PROGRAMMABLE STIMULATOR SYSTEM FOR STUDY OF CARDIAC ARRHYTHMIAS

by

Andrew H. Chung

S.B., Massachusetts Institute of Technology (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1993

Author ....
Department of Electrical Engineering and Computer Science
August 27, 1993

Certified by ....
Richard J. Cohen
Professor
Thesis Supervisor

Accepted by ....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

PROGRAMMABLE STIMULATOR SYSTEM FOR STUDY OF CARDIAC ARRHYTHMIAS

by

Andrew H. Chung

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 1993, in partial fulfillment of the
requirements for the degree of
MASTER OF SCIENCE

Abstract

This thesis presents a microcomputer-based stimulator system which is used to produce a wide variety of pacing protocols for the studies of cardiac arrhythmias. The stimulator has two independent output channels: the first channel is for a stimulation through a single-site electrode and the second is for a moderate level stimulation through a multi-site electrodes sock or defibrillation catheters and patches. It is also equipped with an ability to amplify the intracardiac electrogram and synchronize the pacing with the electrical activity of the heart. The control software can be programmed for an induction of cardiac arrhythmias and other variety of pacing protocols.

The stimulator is used to investigate methods for the prevention and termination of lethal cardiac arrhythmias, such as ventricular tachycardia and fibrillation, by multi-site pacing. It detects the ventricular premature beats and applies a triggered pacing. This stimulator system is tested in the laboratory animal models of arrhythmias and is shown to function successfully.

Thesis Supervisor: Richard J. Cohen
Title: Professor

Acknowledgements

I would like to express my sincere appreciation to Prof. Richard J. Cohen for his guidance, advice, and encouragement. He provided an environment where I can learn and pursue my research project independently.

Other members of Dr. Cohen's group deserve my thanks for the many hours they spent helping me solve problems I faced. I would like to express my thanks to Kohei Ono, who was very much a part of the effort during the hardware development; Zhihao Yin and Darin Sherman, who were very resourceful during the software development; Dr. Aldea, Tom Mullen, Antonis Armoundas, and Ray Lee for their help in doing the animal experiments. Without their help and discussions, this thesis would have been impossible to complete. Last, thanks to Tom Mullen and Terri Bae for their help in editing this thesis.

Most of all, I would like to express my deep love for my parents and my sister. Without their constant love and care, I would not have made it through this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sudden cardiac death victimizes approximately 400,000 people in the United States alone every year. Because the event is both sudden and unexpected, almost two-thirds of the victims die before reaching the hospital [24]. The primary cause of sudden cardiac death in the vast majority of victims is ventricular fibrillation (VF), a disorganized pattern of electrical activity in the principal pumping chambers of the heart. The disorganized pattern of electrical activity results in random contractions and relaxations of cardiac muscle fibers which are ineffectual in pumping blood. Unless cardiopulmonary resuscitation is initiated immediately, death ensues in approximately four minutes from a lack of perfusion of the brain. The only reliable method for termination of VF is defibrillation, the delivery of a sufficiently strong direct current electric discharge to the fibrillating heart to reset the electrical activity of the heart.

Ventricular rhythm disturbances or ventricular arrhythmias, such as fibrillation and tachycardia, are usually developed during the acute phase of myocardial infarction, more commonly known as heart attack. These arrhythmias normally disappear when the infarction stabilizes. However, some patients do develop ventricular arrhythmias usually in the form of tachycardias some weeks or months after the onset of the infarction. Ventricular tachycardia itself is a dangerous rhythm disturbance because the heart rate is usually too fast and too poorly synchronized to allow effective pumping action by the heart. Ventricular tachycardia often develops into ventricular fibrillation. Therefore, these tachycardias must be treated. In some cases, antiarrhythmic drug treatment is effective in suppressing such tachycardias. However, current drug therapy are only modestly useful in prevention of these ventricular arrhythmias. Recently, an automatic internal cardiac defibrillator (AICD) has been implanted in high-risk patients. The internal defibrillator senses established VF and automatically administers a large counter electric shock to terminate it. Although the implantable defibrillator is useful for termination of ventricular fibrillation, it is not preventive. In addition, use of the internal defibrillator is also problematic in individuals with multiple recurrent episodes of fibrillation since internal defibrillation is traumatic and painful, and the patient generally loses consciousness prior to defibrillation.

This thesis presents a programmable stimulator system which is used in the studies of electrical pacing methods for the prevention of ventricular arrhythmias, tachycardia and fibrillation. Instead of treating established fibrillation by administration of a large countershock, a stimulator system is used to detect the initiation of ventricular arrhythmias and apply an electrical pacing to prevent ventricular tachycardia and fibrillation from developing. In order to study the prevention method, a stimulator is equipped with capabilities to sense the premature ventricular activities and deliver a moderate level stimulation through a multi-site electrode sock. A personal computer based system for amplifying the ventricular electrogram and delivering the stimuli of sufficient amplitude is developed. The pacing protocols to prevent and treat ventricular arrhythmias using an epicardial electrode array are studied on the experimental animal models, which are used to duplicate the arrhythmogenic effects of myocardial ischemia.

This thesis is organized as follows:

i. A brief review of the major features of the structure and function of the normal human heart is presented as a reference for subsequent discussion. Also, the most recent theory regarding the initiation of the cardiac arrhythmias is described. (Chapter 2)

ii. An overview of the programmable stimulator system, which consists of commercial components and an integrated system, is presented. The design of the integrated system is presented in detail. (Chapter 3)

iii. A review of the control program, which is used for the user interface and output control, is presented. Several stimulation protocols used for the experiment are discussed. (Chapter 4)

iv. Experimental study, where the system is used for induction of arrhythmias and multi-site pacing, is presented. (Chapter 5)

Chapter 2

Background

In order to lay the groundwork for later description of the pacing system specifications, this chapter is focused on providing the necessary physiological background. The first part of this chapter describes basic anatomy and physiology of the normal human heart. The mechanical events of the heart are then related to the electrocardigrm. Finally, the mechanisms of cardiac arrhythmias, which interfere with the normal functioning of the heart, are discussed and several examples of cardiac arrhythmias are presented.

2.1 The Heart

The human heart is a hollow, thickly muscular organ which facilitates blood flow through the vascular system by rhythmic contractions. The heart is responsible for supplying all organs of the body with the necessary life-sustaining nutrition and oxygen.

2.1.1 Anatomy of the Heart

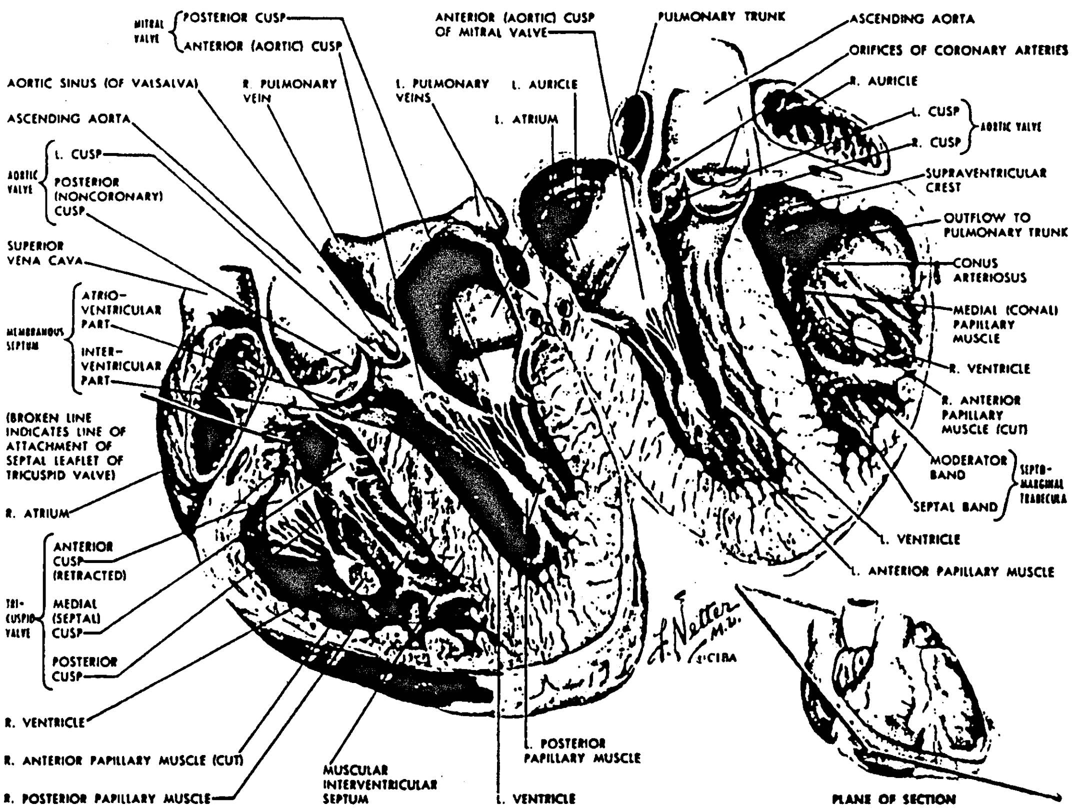

Figure 2-1: Cross-section of the heart (Reproduced from Netter FH: Heart, Vol. 5, The CIBA Collection of Medical Illustrations, CIBA, 1981)

The average human adult heart measures 12 cm from apex to base, 9 cm in width at its broadest point, and 6 cm in anterior-posterior dimension [25]. A cross-section of the heart is shown in Figure 2-1. Internally, the heart is divided into four distinct chambers; right and left atria, two small upper chambers with thin walls, and right and left ventricles, two lower chambers with thicker walls. The atrium and ventricle on the right side are separated from those on the left by a septum. There is an opening between the atrium and ventricle, which is guarded by a valve.

