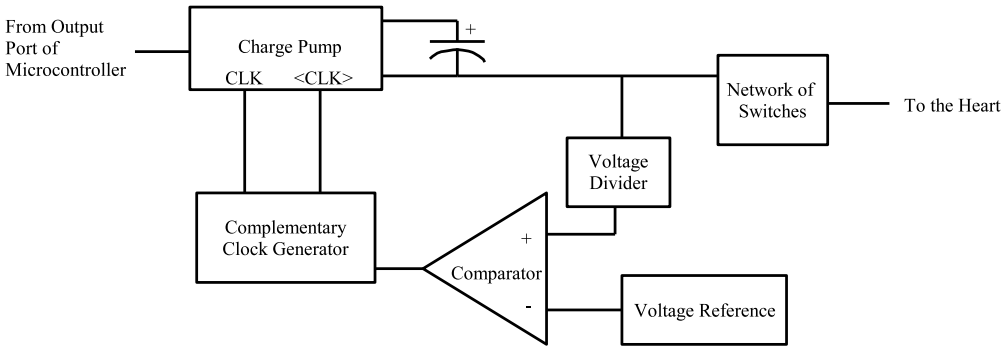


Figure 6.14. Block diagram of the output stage.

The output stage use a diode array to implement the charge pump since the constrain in terms of area is more relaxed for PCBs than for ICs. Schottky diodes are used to implement the charge pump because they have lower forward biased voltage drops when compare to their silicon counter parts. Figure 6.15 shows the schematic of the charge pump with the stimulation switches.

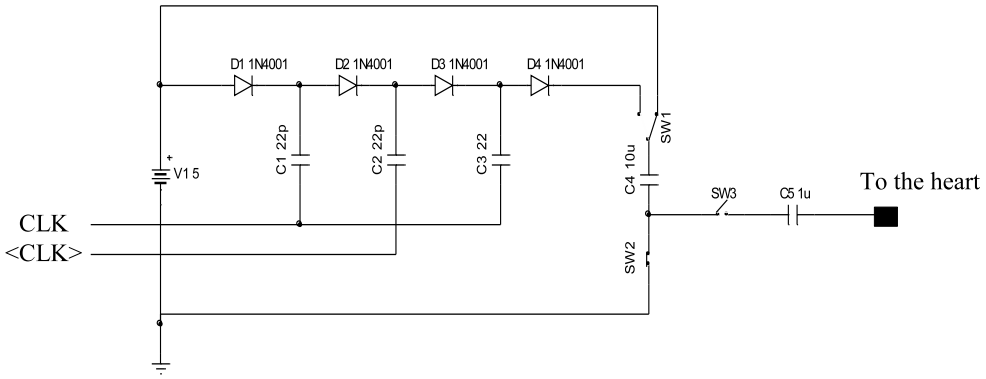


Figure 6.15. Schematic of the charge pump.

The operation of the output stage is straight forward, the charge pump charge the capacitor until its stored voltage pass the threshold, T_{high}, set by the DAC and the resistor network that creates a hysteresis. When this occurs the voltage comparator output a zero which turn off the clock generator. When the clock generator is turn off

the output voltage will start to decrease slowly until T_{low} is reach and the comparator turn on the clock generator again. The clock generation is done using an oscillator driver (sn74lvc1404) with a 2 MHz crystal oscillator. This driver generates the clock and complementary clock signals require for the charge pump operation. Figure 6.16 shows the start up time which is 4 s to a 99% of the signal. As shown in the figure the output voltage achieve is 6.5 V from a 2.8 Vbat.

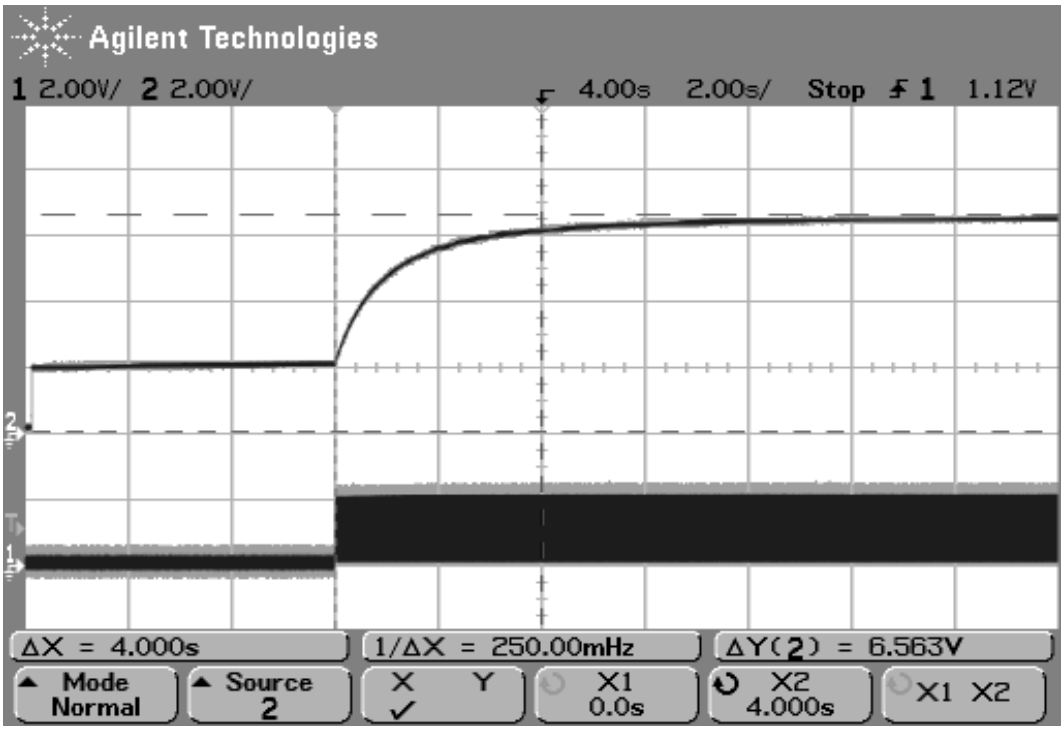


Figure 6.16. Measurement of the charge pump start up time.

The hardware was intended for low power operation. All the ICs used were carefully evaluate to choose the ones with lower power consumption. The current consumption of the software should be around 10 μA to give a 10 μA headroom for hardware consumption. Next section discuss throughly the software verification for PROV910's functionalities.

6.4 Software verification

This section presents many oscilloscopes plots. As was mention before, the oscilloscope used for data acquisition is Agilent 54622D. Since it is a mixed-signal oscilloscope the digital interrupts were measure in conjunction with analog signals. The plots are divided in the following sections:

Measured Timing diagrams

1. Basic Pace timing diagram
2. Pulse Width timing diagram
3. Refractory timing diagram
4. EGM timing diagram
5. Port1 timing diagram
6. Port2 timing diagram
7. Watchdog timing diagram

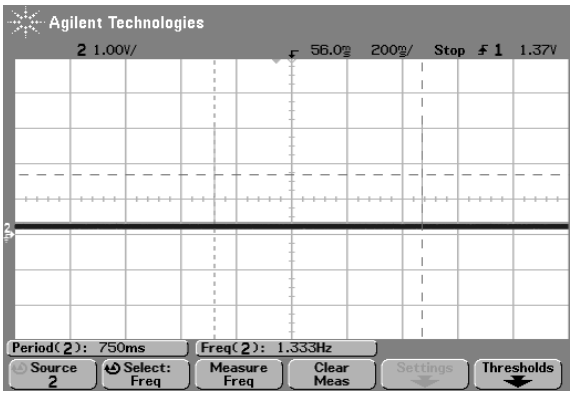
and functionality plots

1. VVI mode
2. VVT mode
3. VOO mode
4. Maximum frequency for VVI and VVT

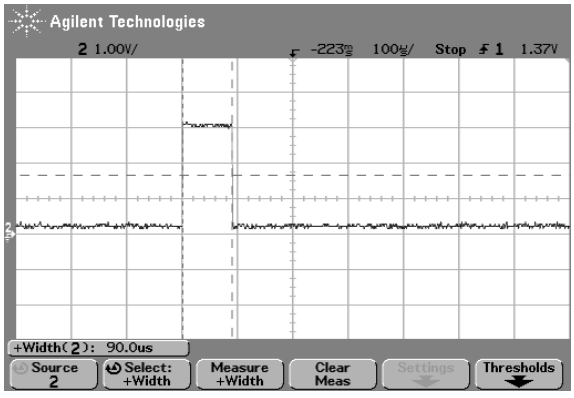
6.4.1 Measured timing diagrams

As was discussed in Chapter 5 the software was designed using the eternal loop concept. There, expected timing diagrams were presented for each of the interrupts. This subsection validates the expected timing diagrams presented in Chapter 5. All the timing diagrams, with a zoom in, are presented for each of the interrupts. no explanation will be given since they where discussed in Chapter 5.

Basic Pace timing diagram



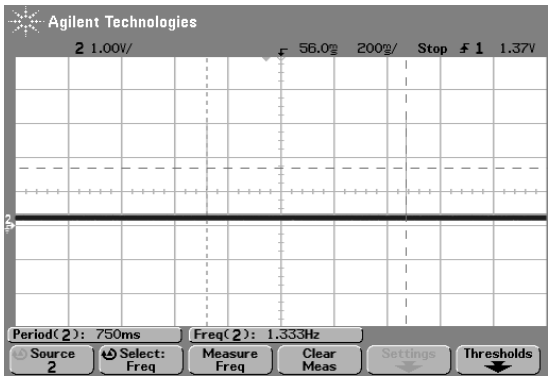
(a) Timing diagram with measured Basic Pace interval



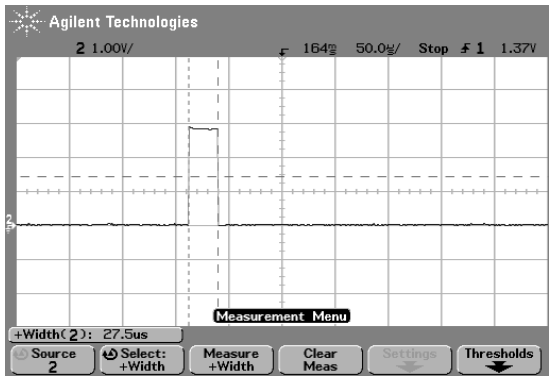
(b) Zoom in of the Basic Pace time diagram

Figure 6.17. Timing diagram of the Basic Pace Interval.

Pulse Width timing diagram



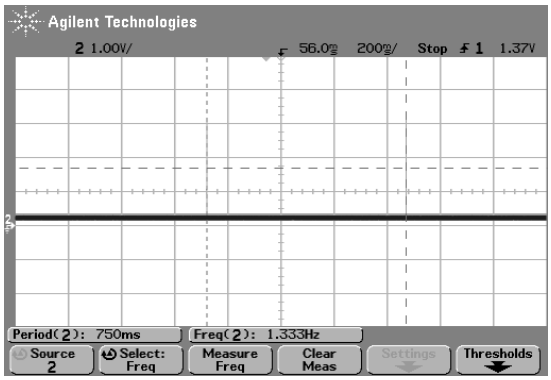
(a) Timing diagram for Basic Pace



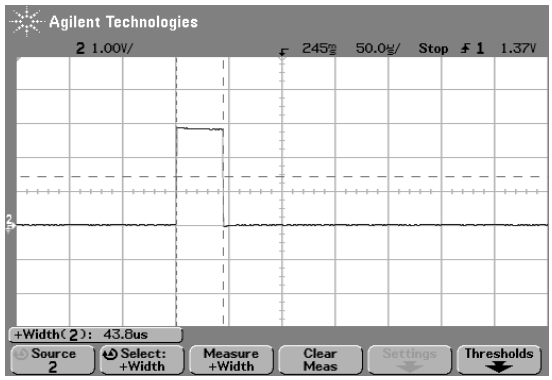
(b) Zoom of the Pulse Width time diagram

Figure 6.18. Timing diagram for the Pulse Width interval.