2.1.2 Cardiac Cycle

In a normal cardiac cycle, oxygen-depleted, venous blood returning from the peripheral circulation empties into the right atrium (RA) via the superior and inferior vena cavae (refer to Figure 2-1). From right atrium, blood is pumped through the tricuspid valve into the right ventricle (RV). This is the resting or filling phase of the cardiac cycle and is called diastole. The blood is then pumped from the right ventricle through the pulmonic valve into the pulmonary circulation in a contractile phase called systole. In the lungs, the blood is oxygenated and delivered to the left side of the heart. In parallel with the events on the right, oxygenated blood empties into the left atrium (LA) via several pulmonary veins. Left atrial contraction then propels this blood through the mitral (bicuspid) valve into the left ventricle (LV). The left ventricle, by far the most muscular of all the heart chambers, pumps blood through the aortic valve into the ascending aorta. From there, blood is distributed through the systemic circulation, eventually to return to the right atrium thereby completing the circuit. The pressures inside of the heart chambers and the aorta during the cardiac cycle are shown in Figure 2-2, with phases of systole and diastole noted. Corresponding heart sounds and electrocardiogram are also provided.

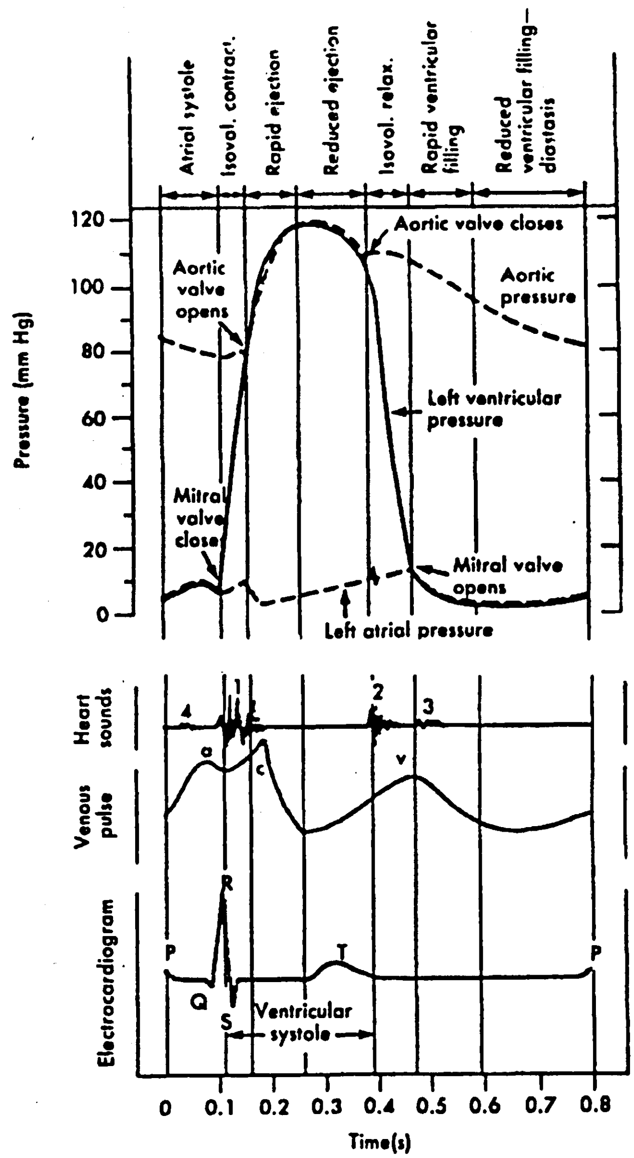

Figure 2-2: Cardiac Cycle (Reproduced from Berne and Levy: Cardiovascular Physiology, 1992)

2.1.3 Cardiac Conduction System

A specialized electrical conduction system normally regulates the steady beating of the heart. The signal giving rise to the cardiac cycle emanates from a cluster of conduction tissue cells collectively known as the sinoatrial node, as shown in Figure 2-3. This node, located at the top of the right atrium, establishes the tempo of the heartbeat; hence, it is often referred to as the cardiac pacemaker. It sets the tempo simply because it issues impulses more frequently than do other cardiac regions, once about every 830 milliseconds. If something provoks another part of the heart to fire at a faster rate, it would become the new pacemaker. Although the sinoatrial node can respond to signals from outside the heart, it usually becomes active spontaneously, a capability known as automaticity.

Impulses born at a cell in the sinoatrial node speed nearly instantly through the rest of the node, and from there, they course through the entire heart in the span of 160 to 200 milliseconds. Travelling along conduction fibers, they first race across both atria and then regroup at the atrioventricular node, a cellular cluster centrally located atop the ventricles. After a pause, they course down the ventricles along an A-V (His) bundle that divides into two branches known as left and right bundles branches; these further ramify to form arbors of thinner projections called Purkinje fibers. One arborized bundle serves each ventricle, sending signals first along the surface of the septum to the tip of the heart, the apex, and, from there, up along the inner surface of the external or lateral walls to the top of the ventricle.

As impulses from the conduction fibers reach muscle, they activate the overlying cells. Muscle cells, too, are capable of relaying impulses, although more slowly than do conduction fibers. The cells of the inner surface of the wall, the endocardium, depolarizes first and relay the impulses through the thickness of the muscle to the outer surface, the epicardium. Depolarization, in turn, triggers contraction that forces

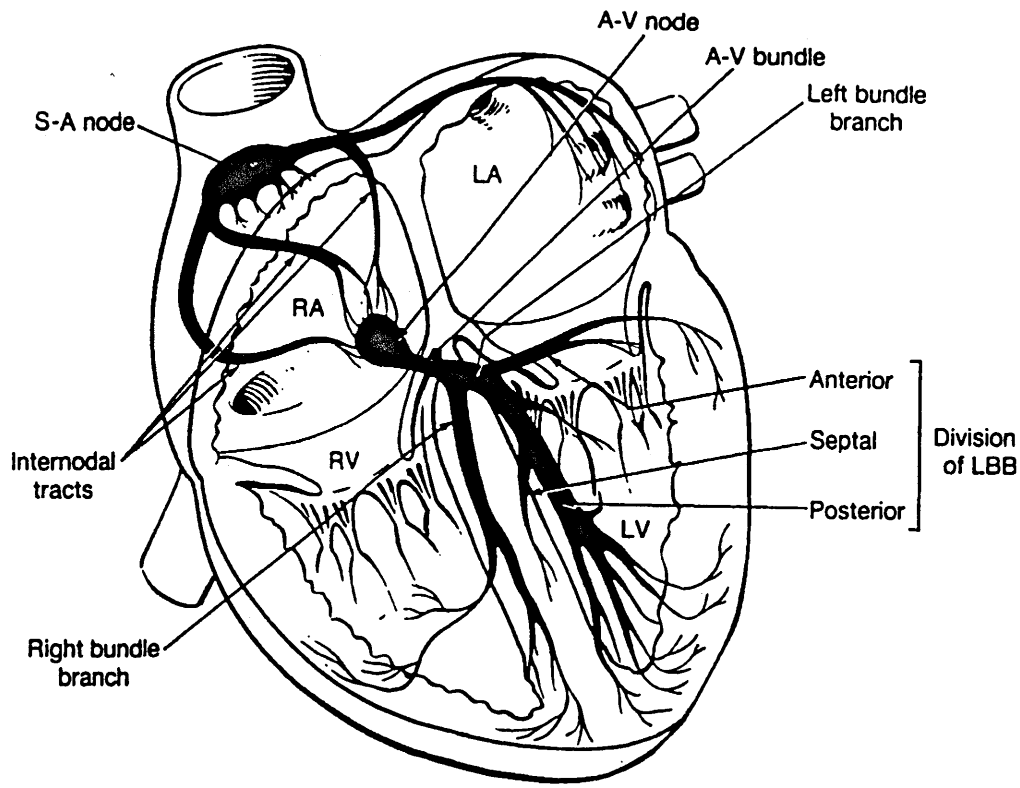

Figure 2-3: Electrical conduction system of the heart (RV, right ventricle; LV, left ventricle; LBB, left bundle branch system; A-V, atrioventricular; RA, right atrium; LA, left atrium)

blood through the heart and into the arterial circulation.