Refractory Period timing diagram



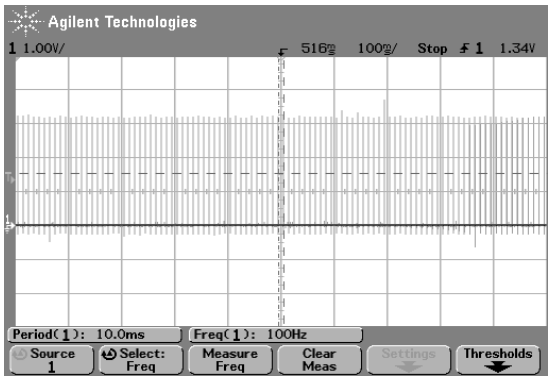
(a) Timing diagram for the Basic Pace interval



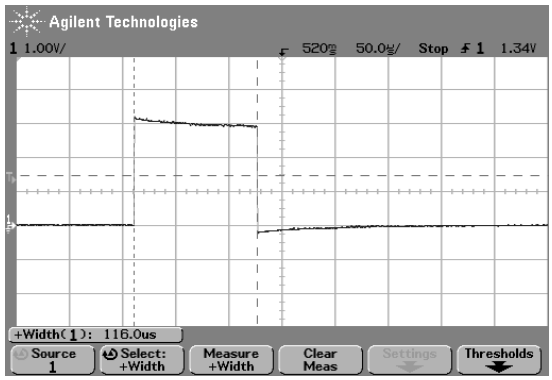
(b) Zoom of the Refractory Period time diagram

Figure 6.19. Timing diagram for the Refractory interval.

Electrogram timing diagram



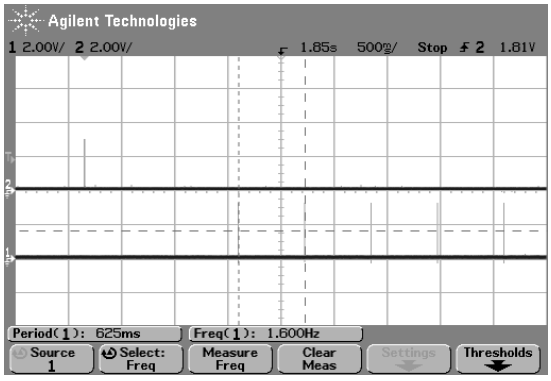
(a) Timing diagram for the EGM interval



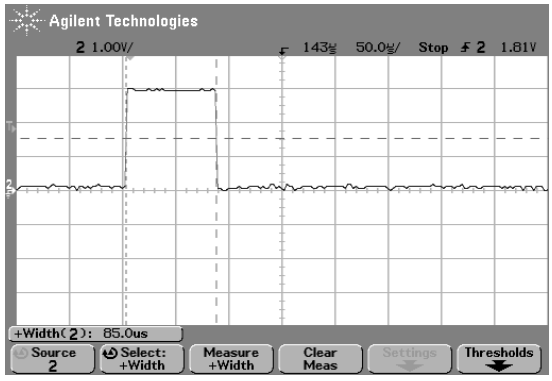
(b) Zoom of the EGM time diagram

Figure 6.20. Timing diagram for the EGM interval.

Port1 timing diagram



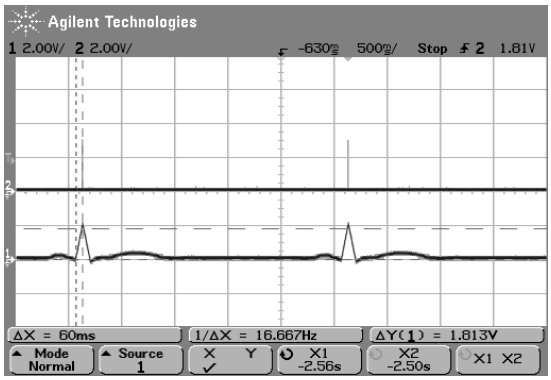
(a) Timing diagram for the Port1 interrupt



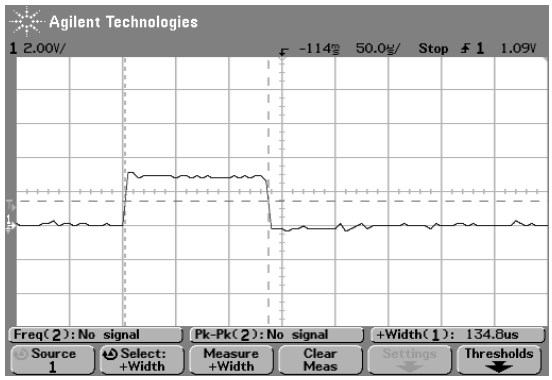
(b) Zoom of the Port1 time diagram

Figure 6.21. Timing diagram for the Port1 interrupt.

Port2 timing diagram Watchdog timing diagram

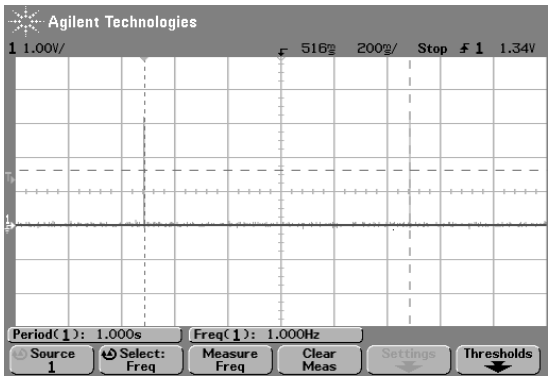


(a) Timing diagram for the Port2 interrupt

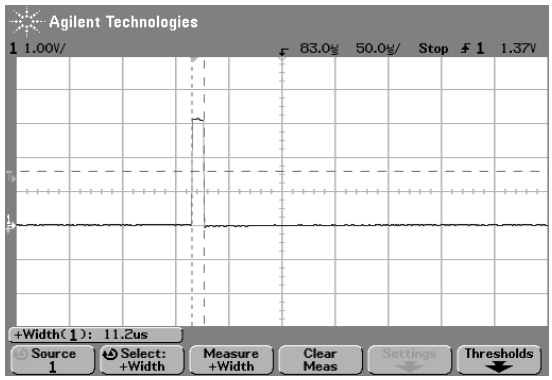


(b) Zoom of the Port2 time diagram

Figure 6.22. Timing diagram for the Port2 interrupt.



(a) Timing diagram for the Watchdog interval



(b) Zoom of the Watchdog time diagram

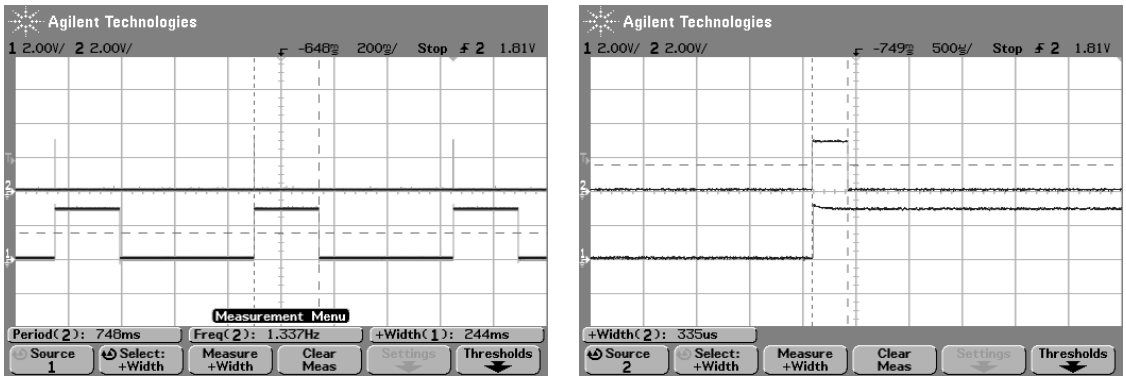
Figure 6.23. Timing diagram for the Watchdog interval.

6.4.2 Software functionality plots

This section will present the PROV910 working in its three different operational modes. When presenting the VVI and VVT modes in some cases a heart signal is display. The heart signal was created using Agilent 33120 15 Mhz Arbitrary Waveform Generator and Agilent intuilink arbitrary waveform editor. This is only to better distinguished between the signals display in the plots, since in reality the input port 2 receive a pulse generated by the comparator and not the actual heart signal.

VVI mode

Figure 6.24 presents the behavior of PROV910 in VVI mode without any external stimulus. Since no signal is applied, the behavior of the VVI is very much similar to the VOO. However, in the VVI mode the comparator remains working while in the VOO it is completely turn off. The zoom in Figure 6.24(b) shows that the refractory period start before sending the output pulse which is the proper operation for the pacemaker.

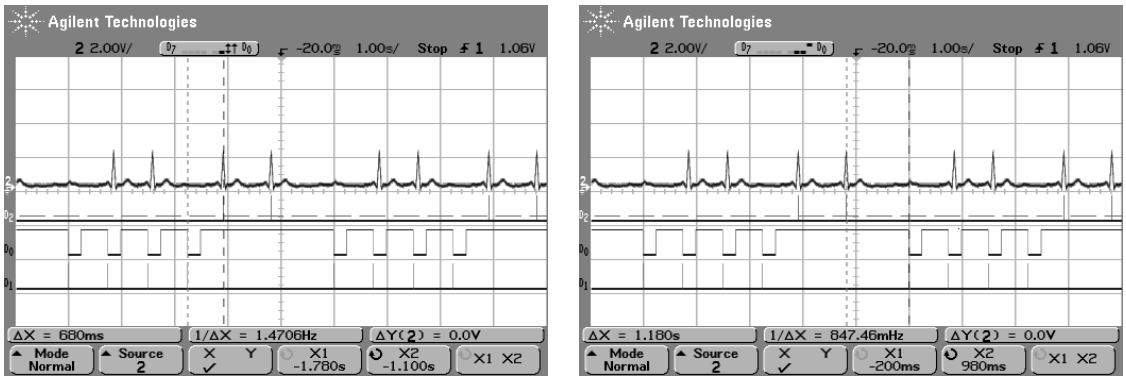


(a) VVI mode without external stimulus (b) Zoom to VVI mode without external stimulus

Figure 6.24. VVI mode operation without external stimulus sensed.

When a signal is applied to port 2.0 the PROV910's stimulus is inhibited. Figure 6.25 shows the behavior of the pacemaker while receiving signals from the heart. Figure

6.25(a) mark how the heart signal inhibit the pulse because it fall inside the escape interval. Figure 6.25(b) illustrates the functionality of hysteresis. Since hysteresis provoke a longer escape interval, a second spontaneous QRS fell inside the escape interval.



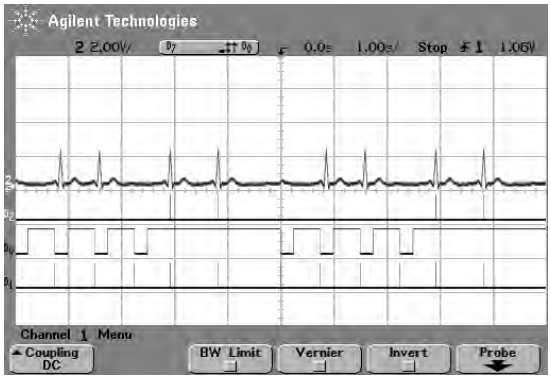
(a) VVI mode with external stimulus, Escape Interval

(b) Zoom to VVI mode with external stimulus, Hysteresis

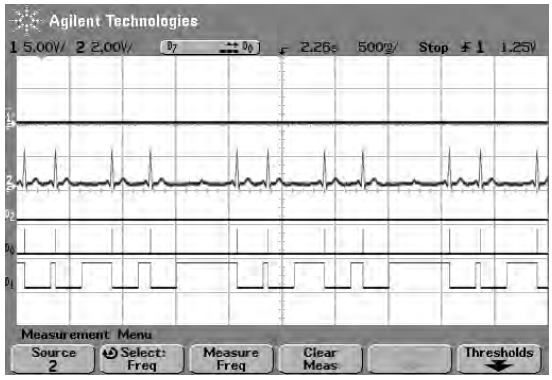
Figure 6.25. VVI mode operation with external stimulus sensed.