2.2 Electrocardiogram

The series of mechanical events outlined above can be tracked at the body surface by monitoring the surface potentials which are caused by the flow of ionic currents within the cardiac structures. The recording of such body surface potentials is known as electrocardiography, and a single recording of the potential difference between two points on the body surface is referred to as an electrocardiogram. Different morphologies of the cardiac action potentials at various locations within the heart are combined to give the electrocardiogram, shown in Figure 2-4. A theoretical basis for the relationship between body surface potentials and myocardial transmembrane potentials can be found in major texts [4].

The normal electrocardiogram consists of a P wave, a QRS complex, a T wave, and an infrequent U wave (refer to Figure 2-4). The P wave, temporally the first component of the ECG, is a result of atrial depolarization. The QRS complex, which follows the P wave after a delay of approximatley 150 millisecondss (A-V nodal delay), is the result of ventricular depolarization. The T wave, which follows the QRS complex by a variable period of time, is the result of ventricular repolarization. The repolarization of the atria is not visible in the surface ECG partly because the mass of the atria is relatively small and the time course of the atrial repolarization coincides with ventricular depolarization. The infrequently seen U wave is thought to be the result of the repolarization of the His-Purkinje system. It follows the T wave by roughly 25 milliseconds because the action potential duration of the Purkinje cells is approximately 25 milliseconds longer than that of ordinary myocardial cells [4].

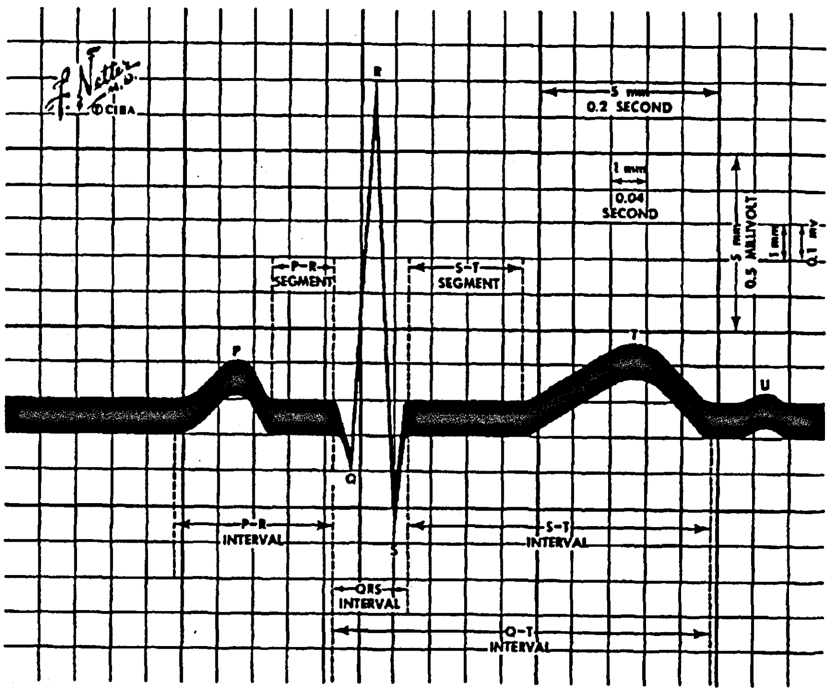

Figure 2-4: Schematized ECG tracing (Reproduced from Netter FH: Heart, Vol. 5, The CIBA Collection of Medical Illustrations, CIBA, 1981)

2.3 Arrhythmias

Disturbances in the heart's electrical activity may cause significant abnormalities in its mechanical function, and are the basis of much cardiac morbidity and mortality. In fact, malfunctiion of the heart's electrical behavior is the principal cause of sudden cardiac death. Arrhythmias are generally thought to occur on the basis of one or a combination of three potential mechanisms: automaticity, triggered activity, or re-entry [7].

2.3.1 Pathophysiology of Arrhythmias

Most evidence suggests that re-entry is the mechanism of clinical ventricular arrhythmias [17]. In the presence of slow conduction and/or unidirectional block, it is possible to establish in the myocardium a so-called "re-entrant circuits" of excitation, as shown in Figure 2-5. Some workers have subdivided the re-entrant mechanism into two catagories, "random re-entry" and "ordered re-entry". Random re-entry can cause atrial or ventricular fibrillation, whereas ordered re-entry can cause tachycardias. The distinction between the two is that during random re-entry, propagation occurs over re-entrant pathways that continuously change their size and location with time. Whereas an ordered re-entry implies a relatively fixed re-entrant pathway. In all configurations, however, The wavelength of the impulse in the reentrant circuit (conduction velocity X refractory period) must be shorter than the length of the circuit, so that the tissue into which the impulse is re-entering has had time to recover its excitability.

Re-entry may occur in patients who have bundles of surviving muscle fibers in healed myocardial infarcts. The interfusion of dead cells and diseased cells creates the anisotropic medium for the developement of re-entrant circuits. There are many possible geometric arrangements for such loops which may exist in many locations of

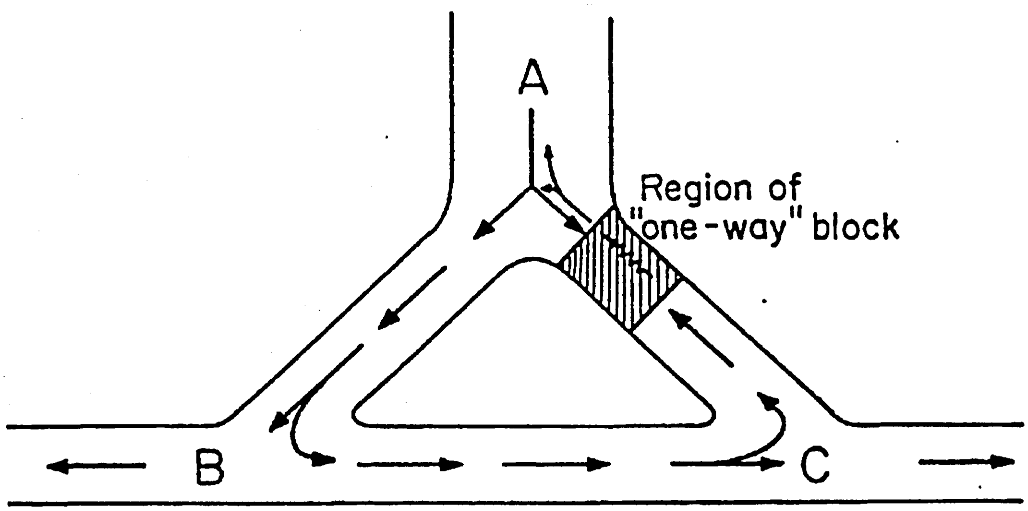

Figure 2-5: Reentrant Circuit (Reproduced from Smith: Ph.D. Thesis, 1985)

the heart and in many sizes, both microscopic and macroscopic. It had been proposed that the microscopic re-entrant circuits are the most common form [15].

2.3.2 Clinical Examples of Arrhythmias

As explained earlier, patients who had suffered myocardial infarction are susceptible to ventricular arrhythmias because of the presence of the re-entrant pathways in the surrounding areas of infarct. In this section, some of the clinical ventricular disrhythmias are reviewed.

Ventricular Premature Beats

The most common ventricular arrhythmia is the ventricular premature beats (VPB). A ventricular premature beats occur when ventricular activation is initiated as a site within the ventricles and not as a result of the normal chain of cardiac activation initiated in the sino-atrial (SA) node. Thus, the normal timing of events is disrupted with the result being inadequate time for filling of the ventricles, and consequently decreased ejection of blood into the ascending aorta. As shown in ECG in Figure 2-6, the P wave is absent and the QRS complex is quite wide (greater than 0.12 seconds) and bizzare. The width of QRS is enlarged due to the VPB not utilizing the His-Purkinje system with its rapid conduction velocity.

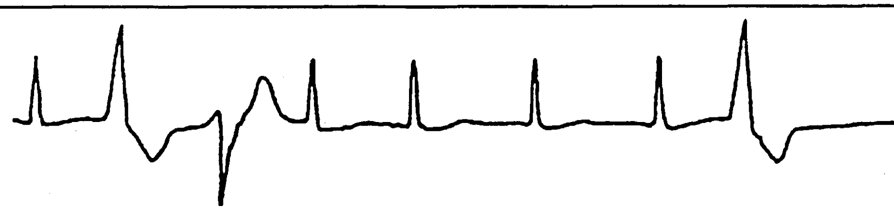

Figure 2-6: Ventricular premature beats (Reproduced from Smith: Ph.D. Thesis, 1985)

VPB's are often found in otherwise normal individuals and probably have little significance if they are infrequent. In heart diseases, VPB's may be a risk factor for increased incidence of more serious ventricular arrhythmias and sudden death. VPB's which fall on the T-wave of the previous beat are considered particularly dangerous. The period near the T-wave peak is oftern referred to as the "ventricular vulnerable period," as shown in Figure 2-7. At the time corresponding to the peak of the T-wave, the ventricular myocardium is just beginning to repolarize. Some cells may be in the relatively refractory period, while others may be more fully recovered, and still others quite refractory. The electrical properties of the myocardium are thus quite varied, and conditions favoring re-entrant loops are likely. Thus, an extra stimulus in the form of an isolated VPB which is very early-cycle may trigger a repetitive ventricular ectopic rhythm such as ventricular tachycardia or ventricular fibrillation [4].

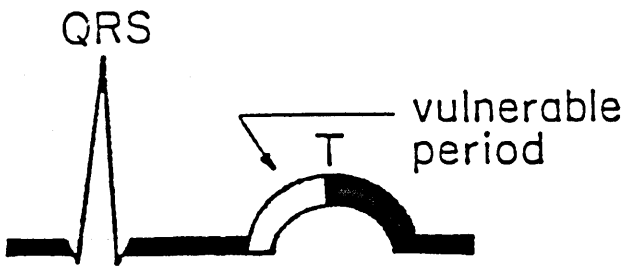

Figure 2-7: Ventricular ventricular period, the time during which ventricular arrhythmias can be induced (Reproduced from Smith: Ph.D. Thesis, 1985)

Ventricular Tachycardia

Another arrhythmia of ventricular origin is ventricular tachycardia, wherein rapid repetitive stimulation and contraction of the ventricles occur in the absence of normal atrially conducted stimulation. The rates of ventricular tachycardia vary from 150 to 300 beats per minute. Rates above 250 beats per minute fails to provide sufficient cardiac output to maintain life since the rapid rate does not permit sufficient filling of the ventricles [13]. As shown in ECG of Figure 2-8, a train of ventricular complexes similar to VPBs are present, with no coupled preceding atrial activity.

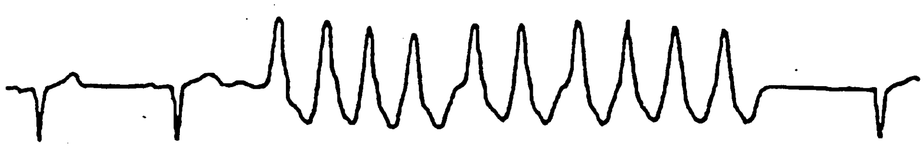

Figure 2-8: Ventricular tachycardia (Reproduced from Smith: Ph.D. Thesis, 1985)

Ventricular Fibrillaion

The last arrhythmia to be reviewed herein, and the one that is most often associated with sudden cardiac death, is ventricular fibrillation. Ventricular fibrillation is characterized by asynchronous, chaotic electrical activity resulting in disorganized and ineffectual contractions. Cardiac output during ventricular fibrillation is negligible, and as such the condition is incompatible with life. As shown in Figure 2-9, ventricular fibrillation is manifested by a random oscillation of potential, with no QRS complexes.

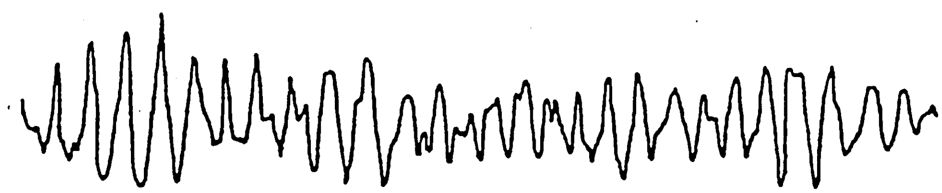

Figure 2-9: Ventricular fibrillation (Reproduced from Smith: Ph.D. Thesis, 1985)

Chapter 3

Programmable Stimulator System

A programmable stimulator system is an essential piece of equipment for the study of arrhythmias. In our studies of arrhythmias, there was a need for a device which produced more than the standard constant current or voltage S_1 pacing train. A device is needed to produce a wide variety of pacing protocols, such as induction of arrhythmias, and synchronize the delivery of pulses with the electrical activity of the heart. Also, since some of the pacing is done through a multi-site electrodes sock, the system has to provide voltage and current pulses for a moderate level of amplitude. In this chapter, a microcomputer-based stimulator system that can be programmed to produce a wide variety of pacing protocols is presented.

3.1 System Overview

A block diagram of the programmable stimulator system is shown in Figure 3-1. The programmable stimulator system consists of a 80386 microcomputer with a digital-to-analog board, a software program, called *Pacer*, a stimulator hardware, and monitoring/recording equipments. The stimulator hardware again conisits of three

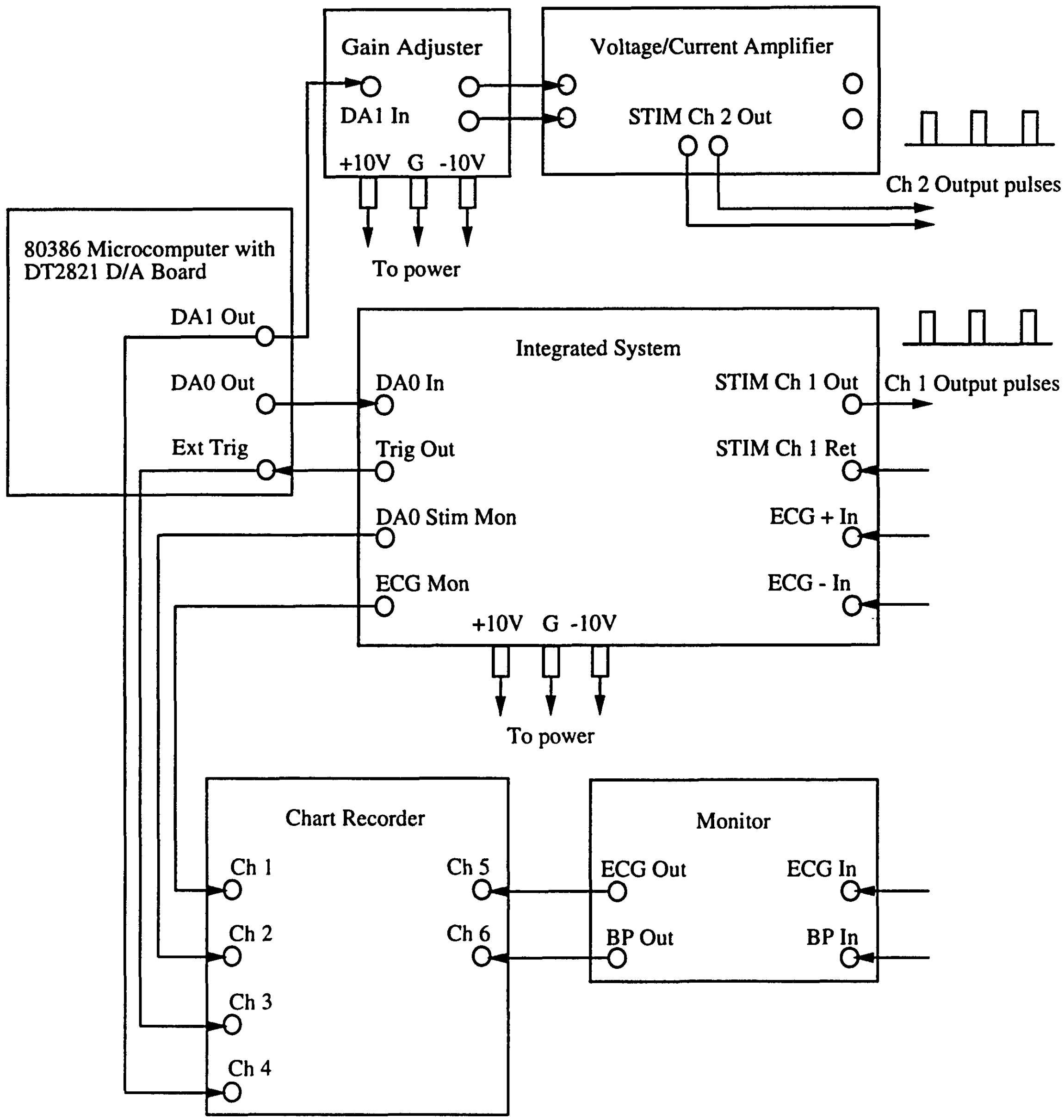

Figure 3-1: Diagram of the programmable stimulator system

sub-components: a) a integrated system, b) a gain adjuster, and c) a voltage/current amplifier.