VVT mode

In the VVI mode a pulse is generated every time a stimulus is sensed. Figure 6.26 shows the behavior of the trigger mode. Note that during each interrupt of port 2 a pulse is generated. Figure 6.26(b) shows the comparator turning on and off after each pulse. This is important to avoid the sensing of the pulse generate by the pacemaker itself.



(a) VVT mode with external stimulus, Port2 interrupts



(b) VVT mode with external stimulus, comparator

Figure 6.26. VVT mode operation with external stimulus sensed.

VOO mode

The VOO mode was presented when discussing the time diagrams for Port1. However it is presented here to get a better understanding of its functionality.

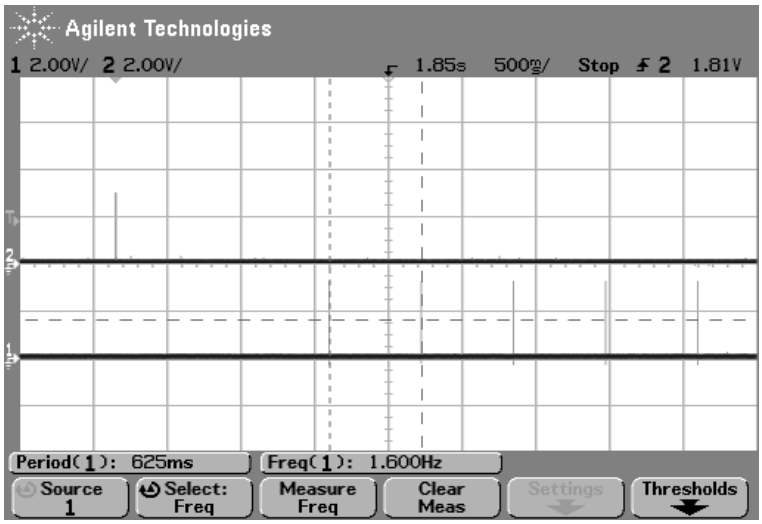


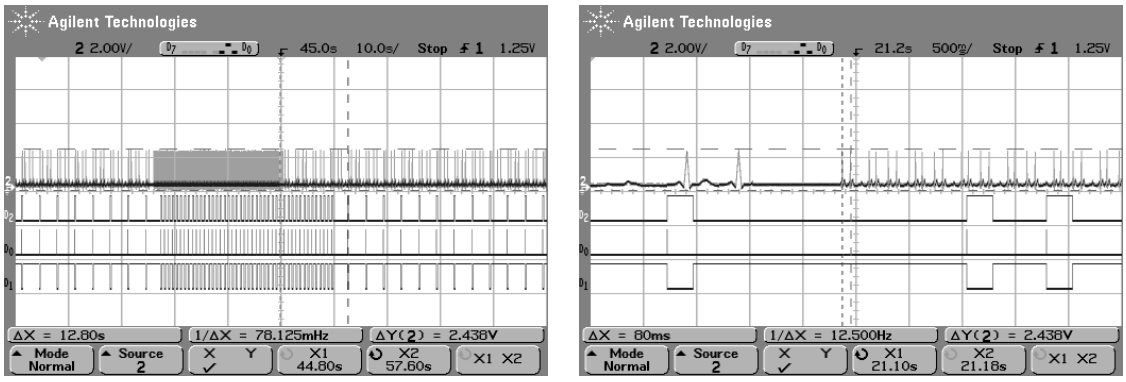
Figure 6.27. VOO mode, asynchronous pacing.

Basically the VOO is only activated by an external magnet applied to the pacemaker. When this occurs all the interrupts are disable and the comparator is turn off. VOO

is used by doctors to treated some heart conditions, like when arrhythmias occurs. Note that for this case the rate of the VOO indicates V_{batt} is fine.

Maximum frequency conditions for VVI

For safety when a frequency larger than 11 Hz occurs, the PROV910 enters in VOO mode. This behavior is illustrated in Figure 6.28. As shown in Figure 6.28(b) the frequency event is of 12.5 Hz which triggers the safety mechanism. The normal pace is restored in the next battery interval. This parameter is programmed by the user and have a default value of 1 min.



(a) VVI mode with high frequency condition (b) Zoom to VVI mode with high frequency condition

Figure 6.28. VVI mode with high frequency condition.

These plots demonstrate the functionality of PROV910 pacing system. The typical current consumption of the software is 5 μA with a maximum of 8 μA, which gives a headroom of 12 muA for the hardware implementation. The goal of the research was achieved, which are: 1) low power consumption of the software using a general purpose microcontroller and 2) system functionality of the pacemaker application.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

A complete methodology for the design of a cardiac pacemaker has been presented. Furthermore, a control software source code that was validated in a pacemaker prototype has been developed. The software has a typical current consumption of 5 μA with a maximum of 8 μA. This is a significant achievement since based of the information gathered in Table 2.2 a pacemaker consumed around 20 μA to generated a stimulus with an average current of 2 μA. Taking into account the front end and output stage architectures proposed in open literature [40, 41, 42, 43, 44, 45], the external interface hardware can be implemented with a current consumption of 2 μA. This give a I_{total} of 9 μA including the current consumed by the stimulus which compares with the current consumption of commercial pacemakers.

Extensive background information based on the available open technical literature in cardiac pacemakers was presented. This literature was the foundation for the current research and its organization by categories can be useful as reference for beginners in the matter of pacemaker design. Insights about current trends in pacemaker design were discussed, focused in hardware implementations and software algorithms.

A detailed discussion of microcontrollers with emphasis on important aspects for low power consumption was offered. This discussion includes aspects like memory organization, CPU architectural configurations, and instruction set philosophies. The definition of a pacemaker as embedded system and the importance of a microcontroller as the central control unit were also established.

The methodology for the development of a pacemaker system was presented. A definition of the requirements for a pacing system was show and an extensive analysis of available low power microcontrollers was given, comparing them in terms of average current, low power modes, clock systems and leakage currents. A power profile of the msp430's instruction set was developed using an average current testbench. The base energy cost, the inter-instruction cost, and the operand cost was tabulated to give a better reference for the software designer. The results in these tables further proved the significance of designing low power software. Up to a 20% power reduction can be achieved by using the software design guidelines outlined in this research.

A fully explained discussion of pacemaker's software design, including hardware partitioning, considerations for software specifications, and a comprehensive set of software flowcharts were given. The software was designed in conjunction with the hardware to get better comprehensive system that minimizes the current consumption of the whole system.

Finally an external hardware design for the front end and output stage were offered. A block diagram at the system level was included which illustrates the configuration of each of them. A detail description of each part of the block diagram points out the desirable characteristics and guide the designer to the key parameters that are essential for hardware implementation. The next subsection presents a list of the contributions done in this research.

7.2 Contribution of this work

The main contributions of this work can be outlined as follows:

- The generation of an open-source pacemaker control code using assembly language for a general purpose microcontroller.
- A software design that achieve a typical current consumption of 5 μA
- An open-source design methodology for a cardiac pacemaker was presented. This methodology develops the software in conjunction with the hardware achieving good results and proposed a set of techniques for the comparison of microcontrollers in terms of low power consumption.
- A source code was developed that illustrates the functionalities of a cardiac pacemaker for educational purposes including flowcharts and timing diagrams.
- The instruction set power characterization of the msp430 microcontroller which can help programmers to develop software that achieve ultra low power consumption. A set of guidelines was created using the experimental data of the instruction set power characterization. These guidelines help to generate efficient low power embedded software code for the msp430.

7.3 Future work

Several research topics can be derived from this work as future work. These can be summarized as follows:

- A detailed power characterization of the msp430's instruction set can be performed using statistical analysis. Statistical design of experiments (DOE) can

be used to block undesirable variables and isolate the variables of interests. The variables of interest would be the msp430's core instructions, their interaction, and the operand contents of the instructions. A well designed experiment can be implemented to minimize the sample size to represent a 95% of the population.

- The development of an educational laboratory kit that demonstrates the functionality of the different operation modes of a pacemaker and how to test them using off-the-shelf equipment like oscilloscopes, multimeters and function generators. This kit would be useful to introduce students to the pacemaker concept and to understand special requirements within this growing industry.

- The design of an interface integrated circuit that connects the microcontroller with the heart. This IC can be design using the block diagrams and the references presented in this research. The development of this IC should be done to consumed at most 3 μA of average current. The interface IC can be implemented with the msp430 using the software developed in this work to observed the performance of the system as a whole. Contributions to the research in terms of materials (batteries, leads, etc.) of leading companies in the pacemaker business should be considered.

APPENDICES

APPENDIX A

PROV910 pacemaker Source Code

```
;*****************************************************************************************************
;   Standard Format
;
;   Source name             : Main_Basic.s43
;   Executable name         : Basic_LP_Algorithm.exe
;   Code model              : 1.0
;   Author                  : Sigfredo E. González Díaz
;   Description             : PROV910 Main Control.
;   Harware Used            : SVS, TimerA, TimerB, WD_Timer
;
;*****************************************************************************************************
#include  <msp430x16x.h>

;-----------------------------------------------------------------------------------------------------
                ORG 1100h     ; RAM Memory definition for Variables
;-----------------------------------------------------------------------------------------------------

BP_Const        DW  0000      ; Constants to be used by program, RAM Location = 1100h = 4352
PW_Const        DW  0000      ; 4354
Rate_Count      DW  0000      ; 4356
Hyst_Const      DW  0000      ; 4358
Magnet_Const    DW  0000      ; 4360
WDT_Const       DW  0000      ; 4362
EGM_Const       DW  0000      ; 4364
BatteryStatus   DW  0000      ; 4366
tMeas           DW  0000      ; 4368
ElectrodeStatus DW  0000      ; 4370
Prev_Freq       DW  0000      ; 4372
Freq_Count      DW  0000      ; 4374
Prev_Rate       DW  0000      ; 4376
VVT_Rate        DW  0000      ; 4378
Max_VVT         DW  0000      ; 4380
Hist_Paced      DW  0000      ; 4382
Hist_Sensed     DW  0000      ; 4384
Hist_EGM        DW  0000      ; 4386
Hist_VOO        DW  0000      ; 4388
Hist_Electrode  DW  0000      ; 4390
```

```
Hist_Battery    DW  0000      ; 4392
EGMCurrent      DW  0000      ; 4394
EGMstart        DW  0000      ; 4396, First address to store EGM #1

;-------------------------------------------------------------------------------------------------------
            ORG     4000h                           ; Define the address in which the program
;-------------------------------------------------------------------------------------------------------
RESET       mov.w   #3900h,SP                       ; Initialize '1611 stackpointer to the top of RAM
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; Stop WDT
StopXT2     bis.b   #BIT7,&BCSCTL1                  ; Stop XT2

; Module B - Default Values Setup ----------------------------------------------------------------------
;*******************************************************************************************************
            REPTI reg,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15 /* Clear all registers using Repeat */
            xor reg,reg                                     /* - instruction and "exclusive or" */
            ENDR                                            /* - to consume 1 cycle per clear   */

            mov.w   #7098h,R4                       ; Set the Default Pacemaker parameters (#7098)
                                                    ; - Basic Pacing = 80 bpm (24)
                                                    ; - Pulse Amplitude = 3V (3)
                                                    ; - Sensitivity = 492 mV (7)
                                                    ; - Pacing Mode = VVI (0) (#7098h), VVT (#7498h)
            mov.w   #0004h,R5                       ; Set the Default Pacemaker parameters (#0004)
                                                    ; - Pulse Width = .4 ms (4)
                                                    ; - EGM rate = 400 Hz (0)
                                                    ; - VVT upper rate = 80 bpm (0)
                                                    ; - Sensing Polarity = Bipolar (0)
                                                    ; - Pacing Polarity = Bipolar (0)
                                                    ; - New Parameter? = No (0)
            mov.w   #08143h,R6                      ; Set the Default Pacemaker parameters (#8143)
                                                    ; - Refractory = 245 ms (3)
                                                    ; - Hysteresis = 142 bpm (10)
                                                    ; - WDT interval = 1 min (0)
                                                    ; - EGM enable? = yes (1) (#A143h)
                                                    ; - Electrode Status? 0(good) 1(bad)
                                                    ; - Hysteresis on? = Yes (1) off (0)=>(2143h)
            mov.w   #0,BatteryStatus                ; Set the Default Battery Status, 2.9V

; Module C - Battery Check -----------------------------------------------------------------------------
;*******************************************************************************************************
;   Description             : PROV910 Battery Status Monitoring.
;
;   SVS:
;   Variable/s used         : BatteryStatus, Hist_Battery
;   Action                  : Use the MSP430 Supply Voltage Supervisor to evaluate
;                             if the Battery Charge is at critical level
;
;   Output Location         : Memory labeled BatteryStatus, RAM = 4366
;                             Memory labeled Hist_Battery, RAM = 4390
;*******************************************************************************************************
;-------------------------------------------------------------------------------------------------------

SVS             push    R11                 ; Stored R11 on stack and initialize it
                push    R14                 ; Stored R14 on stack and initialize it
```

```
                inc.w   Hist_Battery        ; Histogram, counts frequency of Battery check
                mov.w   BatteryStatus,R11   ; Move Battery Status to R11
                mov.b   #41h,R14            ; Set 65 loops to wait for SVS to turn on and stabilize
Setup_SVS_On    mov.b   SVS_9(R11),&SVSCTL  ; Turn on SVS, set threshold to previous battery status
Set_loop        dec.b   R14                 ; Decrease R14, each decrease takes ~1us for DCO = 800kHz
                jnz     Set_loop
                bit.b   #BIT0,&SVSCTL       ; Test if AVcc = V_bs (example if V_bs >= 2.9V)
                jz      End_SVS             ; jump to end of routine
                add.w   #1,BatteryStatus    ; If AVcc <=  V_bs, then set V_bs = V_bs + 1
                                            ; (example, next pass V_bs = 2.8V)

End_SVS         clr.b   &SVSCTL             ; Turn off SVS
                pop     R14                 ; Restored R14
                pop     R11                 ; Restored R11

; Module D - Electrode Check ---------------------------------------------------------------------------
;*************************************************************************************************
;   Description             : PROV910 Electrode Status Monitoring.
;
;  Electrode:
;   Variable/s used         : ElectrodeStatus, Hist_Electrode, tMeas
;   Action                  : Use the MSP430 Comparator A to evaluate
;                             if the Electrode Impedance is at critical level
;
;   Output                  : Electrode Flag in Register R3, BIT E
;   Output Location         : Memory labeled ElectrodeStatus, RAM = 4370
;                           : Memory labeled Hist_Electrode, RAM = 4388
;                           : Memory labeled Hist_Electrode, RAM = 4368
;*************************************************************************************************
;-------------------------------------------------------------------------------------------------

Electrode       push    R11                                 ; Stored R11 on stack and initialize it
                push    R12                                 ; Stored R12 on stack and initialize it

Setup_CompA     mov.b   #CARSEL+CAREF_1+CAIES,&CACTL1       ; Set Vref = .25Vcc applied to - terminal
                                                            ; Enable Interrupts and negative trigger
                mov.b   #P2CA0+CAF,&CACTL2                  ; Connects CA0 to P2.3 and Filters output
                inc.w   Hist_Electrode                      ; Histogram of Electrode measurement
Setup_Port2     bic.b   #CAPD2,&CAPD                        ; Enable Port 2.2

MeasureRmeas    bic.b   #CAPD4,&CAPD                        ; Enable Port 2.2
                bis.b   #BIT4,&P2DIR                        ; Set P2.4 as output
                bic.b   #BIT2+BIT3,&P2DIR                   ; Set P2.2 & P2.2 as inputs
                bis.b   #BIT4,&P2OUT                        ; Charge to 7*tau = 7*33n*500 = 10.6us, MCLK = 800kHz
                mov.w   #50,R11                             ; Set counter to 40 us to charge Cap
Charge_delay2   dec.w   R11                                 ;
                jnz     Charge_delay2                       ;
                bis.b   #CAON,&CACTL1                       ; Turn on comparator A, CompOut = 1
Disch_Rmeas     bic.b   #BIT4,&P2OUT                        ; Discharge Cap, when Vcap < .25Vcc CompOut = 0
wait_Disch2     inc.w   R11                                 ; 2 cycles, Each step = 1.25us for MCLK = 800kHz
                bit.b   #BIT0,&CACTL2                       ; 7 cycles, Check if CompOut = 0
                jnz     wait_Disch2                         ;
                mov.w   R11,tMeas                           ; Move counter value to RAM memory
                bic.b   #CAON,&CACTL1                       ; Turn on comparator A, CompOut = 1
                cmp.w   #3,R11                              ;
```

```
                jne     Lead_Fracture                   ;
                mov.w   #3333h,ElectrodeStatus          ; 3333 code for good lead
                bic.w   #BIT9,R5                        ; Reset Electrode bit to indicate good impedance
                jmp     End_Electrode                   ;
Lead_Fracture   cmp.w   #8,R11                          ;
                jlo     Isul_Defect                     ;
                mov.w   #4444h,ElectrodeStatus          ; 4444 code for Lead Fracture
                bis.w   #BITE,R6                        ;
                jmp     End_Electrode                   ;
Isul_Defect     mov.w   #2222h,ElectrodeStatus          ; 2222 code for Isulation Defect
                bis.w   #BITE,R6                        ;
End_Electrode   pop     R12                             ; Restored R12
                pop     R11                             ; Restored R11

;-------------------------------------------------------------------------------------------------------
;*******************************************************************************************************
;   Description             : Set Time parameters of the Pacemaker
;
;   Time_Manager
;   Register/s used         : R4, R5, and R6.
;   Variable/s used         : Basic Pacing, Pulse Width, Refractory, Sensitivity, and Pulse Amplitude
;
;
;   Action                  : Use the MSP430 TimerB to generate interrupts based on programmed
;                           : - pacemake's parameters and use TimerA as a DAC to generate DC voltages.
;
;   Output          :
;   Output Location : TB0IV Location = FFFAh and TB1IV Location = FFF8h
;
;*******************************************************************************************************
;-------------------------------------------------------------------------------------------------------

SetupWDT        mov.w   #WDT_ADLY_1000,&WDTCTL          ; Setup WDT to Interval mode, ACLK, interval ~ 1s
                bis.b   #WDTIE,&IE1                     ; Enable Watchdog interrupt

SetupP1         clr.b   &P1OUT                          ; Clears Output Port 1
                clr.b   &P1DIR                          ; Sets all P1.x to inputs
                clr.b   &P1IES                          ; Sets all interrupts to rising edge
                bis.b   #00Ch,&P1SEL                    ; Select TA2, TA3,
                bis.b   #06Ch,&P1DIR                    ; Sets P1.1, 2, 5, and 6 as outputs
                bis.b   #BIT4,&P1IE                     ; Enable Interrupts for P1.4

SetupP2         clr.b   &P2OUT                          ; Clears Output of Port2
                bis.b   #BIT3,&P2SEL                    ; Selec Comparator Input CA0
                bic.b   #BIT0,&P2IES                    ; Set Interrupt at rising edge for P2.0
                bic.b   #BIT3+BIT2+BIT0,&P2DIR          ; Set P2.0, 2, and 3 as input
                bis.b   #BIT1,&P2DIR                    ; Set direction of Port2.1 to output
                bis.b   #BIT0,&P2IE                     ; Enable Interrupts for P2.0 to receive
                                                        ; - signal from comparator

SetupP3         bic.b   #BIT0,&P3OUT                    ; Reset Output Pin P3.0