Two analog pulses are generated by a fast digital-to-analog converter at a rate of 1,000 samples per second per channel (DA0 Out and DA1 Out in Figure 3-1). The amplitude and timing of the voltage pulses from a digital-to-analog converter are controlled precisely by a software and supplied to two inputs of the stimulator hardware (DA0 In and DA1 In in Figure 3-1).

Stimulator hardware converts two voltage inputs to two independent outputs for electrical stimulation. First channel sends current stimuli to a single electrode for pacing and inducing arrhythmias. These current stimuli are generated by the voltage-to-current converter of the integrated system (STIM Ch 1 Out in Figure 3-1). On the other hand, second channel sends either voltage or current stimuli to a multi-site electrodes sock. They are generated by the commercial high-performance voltage/current amplifier (STIM Ch 2 Out in Figure 3-1). These pacing stimuli can be applied at a fixed rate or be triggered by an external signal. These output pulses are used to stimulate the heart of an open-chest animal.

Following the stimulation, the electrical signal generated by the heart is recorded by the electrode placed on the epicardial surface of the heart (ECG $\pm$ In in Figure 3-1). This signal is conditioned by the electrocardiogram amplifier of the integrated system, and it is used to detect abnormal electrical activities of the the heart. When abnormal events are detected, a pulse is sent to the external trigger port of a digital-to-analog board to initiate another stimulation of the heart.

In order to monitor and record the animal experiments, all stimuli as well as the electrical activity of the heart can be displayed on the screen of an oscilloscope, and recorded on the analog tape and/or the Astro Med 8-channels chart recorder for hardcopy. In the following sections, various commercial components used in the programmable stimulatr system, as well as the integrated system, will be described.

3.2 System Components

Some of the components used in the development of the programmable stimulator system, such as a digital-to-analog board and a voltage/current amplifer, are commercially available equipments. In this section, some of their specification are given for reference.

3.2.1 Digital-to-Analog (D/A) Board

Data Translation DT 2821 board is used for the digital-to-analog conversion. This board plugs into any IBM compatible microcomputer and contains conversion circuitry for both analog-to-digital and digital-to-analog operations. When used as the digital-to-analog operation, this board produces two voltage outputs, ranging from -10 to 10 volts, with the resolution of 5 millivolts [10]. (Refer to the DT 2821 board manual for further information.)

3.2.2 Voltage/Current Amplifier

A Kepco BOP 72-6M bipolar operational power supply (BOP) is used as a precision voltage or current source for the stimulation through the multi-site electrodes sock. This BOP provides the flexibility to pace by voltage or current and a moderate amount of stimulation outputs for multi-site pacing. The output of BOP can be controlled by voltage signals. As a bipolar voltage amplifier, the BOP output ranges from -72 to 72 volts and has a gain factor of 7.2 (V/V). On the other hand, as a bipolar current amplifier, the BOP output ranges from -6 to 6 amp and has a gain factor of 0.6 (A/V). Therefore, a ± 10 volt input signal will program the BOP output as voltage or current through its rated output ranges. Also, built-in preamplifiers, for the voltage and current channels of the BOP, provide the interface with high as well as low impedance signal sources [19]. (Refer to the KEPCO BOP 72-6M power suppliy manual for further information.)

3.3 Integrated System

It should be noted that the integrated system, shown in Figure 3-1, is specifically developed for this study. It cosists of several functional units: a voltage-to-current converter, an electrocardiogram amplifier, and a detector. Special care has been taken to ensure safety of the integrated system by preventing electroshock hazards. By use of isolation amplifiers, the current pulses are separated from ground guarding against unwanted dangerous interference from the mains.

3.3.1 Current Amplifier

As mentioned previously, pacing and extra stimuli of channel one are current pulses. Since the D/A board generates voltage pulses only, a voltage-to-current converter is needed. The current amplifier, shown in Figure 3-2, consists of: a) a buffer for impedance matching, b) a gain control, c) an isolation amplifier, d) a RC circuit to prevent a DC shock, e) a voltage-to-current converter, and f) a current output monitor with a LED indicator.

RC High-Pass Filter to Prevent DC Shock

The voltage pulses of the D/A board are fed to an unit-gain buffer, a gain control, and an isolation amplifier. The output of the isolation amplifier is then fed into a simple RC circuit, which acts as a high-pass filer to cutoff DC voltage. This RC cirucit protects the experimental animal from the DC shock when DC is applied accidentally to the current amplifer. The RC circuit and the shape of pulses before and after the

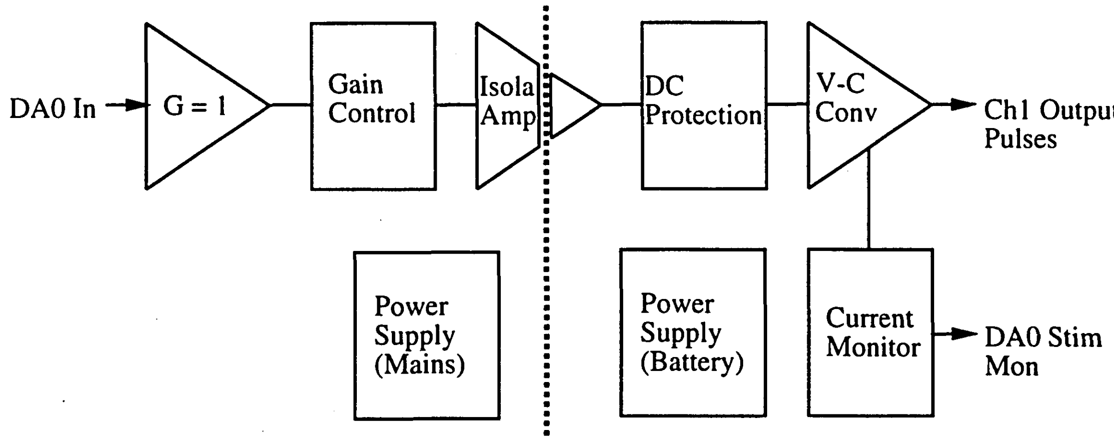

Figure 3-2: A block diagram of the current amplifier sub-system

RC circuit is shown in Figure 3-3. Note that the uniform shape of the inputs pulse is modified to the exponentially decaying pulse with the time constant of RC, in this case 10 msec.

Voltage-to-Current Converter

The ouput of the RC cirucit is then used to control the current source. The current source uses the xxx transistor . Assuming that the load resistance of the heart during the pacing would be less than 200 ohm, the output of the current source is linear upto about 30 mA.

Current Output Monitor

The output of the voltage-to-current converter is monitored by recording the voltage developed across 100 ohm resistor. The voltage is recorded on the chart recorder and this enables the investigator to confirm the actual current output delivered to the load at the time of pacing. Isolation amplifier is again used to protect animal

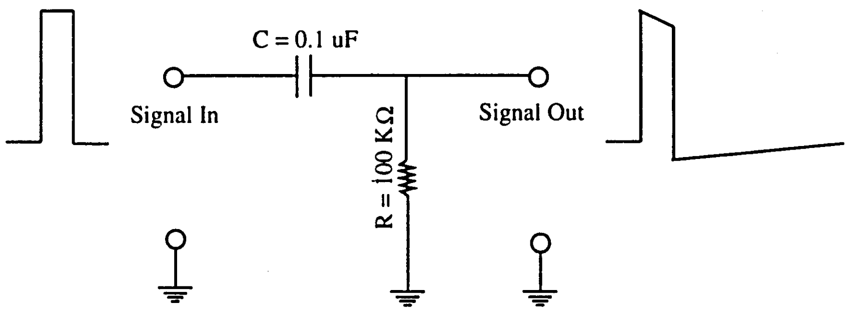

Figure 3-3: RC circuit used for the DC shock protection with pulse shapes before and after the RC cirucit

from fault-ground shock. Also, unity gain buffers are used for impedance matching. In addition, a LED display is added to blink at each pacing stimuli.

3.3.2 Intracardiac Electrocardiogram Amplifier

In order to pace the heart based on the electrical activity of the heart, an intracardiac electrocardigram amplifier is needed. This amplifier records the epicardial or intracavitary cardiac signals from the electrode and conditions it as an input the detector.