                bis.b   #BIT0+BIT1,&P3DIR               ; Set P3.0 into output direction

SetupP4         clr.b   &P4DIR                          ; Set all P4.x to output
                clr.b   &P4OUT                          ; Set all P4.x to zero
```

```
                bis.b   #0FFh,&P4SEL                    ; Select TB0, TB1, TB2, TB3, TB4, TB5, TB6
                bis.b   #0EFh,&P4DIR                    ; and sets all as outputs except TBOUT

SetupP5         clr.b   &P5OUT                          ; Set direction of P5 as outputs
                clr.b   &P5OUT                          ; Sets all outputs of P5 to zero

SetupP6         clr.b   &P6OUT                          ; Set direction of P6 as outputs
                clr.b   &P6OUT                          ; Sets all outputs of P6 to zero

Setup_Security  mov.w   Max_Freq,Prev_Freq              ; Initialize Previous Frequency
                mov.w   Max_Rate,Prev_Rate              ; Initialize Previous Rate

SetupTACCTLx    mov.w   #OUTMOD_7,&CCTL1                ; CCR1 toggle, interrupt enabled
                mov.w   #OUTMOD_7,&CCTL2                ; CCR2 toggle, interrupt enabled

SetupTA         mov.w   #TASSEL_1,&TACTL                ; ACLK, Up-mode
                mov.w   #255,&CCR0                      ; Sets Resolution of DAC

SetupTBCCTLx    mov.w   #OUTMOD_0+CCIE,&TBCCTL1         ; CCR1 Output Mode, interrupt enabled
                mov.w   #OUTMOD_0+CCIE,&TBCCTL2         ; CCR2 Output Mode, interrupt enabled
                mov.w   #OUTMOD_0+CCIE,&TBCCTL3         ; CCR3 Output Mode, interrupt enabled

SetupTB         mov.w   #TBSSEL_1+TBIE,&TBCTL           ; ACLK, Continuous-mode

Basic_Pace      mov.w   R4,R11                          ; Move the Register R4, that contains the Basic
                                                        ; - Pacing offset
                and.w   #0003Fh,R11                     ; Use mask to get only the bits that contains
                                                        ; - the Basic Pacing offset
                rla.w   R11                             ;
                mov.w   Basic_Pacing_0(R11),&TBCCR1     ; Set the Counter Register to Basic_Pacing_20
                mov.w   Basic_Pacing_0(R11),R7          ; R7 contains the Basic Pace Interval

Pulse_Width     mov.w   R5,R11                          ; Move the Register R5 to R11
                and.w   #01Fh,R11                       ; Use mask to get only the bits that contains
                rla.w   R11                             ; the Pulse Width offset
                mov.w   Pulse_Width_0(R11),&TBCCR2      ; Set the Pulse Width
                mov.w   Pulse_Width_0(R11),R8           ; Move Basic Pace + Pulse Width to R8

Refractory      mov.w   R6,R11                          ; Move the Register R6 to R11
                and.w   #0001Fh,R11                     ; Use mask to get only the bits that contains
                rla.w   R11                             ; - the Refractory offset
                mov.w   Refractory_0(R11),&TBCCR3       ; Set the Refractory Period
                mov.w   Refractory_0(R11),R9            ; Set R9 = Refractory Period

Hysteresis      mov.w   R6,R11                          ; Move R6 to R11 for logic manipulation
                and.w   #03E0h,R11                      ;
                mov.w   #00004,R14                      ;
loop_hyst       rra     R11                             ; Setup Register for Hysteresis Check
                dec.w   R14                             ;
                jnz     loop_hyst                       ;
                mov.w   Hysteresis_0(R11),R10           ; Store Hysteresis into R10

Sensitivity     mov.w   R4,R11                          ; Move R6 to R11 for logic manipulation
                and.w   #0F000h,R11                     ; Mask to obtain Sensitivity
                mov.w   #5,R14                          ;
```

```
loop_sense      rlc.w   R11                             ; Setup Register for Sensitivity Check
                dec.w   R14                             ;
                jnz     loop_sense                      ;
                mov.w   Sensitivity_1(R11),&CCR1        ; Set DAC1 for Front End sensitivity

Pulse_Amp       mov.w   R4,R11                          ; Move R4 to R11 for logic manipulation
                and.w   #00FC0h,R11                     ; Mask to obtain Pulse Amplitude
                mov.w   #5,R14                          ;
loop_Amp        rra     R11                             ; Setup Register for Pulse Amplitude Check
                dec.w   R14                             ;
                jnz     loop_Amp                        ;
                mov.w   Pulse_Amp_1(R11),&CCR2          ; Set DAC2 for Output Pulse Amplitude Comparison

VVT_Upper_Rate  mov.w   R5,R11                          ; Move the Register R5, that contains
                                                        ; -  the Upper rate offset
                and.w   #1F00h,R11                      ; Use mask to get only the bits that contains
                mov.w   #7,R14                          ; - the Upper Rate offset
setup_Upper     rra     R11                             ; Only rotate 7 times to multiply by 2 to
                dec.w   R14                             ; - address words instead of bytes
                jnz     setup_Upper                     ;
                mov.w   Upper_Rate_0(R11),Max_VVT       ; Store upper VVT rate in variable

Watchdog        mov.w   R6,R11                          ; Move R6 to R11 for logic manipulation
                and.w   #00C00h,R11                     ; Mask to obtain Watchdog interval
                mov.w   #9,R14                          ;
loop_WDT        rra     R11                             ; Setup Register for Watchdog Interval Check
                dec.w   R14                             ;
                jnz     loop_WDT                        ;
                mov.w   WDT_Interval_0(R11),R13         ; Store Interval in R13
                mov.w   WDT_Interval_0(R11),WDT_Const   ; Store Interval in R13

RC_Timer        bic.b   #BIT5,&P1OUT                    ; Set P1.5 low to discharge when needed
                bic.b   #BIT5,&P1DIR                    ; Set as input to charge capacitor

Pacing_Polar    bit.w   #BITE,R5                        ; Test if Pacing Polarity is set
                jz      Sensing_Polar                   ; If not equal leave P5.0 output set to zero
                bis.b   #BIT0,&P5OUT                    ; Sets Pacing Polarity to Unipolar (1)
Sensing_Polar   bit.w   #BITD,R5                        ; Test if Sensing Polarity is set
                jz      Setup_EGM                       ; If not equal leave P5.1 to zero
                bis.b   #BIT1,&P5OUT                    ; Sets Sensing Polarityto Unipolar (1)

Setup_EGM       mov.w   Initial_EGM,EGMCurrent          ; Initialize Current EGM to EGM start address
                bit.w   #BITD,R6                        ; Check if EGM is required
                jz      No_EGM                          ; If not end interrupt
                mov.w   #CCIE,&TBCCTL4                  ; CCR4 interrupt enabled
                mov.w   R5,R11                          ; Move the Register R5, that contains the EGM rate offset
                and.w   #00E0h,R11                      ; Use mask to get only the bits that contains the EGM rate
                mov.w   #4,R12                          ;
EGM_Enable      rra     R11                             ; Only rotate 4 times to multiply by 2 to address words
                dec.w   R12                             ; instead of bytes
                jnz     EGM_Enable                      ;
                mov.w   ECG_RATE_0(R11),&TBCCR4         ; Set the Counter Register to ECG rate
                mov.w   ECG_RATE_0(R11),R15             ; Add Pulse Width to Basic Pacing
                inc.w   Hist_EGM                        ; Histogram of EGM requested
                jmp     TimerA_On
```

```
No_EGM          bic.w   #CCIE,&TBCCTL4                  ; CCR4 interrupt disabled

TimerA_On       bis.w   #MC_1,&TACTL                    ; Turn on TimerA in Up-mode
TimerB_On       bis.w   #MC_2,&TBCTL                    ; Turn on TimerB in Continuous-mode

Mainloop        bis.w   #LPM3+GIE,SR                    ; CPU off and Global Interrpt Enable
                nop                                     ; Required only for debugger

;-------------------------------------------------------------------------------------------------------
TBX_ISR;        Timer B Interrupt Handler
;-------------------------------------------------------------------------------------------------------

                add.w   &TBIV,PC                        ; Add Timer_B offset vector
                reti                                    ; CCR0 - no source
                jmp     TBCCR1_ISR                      ; TBCCR1
                jmp     TBCCR2_ISR                      ; TBCCR2
                jmp     TBCCR3_ISR                      ; TBCCR3
                jmp     TBCCR4_ISR                      ; TBCCR4
                jmp     TBCCR5_ISR                      ; TBCCR4
                reti                                    ; TBCCR6
                reti                                    ; Return from overflow ISR

TBCCR1_ISR      push    R11                             ; Stored R11 on stack and initialize it
                push    R12                             ; Stored R12 on stack and initialize it
                bit.w   #BITB,R4                        ; Test hight frequency.
                jnz     Basic_P                          ; If true no pulse
Reed_check      bit.b   #BIT4,&P1IN                     ; Check if Reed Switch is closed
                jz      Rate_Check
                add.w   Magnet_Const,&TBCCR1            ; Sets the pace depending on the Battery voltage
                bic.b   #BIT4,&P1IFG                    ; Clear the reed interrupt flag
                jmp     Set_Vout                        ; Jumps to release stimulus
Rate_Check      mov.w   &TBR,R11                        ; Move previous Current Time
                mov.w   Prev_Rate,R12                   ;
                mov.w   R11,Prev_Rate                   ; Store current time into Prev_Freq
                sub.w   R12,R11                         ; Substract Current time from previous freq sample
                cmp.w   Max_Rate,R11                    ; Check if Int generate before frequency Threshold
                jlo     TurnOn_RC                       ; Jump to end of routine this set mode to VOO
Basic_P         add.w   R7,&TBCCR1                      ; Move Basic Pacing time to TimerB1 Count Register
                bis.b   #BIT4,&P1IE                     ; Enable Reed switch interrupt
                bic.b   #BIT0,&P2IE                     ; Disable Interrupts from comparator
Set_Vout        bic.b   #BIT1,&P2OUT                    ; Turn off comparator
                bis.w   #BIT2,&TBCCTL2                  ; Set Output of Register 2, Set Pulse_Signal
                bis.w   #BIT2,&TBCCTL3                  ; Set Output of Register 3, Set Blank_Signal
                bis.b   #BIT1,&P5OUT                    ; Set Output P5.1, Set Sensing_Unipolar (Blank V+)
                inc.w   Hist_Paced                      ; Histogram of Paced Events
                jmp     End_BP
TurnOn_RC       inc.w   Rate_Count                      ; Increment high rate event

                bis.b   #BIT1,&P3DIR                    ; Set Output Direction
                bis.b   #BIT1,&P3OUT                    ; Set Output to Vcc
                bis.b   #BIT5,&P1IE                     ; Enable Interrupts for P1.5 for RC Timer
End_BP          pop     R12                             ; Restored R12
                pop     R11                             ; Restored R11
                jmp     TBX_ISR                         ; Jump to Interrupt Handler Label
```

```
TBCCR2_ISR      mov.w   &TBCCR1,&TBCCR2             ; Move Basic Pacing Time to Counter Register 2
                add.w   R8,&TBCCR2                  ; Add Pulse Width to Basic Pacing
                bic.w   #BIT2,&TBCCTL2              ; Reset Output of Register 2, Pulse_Signal
                jmp     TBX_ISR                     ; Jump to Interrupt Handler Label

TBCCR3_ISR      mov.w   &TBCCR2,&TBCCR3             ; Move Basic Pacing + Pulse Width to Register 3
                add.w   R9,&TBCCR3                  ; Add Refractory Period to Register 3
                bic.w   #BIT2,&TBCCTL3              ; Reset Output of Register 3, End Refractory Period
                bit.b   #BIT4,&P1IN                 ; Check if Reed Switch is closed
                jnz     CompOffMag                  ; If switch closed do not turn on Comparator
                bis.b   #BIT1,&P2OUT                ; Turn on comparator
                bit.w   #BITB,R4                    ; Test hight frequency.
                jnz     CompOffMag
                bis.b   #BIT0,&P2IE                 ; Enable Interrupts from comparator
                bic.b   #BIT0,&P2IFG                ; Clear P2.0 Interrupt Flag
CompOffMag      jmp     TBX_ISR                     ; Jump to Interrupt Handler Label

TBCCR4_ISR      add.w   R15,&TBCCR4                 ; Add EGM rate to TBCCR4, sets sampling frequency
                bis.b   #BIT0,&P6SEL                ; Enable A/D channel A0
                mov.w   #ADC12ON+SHT0_2,&ADC12CTL0  ; turn on ADC12, set samp time to ~3.2 us for ADCclk = 5MHz
                mov.w   #SHP,&ADC12CTL1             ; Use sampling timer
                mov.b   #SREF_2,&ADC12MCTL0         ; Vr+=VeREF+ (external)
                bis.w   #ENC,&ADC12CTL0             ; Enable conversions
                bis.w   #ADC12SC,&ADC12CTL0         ; Start conversions
testIFG         bit.w   #BIT0,&ADC12IFG             ; Conversion done?
                jz      testIFG                     ; No, test again
                mov.w   &ADC12MEM0,EGMstart(R10)    ; Move result to current EGM address
                add.w   #2,R10                      ; Increase EGM address pointer
                add.w   #2,EGMCurrent               ; Store address of last ECG store
                bic.w   #BIT0,&ADC12IFG             ; Reset ADC memory_0 Interrupt Flag
                bic.w   #ADC12ON,&ADC12CTL0         ; Turn off ADC12
                bic.b   #BIT0,&P6SEL                ; Disable A/D channel A0
                bis.b   #BIT0,&P6DIR                ; Set A0 to high impedance
                cmp.w   #14000,EGMCurrent           ; Last available RAM?
                jlo     END_ADC                     ; (Separate 500bytes for Stack)
                clr.w   R10                         ; If yes reset EGM counter, creates circular buffer
                clr.w   EGMCurrent                  ; Reset current address
                mov.w   Initial_EGM,EGMCurrent      ; Put start address EGM into EGMCurrent
END_ADC         jmp     TBX_ISR                     ; Jump to Interrupt Handler Label

TBCCR5_ISR      bic.w   #CCIE,&TBCCTL5              ; Disable interrupts for this timer
                bic.w   #BIT2,&TBCCTL2              ; Reset Output of Register 2, Pulse_Signal
                mov.w   &TBR,&TBCCR3                ; Move Basic Pacing + Pulse Width to Register 3
                add.w   R9,&TBCCR3                  ; Add Refractory Period to Register 3
                jmp     TBX_ISR

;-------------------------------------------------------------------------------------------------------
WDT_ISR;        Watchdog Timer Interrupt Handler
;-------------------------------------------------------------------------------------------------------

Batt_Int        bis.b   #BIT2,&P3DIR                ; Test WDT routine erase later
                bis.b   #BIT2,&P3OUT                ; Erase later
                cmp.w   #0,R13                      ; Watchdog Interval Register
                dec.w   R13                         ; Interval is Setup by counting every time it
                jnz     End_WDT                     ; enters the interrupt handler, Int = (R13)*1s
```

```
;-------------------------------------------------------------------------------------------------------
;    Module C - Battery Check
;-------------------------------------------------------------------------------------------------------
SVS2            push    R11                             ; Stored R11 on stack and initialize it
                push    R14                             ; Stored R14 on stack and initialize it

                inc.w   Hist_Battery                    ; Histogram, counts frequency of Battery check
                mov.w   BatteryStatus,R11               ; Move Battery Status to R11
                mov.b   #41h,R14                        ; Set 65 loops to wait for SVS to turn on and stabilize
Setup_SVS_On2   mov.b   SVS_9(R11),&SVSCTL              ; Turn on SVS, set threshold to previous battery status
Set_loop2       dec.b   R14                             ; Decrease R14, each decrease takes ~1us for DCO = 800kHz
                jnz     Set_loop2
                bit.b   #BIT0,&SVSCTL                   ; Test if AVcc = V_bs (example if V_bs = 3.0V)
                jz      End_SVS2                        ; jump to end of routine
                add.w   #1,BatteryStatus                ; If AVcc ~=  V_bs, then set V_bs = V_bs + 1
                                                        ; (example, next pass V_bs = 2.8V)

End_SVS2        clr.b   &SVSCTL                         ; Turn off SVS
                pop     R14                             ; Restored R14
                pop     R11                             ; Restored R11

;-------------------------------------------------------------------------------------------------------
;    Module D - Electrode Check
;-------------------------------------------------------------------------------------------------------

Electrode2      push    R11                             ; Stored R11 on stack and initialize it
                push    R12                             ; Stored R12 on stack and initialize it

Setup_CompA2    mov.b   #CARSEL+CAREF_1+CAIES,&CACTL1   ; Sets Vref = .25Vcc applied to - terminal
                                                        ; Enable Interrupts and negative trigger
                mov.b   #P2CA0+CAF,&CACTL2              ; Connects CA0 to P2.3 and Filters output

                inc.w   Hist_Electrode                  ; Histogram of Electrode measurement
Setup_P2        bic.b   #CAPD2,&CAPD                    ; Enable Port 2.2
                bis.b   #CAPD4,&CAPD                    ; Disable Port 2.5

MeasureRmeas2   bic.b   #CAPD4,&CAPD                    ; Enable Port 2.2
                bis.b   #BIT4,&P2DIR                    ; Set P2.2 as output
                bic.b   #BIT3+BIT2,&P2DIR               ; Set P2.4 as input
                bis.b   #BIT4,&P2OUT                    ; Charge to 7*tau = 7*33n*500 = 10.6us,
                                                        ; -  MCLK = 800kHz
                mov.w   #50,R11                         ; Set counter to 40 us to charge Cap tau =
Charging        dec.w   R11                             ;
                jnz     Charging                        ;
                bis.b   #CAON,&CACTL1                   ; Turn on comparator A, CompOut = 1
Disch_Rmeas2    bic.b   #BIT4,&P2OUT                    ; Discharge Cap, when
                                                        ; - Vcap < .25Vcc CompOut = 0
Discharging     inc.w   R11                             ; 2 cycles, Each step = 1.25us
                                                        ; - for MCLK = 800kHz
                bit.b   #BIT0,&CACTL2                   ; 7 cycles, Check if CompOut = 0
                jnz     Discharging                     ; 2 cycles
                mov.w   R11,tMeas                       ;
                bic.b   #CAON,&CACTL1                   ; Turn on comparator A, CompOut = 1
                cmp.w   #3,R11                          ;
```

```
                jne     Lead_Fracture2                  ;
                mov.w   #3333h,ElectrodeStatus          ; 3333 code for good lead
                bic.w   #BIT9,R5                        ; Set Electrode bit to indicate
                jmp     End_Electrode2                  ; good impedance
Lead_Fracture2  cmp.w   #8,R11                          ;
                jlo     Isul_Defect2                    ;
                mov.w   #4444h,ElectrodeStatus          ; 4444 code for Lead Fracture
                bis.w   #BITE,R6                        ;
                jmp     End_Electrode2                  ;
Isul_Defect2    mov.w   #2222h,ElectrodeStatus          ; 2222 code for Isulation Defect
                bis.w   #BITE,R6                        ;
End_Electrode2  pop     R12                             ; Restored R11
                pop     R11                             ; Restored R12
;-------------------------------------------------------------------------------------------------------
                mov.w   WDT_Const,R13                   ; Reinitialize Watchdog Counter
                bit.b   #BIT0,&P2IE                     ; Test if P2.0 is already set (avoid run condition)
                jnz     TurnOff_RC                      ; If set jump to Turn off RC timer
                bis.b   #BIT0,&P2IE                     ; If not set interrupts for P2.0
                bic.b   #BIT0,&P2IFG                    ; - and clear P2.0 interrupt flag
TurnOff_RC      bic.b   #BIT1,&P3DIR                    ; Set Input Direction
                bic.b   #BIT5,&P1IE                     ; Disable Interrupts for P1.5 for RC Timer
                bic.b   #BIT5,&P1IFG                    ; Clear RC Timer Interrupt Flag
                ;mov.w   Max_Rate,Prev_Rate             ; Initialize Max Ouput Rate
Check_NewParam  bic.w   #BITB,R4                        ; Reset to retest hight frequency condition
                bit.w   #BITF,R5                        ; Check for new parameters
                jz      End_WDT                         ; if 0 jump to END_WDT
Prepare_Stack   sub.w   #4,R1                           ; Decrease content of Stack to eliminate stored
                jmp     Basic_Pace                      ; - SR and PC by Interrupt. Jump to Basic_Pace
End_WDT         bic.b   #BIT2,&P3OUT                    ; Erase later
                reti                                    ; - to set new parameters.

;-------------------------------------------------------------------------------------------------------
P1_ISR;         Port1 Interrupt Handler
;-------------------------------------------------------------------------------------------------------

                push    R14                             ; Saves contents of R14 and Stack
                bit.b   #BIT4,&P1IN                     ; Test if Reed Switch Interrupt Flag is set
                jnz     Setup_VOO                       ; If set jump to end of routine
                clr.b   &P1IFG                          ; Clear P1 interrupt flags
                bit.w   #BITC,R6                        ; Check if charging or discharging
                jnz     Charge_RC                       ; Jump to discharge cap

Discharge_RC    bis.b   #BIT5,&P1DIR                    ; P1.5 as output to dischage capacitor
                bis.b   #BIT0,&P3OUT                    ; Set P3.0 Start Aux pulse
                mov.w   #020h,R14                       ; 1 cycle, each cycle = 1.25us for MCLK = 800kHz
Aux_PW          dec.w   R14                             ; 1 cycle,
                jnz     Aux_PW                          ; 2 cycles, Total cycles = 97, 97*1.25u = .121ms
                bic.b   #BIT0,&P3OUT                    ; Clear P3.0, end Aux Pulse Width
                bis.b   #BIT5,P1IES                     ; Interrupt set to falling edge
                bis.w   #BITC,R6                        ; Set to indicate RC need charging
                jmp     End_P1_ISR                      ; Jump to end of routine

Charge_RC       bic.b   #BIT5,&P1DIR                    ; Set as input to charge capacitor
                ;bis.b   #BIT6,&P1OUT                   ; High to initiate Charging
                bic.w   #BITC,R6                        ; Reset to indicate RC need discharging
```

```
                bic.b   #BIT5,P1IES                     ; Interrupt set to raising edge
                jmp     End_P1_ISR                      ; Jump to end of routine

Setup_VOO       inc.w   Hist_VOO                        ; Histogram of Reed Switch activation
                bic.b   #BIT0,&P2IE                     ; Disable Interrupts from comparator
                bic.b   #BIT1,&P2OUT                    ; Turn off comparator
                cmp.w   #5,BatteryStatus                ; Compare if Battery voltage < 2.3V
                jlo     MagnetNormal                    ; If true jump to Magnet Normal rate (96 bpm)
                cmp.w   #7,BatteryStatus                ; Compare if Battery voltage is > = 2.1V
                jhs     MagnetEOL                       ; If true jump to Magnet EOL rate (74 bpm)
                mov.w   Magnet_ERI,&TBCCR1              ; Set Basic Pace to Magnet of Elective
                mov.w   Magnet_ERI,Magnet_Const         ; - Replacement (84 bpm)
                jmp     End_P1_ISR                      ;
MagnetNormal    mov.w   Magnet_Normal,&TBCCR1           ; Set Basic Pace to Magnet Normal
                mov.w   Magnet_Normal,Magnet_Const      ; Store normal magnet pace in memory