General Description

The intracardiac electrocardiogram (ECG) amplifier is used for the amplification of epicardial electrocardiogram, which has a small differential signal riding on a large common mode signal, signals that are common to all the electrodes. For example, if there were a millivolt electrocardiogram signal and a volt of common-mode signal at

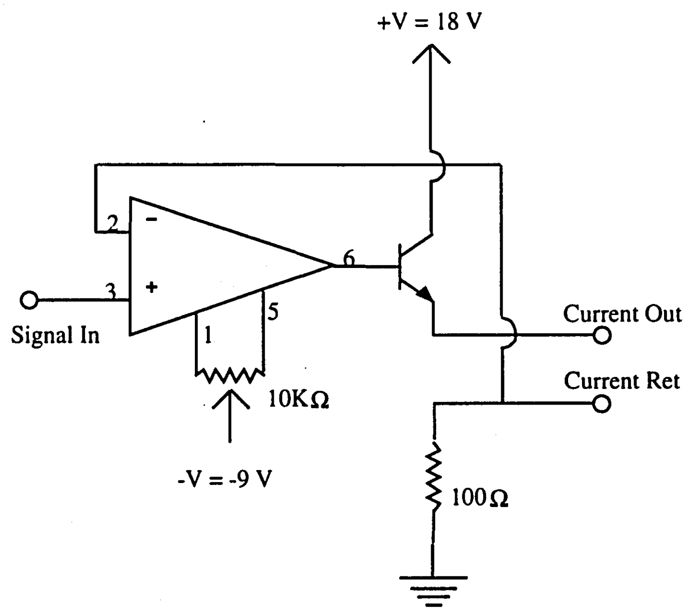

Figure 3-4: Circuit diagram of a voltage-to-current converter

the electrodes and transmission along the two wires differed by 0.1 %, then a millivolt of differential signal would be developed from the common-mode signal, which is as large as the electrocardiogram itself.

In order to solve these problems involving a common-mode signal, identical conditions have to be imposed on the two signal paths. Each must be exposed to the same noise, same source, and destination impedance. A high input-impedance differential-input amplifier is an ideal choice to solve these problems.

The intra-cardiac electrocardiogram amplifier, shown in Figure 3-5, is composed of six basic building blocks: a) an overload protection circuit with high pass filter, b) a direct-coupled, low gain preamplifier, c) an isolation amplifier, d) a high pass filter set at 10 Hz, e) a programmable gain amplifier, and f) a low-pass filter set at 70 Hz.

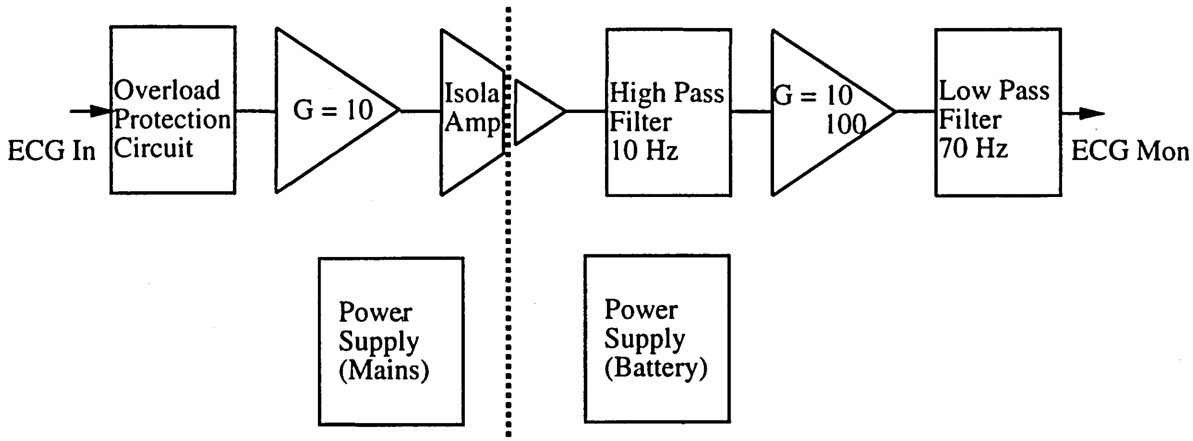

Figure 3-5: A block diagram of the intra-cardiac electrocardiogram amplifier subsystem

Overload Protection Circuit with a High-Pass Filter

An overload protection circuit, shown in Figure 3-6, prevents damage to the rest of the intracardiac electrocardiogram amplifier in case of cardioversion. The protection circuit clips the input signal when an excessive signal such as defibrillator discharge signal is applied to the input. In addition, the built-in high-pass filter rejects the DC (sub-signal) frequencies, especially the half cell potential of several ten or hundreds mV around the surface of electrode.

In the circuit diagram of Figure 3-6, resistors (R1, R2) and zenor diodes (D1, D2) work to protect the differential amplifier from the shock of the defibrillator. Capacitors (C1, C2) and resistors (R1, R2, and R3) compose the high pass filter which eliminates the DC voltage from the surface of electrodes. Note that R3 must be far greater than R1 and R2 to keep the input impedance of the amplifier high. Capacitor C3 works as a high cut filter together with R1, R2, C1, and C2. The characteristics of the filter is adjusted so that the frequency which composes the QRS

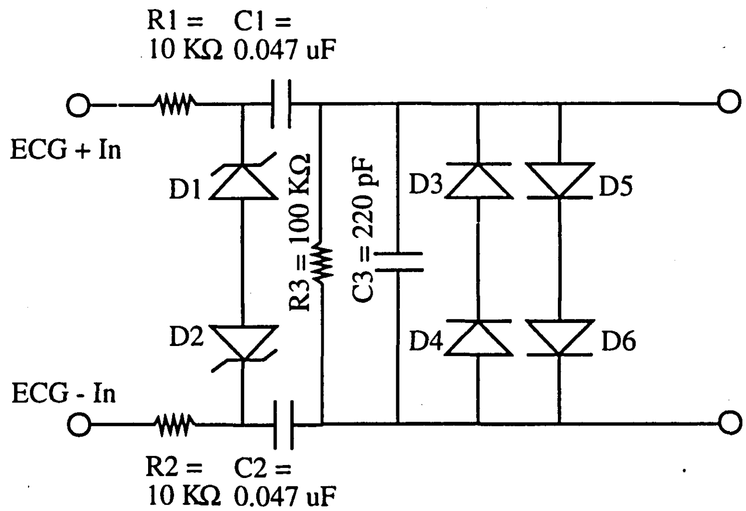

Figure 3-6: A circuit diagram of the overload protection circuit with a high-pass filter

complex, 20-50 Hz, can be entirely passed through. Finally, diodes (D3 - D6) clip the signal to 1.2 Volts.

Differential Amplifier

The direct-coupled, low-gain preamplifier, shown in Figure 3-7, is a high input impedance differential amplifier with single-ended output. The differential amplifier consists of two stages, a balanced input stage and a differential output stage. The balanced input stage is ideal for amplifying small (range of 10 mV) electrocardiogram signals since it provides: a) high input impednace to minimize differences in attenuation in the two signal paths due to unequal source impedances, b) an extremely good common-mode rejection ratio (CMRR) to amplify a small differential signal riding on a large DC offset. The balanced input stage's differential output represents a signal with substantial reduction in the comparative common-mode signal. Then, the output stage is connected in the standard differencing amplifier configuration. This configuration of a balanced input stage followed by a differential amplifier is called the three op-amp insturmentation amplifier.

CMRR and CMR of the differential amplifier are defined as follows:

$$\mathrm{CMRR} = \frac{\text{Output due to unit differential} - \text{mode signal}}{\text{Output due to unit common} - \text{mode signal}} \tag{3.1}$$

$$\mathrm{CMR} = 20\log_{10}\mathrm{CMRR}\ (\mathrm{dB}) \tag{3.2}$$

Also, the differential gain of the instrumentation amplifier in Figure 3-7 is defined as follows:

$$\frac{V_{out}}{V_{in}} = \frac{R_1 + R_2 + R_3}{R_2} \times \frac{R_4}{R_3} \tag{3.3}$$

This differential amplifier provides CMR of about 50 dB, which is acceptable CMR for electrocardiogram, and has a differential gain of 10.