                jmp     End_P1_ISR                      ;
MagnetEOL       mov.w   Magnet_EOL,&TBCCR1              ; Set Basic Pace to Magnet EOL
                mov.w   Magnet_EOL,Magnet_Const         ; Store End of Life pace in memory
                bic.b   #BIT4,&P1IE                     ; Set bit to identify first pass of Reed Switch
                bic.w   #BIT2,&TBCCTL2                  ; In the case Port1 interrupt consides with
                                                        ; - Basic Pace interrupt.
End_P1_ISR      pop     R14                             ; Restore contents of R14 from Stack
                reti

;-------------------------------------------------------------------------------------------------------
P2_ISR;         Port2 Interrupt Handler
;-------------------------------------------------------------------------------------------------------

                push    R11                             ;
                push    R12                             ;
                bic.b   #BIT0,&P2IFG                    ; Clears P2.0 Interrupt Flag
In_Freq_Check   mov.w   &TBR,R11                        ; Move previous Current Time
                mov.w   Prev_Freq,R12                   ;
                mov.w   R11,Prev_Freq                   ; Store current time into Prev_Freq
                sub.w   R12,R11                         ; Substract Current time from previous freq sample
                cmp.w   Max_Freq,R11                    ; Check if Int generate before frequency Threshold
                jlo     Count_Freq                      ; Jump to check if occurs 10 times in a row
VVI_Mode        mov.w   R7,&TBCCR1                      ; Set Basic Pace for Timer_B
                add.w   &TBR,&TBCCR1                    ; Initialize Basic Pace to current time
                inc.w   Hist_Sensed                     ; Histogram of Sensed Events
                bit.w   #BITF,R6                        ; Check if Hysteresis is on
                jz      No_Escape_Int                   ; Jump to Reset Timer_B
                add.w   R10,&TBCCR1                     ; Add hysteresis to Basic Pace
No_Escape_Int   mov.w   &TBCCR1,&TBCCR2                 ; Move Basic Pacing Time + Hysteresis to Counter Register 2
                add.w   R8,&TBCCR2                      ;
                mov.w   &TBCCR2,&TBCCR3                 ; Move Basic Pacing + Pulse Width to Register 3
                add.w   R9,&TBCCR3                      ; Add Refractory Period to Register 3
                bis.w   #MC_2,&TBCTL                    ; Turn On Timer_B in Continuous Mode
                bit.w   #BITA,R4                        ; Check Pacing Mode, VVI (0) or VVT (1)
                jz      End_P2ISR                       ; If 1 jump to VVI_Mode
VVT_Mode        mov.w   &TBR,&TBCCR5                    ;
                bic.b   #BIT1,&P2OUT                    ; Turn off comparator
                add.w   R8,&TBCCR5                      ; Set trigger pulse width
                mov.w   #CCIE,&TBCCTL5                  ; CCR3 Output Mode, interrupt enabled
                bis.w   #BIT2,&TBCCTL2                  ; Set Output of Register 2, Set Pulse_Signal
```

```
                jmp     End_P2ISR                       ;
Count_Freq      bis.w   #BITB,R4                        ; Set to notify hight frequency.
                bic.b   #BIT0,&P2IE                     ; Disable Port 2 Interrupts
End_P2ISR       ;bis.b   #BIT1,&P2OUT                   ; Turn on comparator
                pop     R12                             ;
                pop     R11                             ;
                reti                                    ;

;-------------------------------------------------------------------------------------------------------
;               Definition of Programming Constants
;-------------------------------------------------------------------------------------------------------

                            ; ROM Memory definition for Constants

Basic_Pacing_0  DW 61435      ; 32 bpm, 1, Define Basic_Pacing, units: min^-1, Flash Location =
Basic_Pacing_1  DW 57821      ; 34 bpm
Basic_Pacing_2  DW 54609      ; 36 bpm
Basic_Pacing_3  DW 51735      ; 38 bpm
Basic_Pacing_4  DW 49148      ; 40 bpm
Basic_Pacing_5  DW 46808      ; 42 bpm
Basic_Pacing_6  DW 44680      ; 44 bpm
Basic_Pacing_7  DW 42737      ; 46 bpm
Basic_Pacing_8  DW 40957      ; 48 bpm
Basic_Pacing_9  DW 39318      ; 50 bpm
Basic_Pacing_10 DW 37806      ; 52 bpm
Basic_Pacing_11 DW 36406      ; 54 bpm
Basic_Pacing_12 DW 35106      ; 56 bpm
Basic_Pacing_13 DW 33895      ; 58 bpm
Basic_Pacing_14 DW 32765      ; 60 bpm
Basic_Pacing_15 DW 31708      ; 62 bpm
Basic_Pacing_16 DW 30718      ; 64 bpm
Basic_Pacing_17 DW 29787      ; 66 bpm
Basic_Pacing_18 DW 28911      ; 68 bpm
Basic_Pacing_19 DW 28085      ; 70 bpm
Basic_Pacing_20 DW 27304      ; 72 bpm
Basic_Pacing_21 DW 26567      ; 74 bpm
Basic_Pacing_22 DW 25867      ; 76 bpm
Basic_Pacing_23 DW 25204      ; 78 bpm
Basic_Pacing_24 DW 24574      ; 80 bpm
Basic_Pacing_25 DW 23975      ; 82 bpm
Basic_Pacing_26 DW 23404      ; 84 bpm
Basic_Pacing_27 DW 22860      ; 86 bpm
Basic_Pacing_28 DW 22340      ; 88 bpm
Basic_Pacing_29 DW 21844      ; 90 bpm
Basic_Pacing_30 DW 21369      ; 92 bpm
Basic_Pacing_31 DW 20914      ; 94 bpm
Basic_Pacing_32 DW 20478      ; 96 bpm
Basic_Pacing_33 DW 20060      ; 98 bpm
Basic_Pacing_34 DW 19659      ; 100 bpm
Basic_Pacing_35 DW 19274      ; 102 bpm
Basic_Pacing_36 DW 18903      ; 104 bpm
Basic_Pacing_37 DW 18546      ; 106 bpm
Basic_Pacing_38 DW 18203      ; 108 bpm
Basic_Pacing_39 DW 17872      ; 110 bpm
Basic_Pacing_40 DW 17553      ; 112 bpm
```

```
Basic_Pacing_41 DW 17245      ; 114 bpm
Basic_Pacing_42 DW 16948      ; 116 bpm
Basic_Pacing_43 DW 16660      ; 118 bpm
Basic_Pacing_44 DW 16383      ; 120 bpm

Hysteresis_0    DW 16114      ; 122 bpm, Define Hysteresis, units: min^-1, Flash Location =
Hysteresis_1    DW 15854      ; 124 bpm
Hysteresis_2    DW 15603      ; 126 bpm
Hysteresis_3    DW 15359      ; 128 bpm
Hysteresis_4    DW 15122      ; 130 bpm, 50, 4450
Hysteresis_5    DW 14893      ; 132 bpm
Hysteresis_6    DW 14671      ; 134 bpm
Hysteresis_7    DW 14455      ; 136 bpm
Hysteresis_8    DW 14246      ; 138 bpm
Hysteresis_9    DW 14042      ; 140 bpm, 4460
Hysteresis_10   DW 13845      ; 142 bpm
Hysteresis_11   DW 13652      ; 144 bpm
Hysteresis_12   DW 13465      ; 146 bpm
Hysteresis_13   DW 13283      ; 148 bpm
Hysteresis_14   DW 13106      ; 150 bpm, 60, 4470
Hysteresis_15   DW 12934      ; 152 bpm
Hysteresis_16   DW 12766      ; 154 bpm
Hysteresis_17   DW 12602      ; 156 bpm
Hysteresis_18   DW 12443      ; 158 bpm
Hysteresis_19   DW 12287      ; 160 bpm, 4480

Pulse_Width_0   DW 2          ; .070 ms, Define Pulse_Width, units: ms, Flash Location =
Pulse_Width_1   DW 5          ; .145 ms
Pulse_Width_2   DW 7          ; .220 ms
Pulse_Width_3   DW 10         ; .295 ms
Pulse_Width_4   DW 12         ; .370 ms
Pulse_Width_5   DW 15         ; .445 ms
Pulse_Width_6   DW 17         ; .520 ms
Pulse_Width_7   DW 19         ; .595 ms
Pulse_Width_8   DW 22         ; .670 ms
Pulse_Width_9   DW 24         ; .745 ms
Pulse_Width_10  DW 27         ; .820 ms
Pulse_Width_11  DW 29         ; .895 ms
Pulse_Width_12  DW 32         ; .970 ms
Pulse_Width_13  DW 34         ; 1.045 ms
Pulse_Width_14  DW 37         ; 1.120 ms
Pulse_Width_15  DW 39         ; 1.195 ms
Pulse_Width_16  DW 42         ; 1.270 ms
Pulse_Width_17  DW 44         ; 1.345 ms
Pulse_Width_18  DW 47         ; 1.420 ms
Pulse_Width_19  DW 49         ; 1.495 ms

Refractory_0    DW 6553       ; 200 ms, Define Refractory Period, units: ms, Flash Location =
Refractory_1    DW 7045       ; 215 ms
Refractory_2    DW 7536       ; 230 ms
Refractory_3    DW 8028       ; 245 ms
Refractory_4    DW 8519       ; 260 ms
Refractory_5    DW 9010       ; 275 ms
Refractory_6    DW 9502       ; 290 ms
Refractory_7    DW 9993       ; 305 ms
```

```
Refractory_8    DW 10485      ; 320 ms
Refractory_9    DW 10976      ; 335 ms
Refractory_10   DW 11468      ; 350 ms
Refractory_11   DW 11959      ; 365 ms
Refractory_12   DW 12451      ; 380 ms
Refractory_13   DW 12942      ; 395 ms
Refractory_14   DW 13434      ; 410 ms
Refractory_15   DW 13925      ; 425 ms
Refractory_16   DW 14417      ; 440 ms
Refractory_17   DW 14908      ; 455 ms
Refractory_18   DW 15400      ; 470 ms
Refractory_19   DW 15891      ; 485 ms
Refractory_20   DW 16383      ; 500 ms

WDT_Interval_0  DW 60         ; 1 min Define Batt_Interval, units: min, Flash Location =
WDT_Interval_1  DW 120        ; 2 min
WDT_Interval_2  DW 300        ; 5 min
WDT_Interval_3  DW 600        ; 10 min

Sensitivity_1   DW 3          ;  33 mV, Define Variable Sensitivity, units: mV, Flash Location =
Sensitivity_2   DW 9          ;  98 mV
Sensitivity_3   DW 15         ; 164 mV
Sensitivity_4   DW 21         ; 230 mV
Sensitivity_5   DW 27         ; 295 mV
Sensitivity_6   DW 33         ; 361 mV
Sensitivity_7   DW 39         ; 426 mV
Sensitivity_8   DW 45         ; 492 mV
Sensitivity_9   DW 51         ; 558 mV
Sensitivity_10  DW 57         ; 623 mV
Sensitivity_11  DW 63         ; 689 mV
Sensitivity_12  DW 69         ; 755 mV
Sensitivity_13  DW 75         ; 820 mV
Sensitivity_14  DW 81         ; 886 mV
Sensitivity_15  DW 87         ; 951 mV

Pulse_Amp_1     DW 50         ; 2.0 V, Define Variable Pulse_Amplitude, units: V, Flash Location =
Pulse_Amp_2     DW 60         ; 2.5 V
Pulse_Amp_3     DW 70         ; 3.0 V
Pulse_Amp_4     DW 80         ; 3.5 V
Pulse_Amp_5     DW 90         ; 4.0 V
Pulse_Amp_6     DW 100        ; 4.5 V
Pulse_Amp_7     DW 115        ; 5.0 V
Pulse_Amp_8     DW 125        ; 5.5 V
Pulse_Amp_9     DW 135        ; 6.0 V
Pulse_Amp_10    DW 150        ; 6.5 V
Pulse_Amp_11    DW 160        ; 7.0 V
Pulse_Amp_12    DW 170        ; 7.5 V

Magnet_Normal   DW 20478      ; 96 bpm, Define MagnetNormal, units: min^-1, Flash Location =
Magnet_ERI      DW 23404      ; 84 bpm
Magnet_EOL      DW 26567      ; 74 bpm

SVS_9           DB 0090h      ; 2.90 V, Define Threshold Voltage SVS, units: Volts, Flash Location =
SVS_8           DB 0080h      ; 2.80 V
SVS_7           DB 0070h      ; 2.65 V
```

```
SVS_6           DB 0060h      ; 2.50 V
SVS_5           DB 0050h      ; 2.40 V
SVS_4           DB 0040h      ; 2.30 V
SVS_3           DB 0030h      ; 2.20 V
SVS_2           DB 0020h      ; 2.10 V

ECG_RATE_0      DW 328        ; 100 Hz, Define EGM sampling frequency, 400Hz, units: Hz, Flash Location =
ECG_RATE_1      DW 164        ; 200 Hz
ECG_RATE_2      DW 108        ; 300 Hz
ECG_RATE_3      DW 82         ; 400 Hz
ECG_RATE_4      DW 66         ; 500 Hz
ECG_RATE_5      DW 56         ; 600 Hz
ECG_RATE_6      DW 46         ; 700 Hz
ECG_RATE_7      DW 41         ; 800 Hz
ECG_RATE_8      DW 36         ; 900 Hz

Max_Freq        DW  2584      ; Maximum input frequency  = 12 Hz
Max_Rate        DW  9830      ; Maximum Output Rate = 200 bpm

Initial_EGM     DW  4394      ; First address of EGM

Upper_Rate_0    DW 24574      ; Define Upper_Rate in VVT mode, units: min^-1, Flash Location =
Upper_Rate_1    DW 23975      ;
Upper_Rate_2    DW 23404      ;
Upper_Rate_3    DW 22860      ;
Upper_Rate_4    DW 22340      ;
Upper_Rate_5    DW 21844      ;
Upper_Rate_6    DW 21369      ;
Upper_Rate_7    DW 20914      ;
Upper_Rate_8    DW 20478      ;
Upper_Rate_9    DW 20060      ;
Upper_Rate_10   DW 19659      ;
Upper_Rate_11   DW 19274      ;
Upper_Rate_12   DW 18903      ;
Upper_Rate_13   DW 18546      ;
Upper_Rate_14   DW 18203      ;
Upper_Rate_15   DW 17872      ;
Upper_Rate_16   DW 17553      ;
Upper_Rate_17   DW 17245      ;
Upper_Rate_18   DW 16948      ;
Upper_Rate_19   DW 16660      ;
Upper_Rate_20   DW 16383      ;
Upper_Rate_21   DW 16114      ;
Upper_Rate_22   DW 15854      ;
Upper_Rate_23   DW 15603      ;
Upper_Rate_24   DW 15359      ;
Upper_Rate_25   DW 15122      ;
Upper_Rate_26   DW 14893      ;
Upper_Rate_27   DW 14671      ;
Upper_Rate_28   DW 14455      ;
Upper_Rate_29   DW 14246      ;
Upper_Rate_30   DW 14042      ;
Upper_Rate_31   DW 13845      ;

;-----------------------------------------------------------------------------------------------
```

```
;           Interrupt Vectors Used MSP430x4xx
;-------------------------------------------------------------------------------------------------------
                ORG     0FFFEh      ; MSP430 RESET Vector
                DW      RESET       ;
                ORG     0FFF8h      ; Timer_BX Vector
                DW      TBX_ISR     ;
                ORG     0FFF4h      ; Watchdog Timer Vector
                DW      WDT_ISR     ;
                ORG     0FFE8h      ; Port1 Interrupt Vector
                DW      P1_ISR      ;
                ORG     0FFE2h      ; Port2 Interrupt Vector
                DW      P2_ISR      ;
                END
```

BIBLIOGRAPHY

[1] Richard Sutton and Ivan Bourgeois. *The Foundation of Cardiac Pacing, Pt. I: An Illustrated Practical Guide to Basic Pacing*. Bakken Research Center Series. Futura Publishing Company, first edition, 1991.

[2] Gary Legg. Mechanical hearts tick with chips. *Design News*, June 2001.

[3] Fernando Silveira and Denis Flandre. *Low Power Analog CMOS for Cardiac Pacemakers: Design and Optimization in Bulk and SOI Technologies*. Kluwer Academic Publishers, first edition, 2004.

[4] Richard E. Klabunde Ph. D. *Cardiac Cycle*. Harvey Project, physiology on the web. Cardiovarcular Physiology Concepts, http://www.cvphysiology.com/, August 2005.

[5] John G. Webster, editor. *Medical Instrumentation, Application and Design*. John Wiley and Sons, third edition, 1998.

[6] American Heart Association. Heart disease and stroke statistics - 2004 update, 2004.

[7] BioBusiness Networks Corporations. Corporate overview: Biophan technologies inc. http://www.biobn.com/index.cfm?page=viewcompany&CoID=102.

[8] Endocardial Solutions. Advanced diagnostic solutions for complex cardiac arrhythmias. http://www.endocardial.com/media/pdfs/Annual.pdf, 1999.

[9] D.J. Woollons. To beat or not to beat: the history and development of heart pacemakers. *Engineering science and educational journal*, pages 259 – 268, December 1995.

[10] Medware Inc. First implantable cardiac pacemaker. http://www.slip.net/ medware/pace.htm.

[11] Ph.D Notes from Prof. Eduado Juan. Electrocardiography and the electrocardograph; pacemakers and cardiac pacemaking. Material supplied, 2004.

[12] Berstein A. D., Daubert J., and Fletcher R. D. The revised naspe/bpeg generic code for antibradycardia, adaptive-rate, and multisite pacing. *Journal of Pacing and Clinical Electrophysiology*, Volume 25, 2002.

[13] Renz A. Weil G., Engl W.L. Integrated pacemakers. *IEEE Journal of Solid-State Circuits*, 5(2):67 – 73, April 1970.

[14] R. Walters and G. Bivins. A custom cmos ic for pacemaker applications. In *Solid-State Circuits Conference. Digest of Technical Papers. 1979 IEEE International*, pages 200–201, 1979.

[15] L.J. Stotts, K.R. Infinger, J. Babka, and D. Genzer. An 8-bit microcomputer with analog subsystems for implantable biomedical application. *IEEE Journal of Solid-State Circuits*, 24(2):292–300, 1989.

[16] Chung Wen-Yaw, Lin Heh-Sen, Yang Chung-Huang, Sun Tai-Ping, Chen Guo-Ching, and Hsieh Chang-Horng. Modular-based design and implementation of a programmable vvi mode pacemaker. In *Engineering in Medicine and Biology Society, 1995. IEEE 17th Annual Conference*, volume 2, pages 1689–1690, Montreal, Que., 1995.

[17] Kim Jungkuk and P. Haefner. An automatic pacemaker sensing algorithm using automatic gain control. In *Engineering in Medicine and Biology Society, 1998. Proceedings of the 20th Annual International Conference of the IEEE*, pages 423–425, Hong Kong, 1998.

[18] R. Frohlich, A. Bolz, R. Hardt, M. Hubmann, and M. Schaldach. Automatic amplitude adjustment of the pacemaker output voltage. In *Engineering in Medicine and Biology Society, 1994. Engineering Advances: New Opportunities for Biomedical Engineers. Proceedings of the 16th Annual International Conference of the IEEE*, pages 55–56, Baltimore, MD, 1994.

[19] D.B. Shaw and M. Horwood. Recording pacemakers memory size-what should be recorded. In *Intelligent Cardiac Implants, IEE Colloquium on*, pages 1–3, London, 1993.

[20] M. Schaldach. Systolic time intervals as a control of rate adaptive pacing. In *Engineering in Medicine and Biology Society, 1989. Images of the Twenty-First Century. Proceedings of the Annual International Conference of the IEEE Engineering in*, pages 1407–1410, Seattle, WA, 1989.

[21] M.P.R. Hexamer, M. Meine, A. Kloppe, and E. Werner. Rate-responsive pacing based on the atrio-ventricular conduction time. *IEEE Transactions on Biomedical Engineering*, 49(3):185–195, 2002.

[22] Wilson Greatbatch. *Lithium-Iodine Battery Description and Specifications*. http://www.greatbatch.com/mp/ips/batteryspecifications/LithiumIodine.asp, 2006.

[23] Venkateswara Sarma Mallela, V. Ilankumaran, and N.Srinivasa Rao. Trends in cardiac pacemaker batteries. *Indian Pacing and Electrophysiology Journal*, Technical Series(4(4)):201 – 212, 2004.

[24] Biotronik GmbH & Co. What you should know about your pacemaker. Technical report, Biotronik, 2000.

[25] V. Parsonnet and A.D. Bernstein. The 1989 world survey of cardiac pacing. *Pacing Clinical Electrophysiology*, 14:2073–6, Nov 1991.

[26] S. J. Connolly, C. Kerr, M. Gent, and S. Yusuf. Dual-chamber versus ventricular pacing: Critical appraisal of current data. *Circulation*, 94:578–583, 1996.

[27] I. E. Ovsyshcher, D. L. Hayes, and S. Furman. Dual-chamber pacing is superior to ventricular pacing: Fact or controversy? *Circulation*, 97:2368–2370, 1998.

[28] R.S. Sanders and M.T. Lee. Implantable pacemakers. In *Proceedings of the IEEE*, volume 84, pages 480–486, 1996.

[29] P. Koopman. Embedded system design issues (the rest of the story). In *Computer Design: VLSI in Computers and Processors, 1996. ICCD '96. Proceedings., 1996 IEEE International Conference on*, pages 310–317, Austin, TX, 1996.

[30] Frank Vahid and Tony Givargis. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, first edition, 2002.

[31] T. Jamil. RISC versus CISC. *IEEE Potentials*, 14(3):13–16, 1995.

[32] W. Stallings. Reduced instruction set computer architecture. *Proceedings of the IEEE*, 76(1):38–55, Jan. 1988.

[33] B. Lazzerini. Effective VLSI processor architectures for HLL computers: the RISC approach. *IEEE Micro*, 9(1):57–65, 1989.

[34] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann Publishers, third edition edition, 2003.

[35] CCC. *TEROS SSI 503 Product Specifications*. Centro de Construcción de Cardioestimuladores del Uruguay S.A., http://www.ccc.com.uy/Teros503.htm, 2005.

[36] Dallas Semiconductor Maxim. Maxq2000 - low-power lcd microcontroller. Datasheet, Dallas Semiconductor Maxim, January 2006.

[37] Microchip. Pic18fxx2 - high performance, enhanced flash micorocontrollers with 10-bit a/d. Datasheet, Microchip Technology Inc, 2002.

[38] TI. Msp430x15x, msp430x16x, msp430x161x mixed signal microcontroller. Datasheet, Texas Instruments Inc, March 2005.

[39] Mike Mitchell. Choosing an ultralow-power mcu. *Application Report*, SLAA207, June 2004.

[40] L. Lentola, A. Mozzi, A. Neviani, and A. Baschirotto. A 1μa front-end for pacemaker atrial sensing channels. In *Solid-State Circuits Conference, 2001. ESSCIRC 2001. Proceedings of the 27th European*, pages 253–256, 2001.

[41] F. Silveira and D. Flandre. A 110 na pacemaker sensing channel in CMOS on silicon-on-insulator. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 5, 2002.

[42] A. Gerosa, A. Novo, A. Mengalli, and A. Neviani. A micro-power low noise log-domain amplifier for the sensing chain of a cardiac pacemaker. In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, volume 1, pages 296–299, Sydney, NSW, 2001.

[43] Jen-Shiun Chiang, Hsueh-Ping Chen, and Cheng ming Ying. A 1v 0.54 /spl mu/w fourth order switched capacitor filter with switched opamp technique for cardiac pacemaker sensing channel. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 1, 2003.

[44] A. Novo, A. Gerosa, A. Neviani, A. Mozzi, and E. Zanoni. A CMOS 0.8 μm programmable charge pump for the output stage of an implantable pacemaker. In *Devices, Circuits and Systems, 2000. Proceedings of the 2000 Third IEEE International Caracas Conference on*, pages 1–34, Cancun, 2000.

[45] Jieh-Tsorng Wu and Kuen-Long Chang. MOS charge pumps for low-voltage operation. *IEEE Journal of Solid-State Circuits*, 33(4):592–597, 1998.

[46] S. Serge Barold, Roland X. Stroodandt, and Alfons F. Sinnaeve. *Cardiac Pacemakers Step by Step: An Illustrated Guide*. Futura. Blackwell Publishing, first edition, 2004.

[47] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, 1994.

[48] N. Kavvadias, P. Neofotistos, S. Nikolaidis, C.A. Kosmatopoulos, and T. Laopoulos. Measurements analysis of the software-related power consumption in microprocessors. *IEEE Transactions on Instrumentation and Measurement*, 53(4):1106–1112, 2004.

[49] V. Tiwari and M. Tien-Chien Lee. Power analysis of a 32-bit embedded microcontroller. In *Design Automation Conference, 1995. Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95., IFIP International Conference on Hardware Description Languages; IFIP International Conference on Very Large Scale Integration., Asian and South Pacific*, pages 141–148, Chiba, 1995.

[50] V. Tiwari, S. Malik, A. Wolfe, and M.T.-C. Lee. Instruction level power analysis and optimization of software. In *VLSI Design, 1996. Proceedings., Ninth International Conference on*, pages 326–328, Bangalore, 1996.

[51] J.T. Russell and M.F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. Proceedings., International Conference on*, pages 328–333, Austin, TX, 1998.

[52] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*, pages 185–190, 2000.

[53] Agilent Technologies. *Agilent 34401A Multimeter User's Guide*, fifth edition, March 2000.

[54] S. Nikolaidis, N. Kavvadias, and P.Neofotistos. Instruction level power measurements and analysis. Dekiverable, Aristotle University of Thessaloniki, http://electronics.physics.auth.gr/easy/, September 2002.

[55] C. Brandolese, W. Fomacian, F. Salice, and D. Sciuto. An instruction-level functionality-based energy estimation model for 32-bits microprocessors. In *Design Automation Conference, 2000. Proceedings 2000. 37th*, pages 346–350, 2000.

[56] Texas Instruments Inc. *MSP430x1xx Family User's Guide (slas368d)*, 2005.

[57] Chris Nagy. *Embedded Systems Design Using the TI MSP430 Series*. Embedded Technologies. Elsevier Science, first edition, 2003.

[58] Real-time-clock selection and optimization. Application note, Maxim, Dallas Semiconductor, June 2001.

[59] Wikipedia. Fever — wikipedia, the free encyclopedia, 2006.

[60] Mike Mitchell. Using pwm timer _ b as a dac. Application Report SLAA116, Texas Instruments, December 2000.

[61] Carter B. A single-supply op-amp circuit collection. Application report sloa058, Texas Instruments, Inc, November 2000.

[62] S. Franco. *Design with Operational Amplifiers and Analog Integrated Circuits*. McGraw Hill, third edition, 2002.