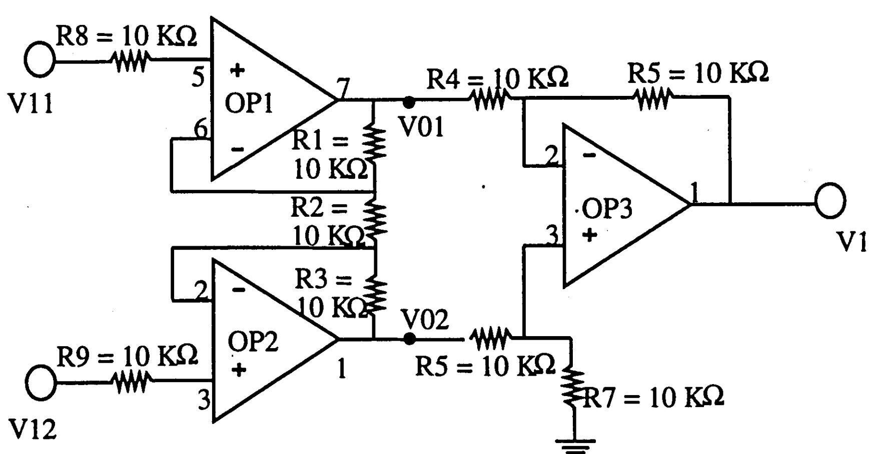

Figure 3-7: A circuit diagram of the differential amplifier

In the circuit diagram of Figure 3-7, a balanced input stage, which is composed of two operational amplifiers (OP1 and OP2) and three resistors (R1, R2, and R3), is implemented to improve CMRR. Its transfer function is characterized by equations 3.4 and 3.5. The differential gain A_{DD} and A_{DC} are about 10 and 0, respectively; the common-mode gain A_{CD} and A_{CC} are 1 and 0, respectively. Matching of R1 to R3 and R4 to R5 affect CMRR. Gain may be adjusted through R2.

$$V_{02} - V_{01} = A_{DD}(V_{12} - V_{11}) + A_{DC}(\frac{V_{11} + V_{12}}{2}) \tag{3.4}$$
$$\text{where } A_{DD} = \frac{R_1 + R_2 + R_3}{R_2} \text{ and } A_{DC} = 0$$

$$(\frac{V_{01} + V_{02}}{2}) = A_{CD}(V_{12} - V_{11}) + A_{CC}(\frac{V_{11} + V_{12}}{2}) \tag{3.5}$$
$$\text{where } A_{CD} = 1 \text{ and } A_{CC} = \frac{R_3 - R_1}{2\ R_2} = 0$$

A differential output stage is composed of one operation amplifier (OP3) and four resistors (R4, R5, R6, and R7). Its transfer function is characterized by equation 3.6. The differential gain A_D is an unity and the common-mode gain A_C is ideally 0 for a differential amplifier. However, CMRR is limited by the resistor matching of R5 to R7. To improve CMRR, high-precision resistors are used in building the circuit.

$$V_1 = A_D(V_{02} - V_{01}) + A_C(\frac{V_{01} + V_{02}}{2}) \tag{3.6}$$
$$\text{where } A_D = (\frac{R_5}{R_4}) \text{ and } A_C = 0$$

High Pass Filter

Since the spectrum of the electrocaridogram signal is distributed above 5 Hz, a high-pass filter is built with a cutoff frequency of 10 Hz. The Chebyshev filter with 2 dB ripple in the pass band is chosen for the high-pass filter. Although Chebyshev filter allows some ripples in the passband, it provides the fastest roll-off from passband to stopband. The Sallen-Key circuit with the unity-gain follower, shown in Figure 3-8, is chosen for the high-pass filter.

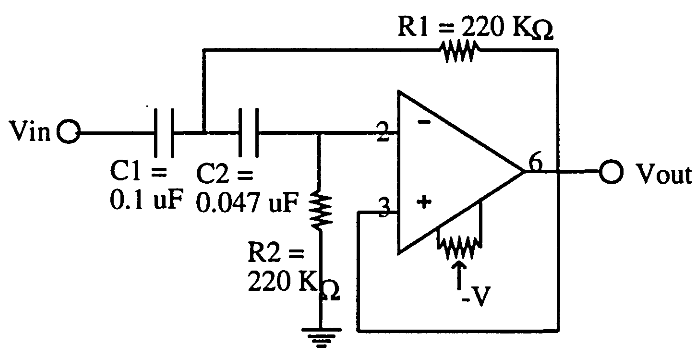

Figure 3-8: Circuit diagram of a 2nd-order Chebyshev high-pass filter with 2dB

Low-Pass Filter

The main purpose of the low pass filter is to reduce the noise due to the muscle movements. Since the electrocardiogram signal is distributed below 60 Hz, a low-pass filter is built with a cutoff frequency of 70 Hz. This low-pass filter and the high-pass filter make up a band-pass filter for the electrocardiogram amplifier.

The Bessel filter is chosen for the low-pass filter since it introduces the least distortions to electrocardiogram. The Bessel filter has a phase response which is linearly related to frequency. It also exhibits short settling time and rise time [1, p.

23]. Therefore, when pulses, such as ECG signals, are transmitted, the output pulses are of approximately the same shape as the input pulses. The Sallen-Key circuit with the unity-gain follower, shown in Figure 3-9, is chosen for the low-pass filter. The transfer function of this low-pass filter is shown below, and its frequency response is shown in Figure 3-10.

$$H(s) = \frac{1}{1 + s(R_2C_2 + R_1C_2) + s^2(R_1C_1R_2C_2)} \quad (3.7)$$

The output of the low-pass filter is the final output of the electrocardiogram amplifier. This signal is then supplied to the input of the detector.

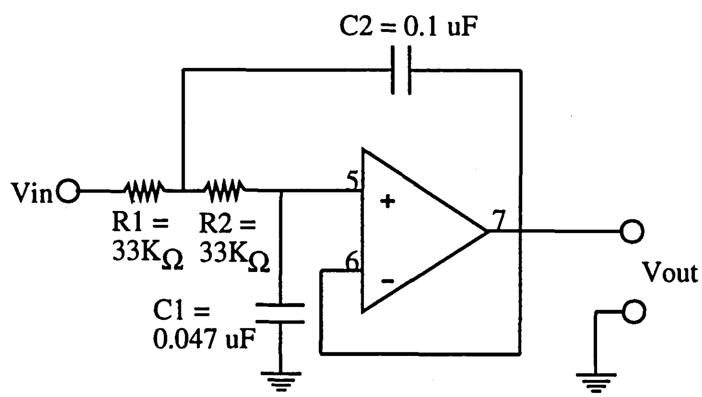

Figure 3-9: Circuit diagram of a 2nd-order Bessel low-pass filter

3.3.3 Detector

A hardware detector, built as a part of the integrated system, uses level detection to trigger a stimulation. This stage receives the filtered electrocardiogram signal from the amplifier and outputs a negative-going (from 10 volt to 0 volt) short duration pulse when the electrocardiogram signal goes above a normal range. This

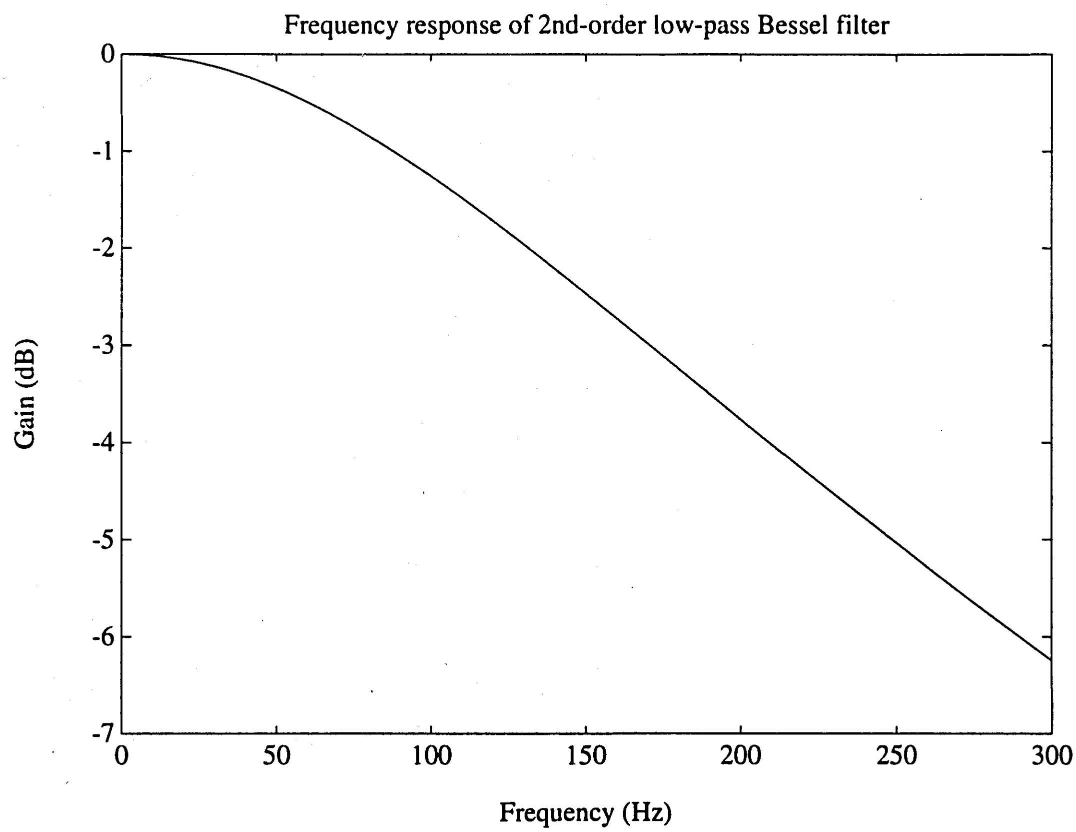

Figure 3-10: Frequency response of 2nd-order Bessel low-pass filter

negative-going pulse is provided to the external trigger of the digital-to-analog board for synchronizing the pacing with the abnormal events of the heart. The detector, shown in Figure 3-11 consists of: a) a full-wave rectifier, b) a voltage comparator, and c) a trigger generator with a trigger inhibit time.

Full-Wave Rectifier

A full-wave rectifier, shown in Figure 3-12, converts all negative values to positive values in order to insure detection of all abnormal events. This is necessary since the voltage detector, which follows in the signal path, cannot have two threshold values.

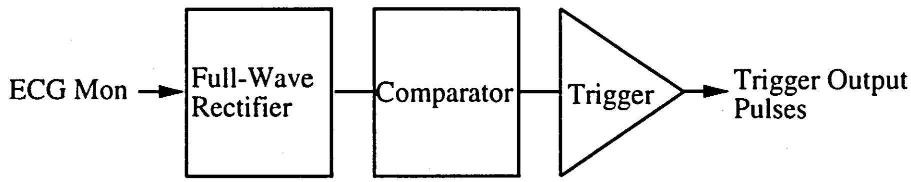

Figure 3-11: A block diagram of the detector sub-system

When the input is positive, diode D1 is off and diode D2 is on. In this case, the first op-amp (OP1) with first two resistors (R1 and R2) becomes an unity-gain inverter. The second op-amp (OP2) with next three resistors (R3, R4, and R5) make up an adder with inverting configuration. The output of the second op-amp should follow the input signal to the full-wave rectifier. On the other hand, when the input is negative, D1 is on and D2 is off. Then, the first op-amp's output is zero and the second op-amp becomes an unity-gain inverter. The output of the second op-amp should be the opposite of the input signal.

Voltage Comparator

The static transfer characteristics of a voltage comparator are shown in Figure 3-13. Voltage comparator has two output levels, V_L and V_H. When the input voltage to the comparator goes above the the threshold value, V_{TH}, the output goes from V_L to V_H. Note that there is a range of input signals which produce an ambiguous output, called the linear range. It is most optimal to have a minimum linear range to decrease ambiguity. The low and high output levels, V_L and V_H, are specified by the type of logic circuitry which is to be actuated by the detector. For the digital-to-analog board, it's the TTL (transistor-transistor-logic) levels, in which less than 0.4 volts is considered logically low, and greater than 2.6 volts is considered high. The threshold voltage, V_{TH}, is controlled by the potentiometer. This voltage comparator

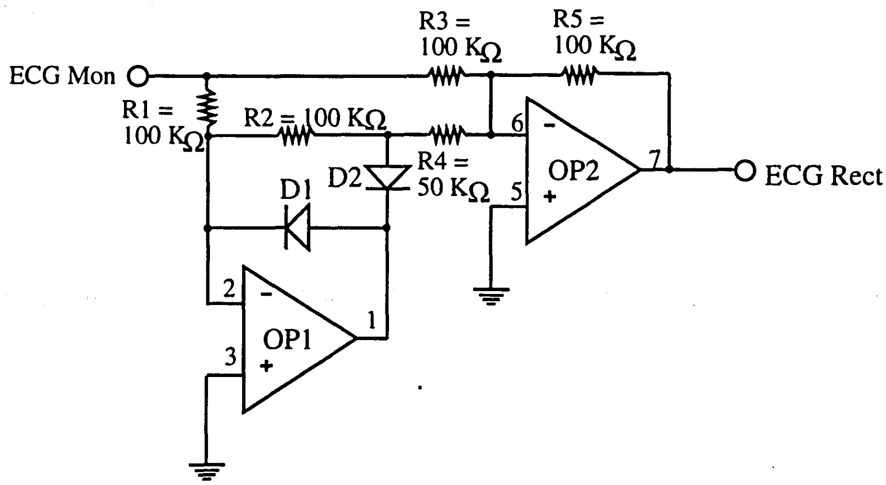

Figure 3-12: Circuit diagram of a full-wave rectifier

is realized using LM311, an integrated cirucit comparator.

Trigger Generator

The output of the comparator is high whenever the input level is higher than the threshold level. Since the digital-to-analog board's output is triggered when the external trigger signal is low, a separate trigger ciruit is needed to generate the short negative pulse to the digital-to-analog board. Also, it is desirable not to trigger again for certain time after the first trigger in order to prevent pacing during the refractory time. The trigger generator with the sensing inhibit time, implemented with the retriggable/resettable monostable vibrator, is shown in Figure 3-14 .

In the circuit diagram of Figure 3-14, the output of the voltage comparator is connected to the positive input of the first monostable vibrator (MV1). When the input to the positive input of MV1 goes from low to high, it triggers MV1 to output a negative-going pulse of two milliseconds width. This negative-going pulse is sent to the external trigger port of the digital-to-analog board and it is used to trigger a pacing.

In order to prevent from triggering during heart's refractory time, a sensing inhibit time generator is implemented using the second monostable vibrator (MV2). This generator disables the trigger generator for approximately 70 milliseconds after a trigger signal is sent out. The width of the sensing inhibit time is determined by a resistor (R2) and a capacitor (C2).

Ω

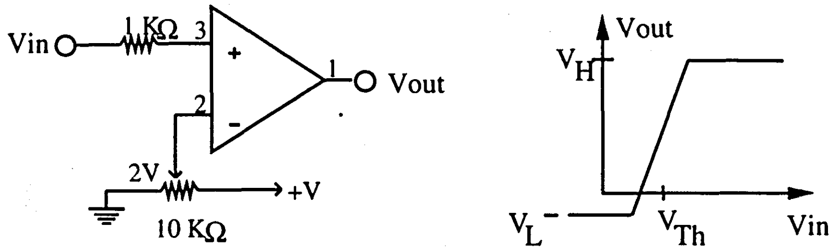

Figure 3-13: Transfer characteristics of a voltage comparator (Note that the output of the comparator is high wheneve the input level is higher than the threshold level.)

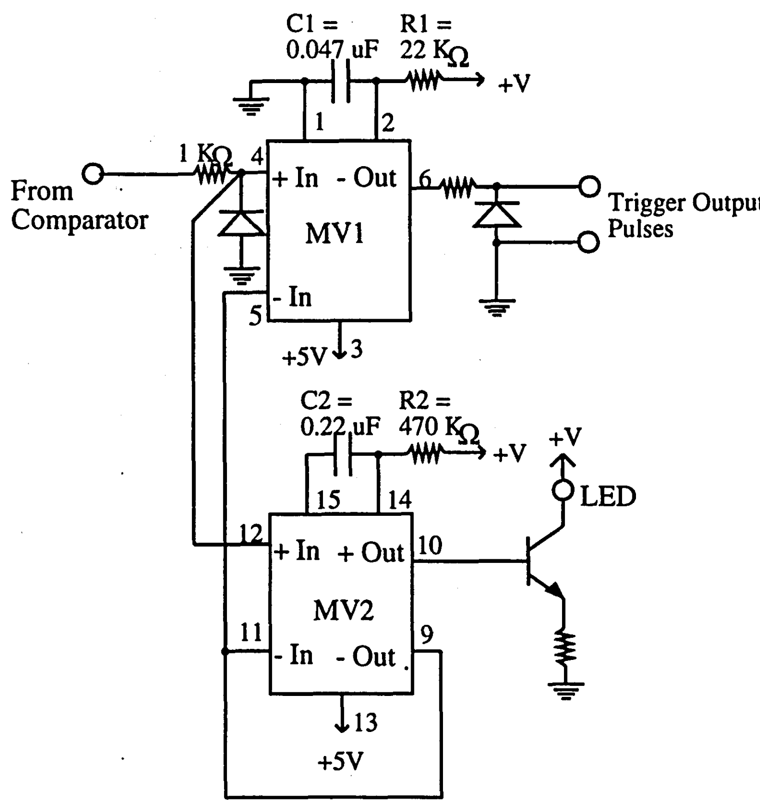

Figure 3-14: Circuit diagram of a trigger generator

Chapter 4

Control Program

The primary function of the control program, *Pacer*, is to provide an easy to use interface for the operator while accurately controlling the output of hardware on the digital-to-analog board. The combination of hardware and software allows for the implementation of many useful features. Preprogrammed stimulation modes are used to limit the number of parameters that the investigator must enter. In addition to the unique modes, there are several functions which can be enabled regardless of the currently selected mode, such as the stoppage of ventricular tachycardia. There is also the ability to receive an trigger signal from an external device, such as the integrated system described in the previous chapter. This trigger signal can be used for synchronous stimulation. *Pacer* is written in C using Borland's Turbo C development environment.

4.1 Program Overview

A flowchart of the control program, *Pacer*, is shown in Figure 4-1. All of the features of the program can be accessed from the menu in the command box. The

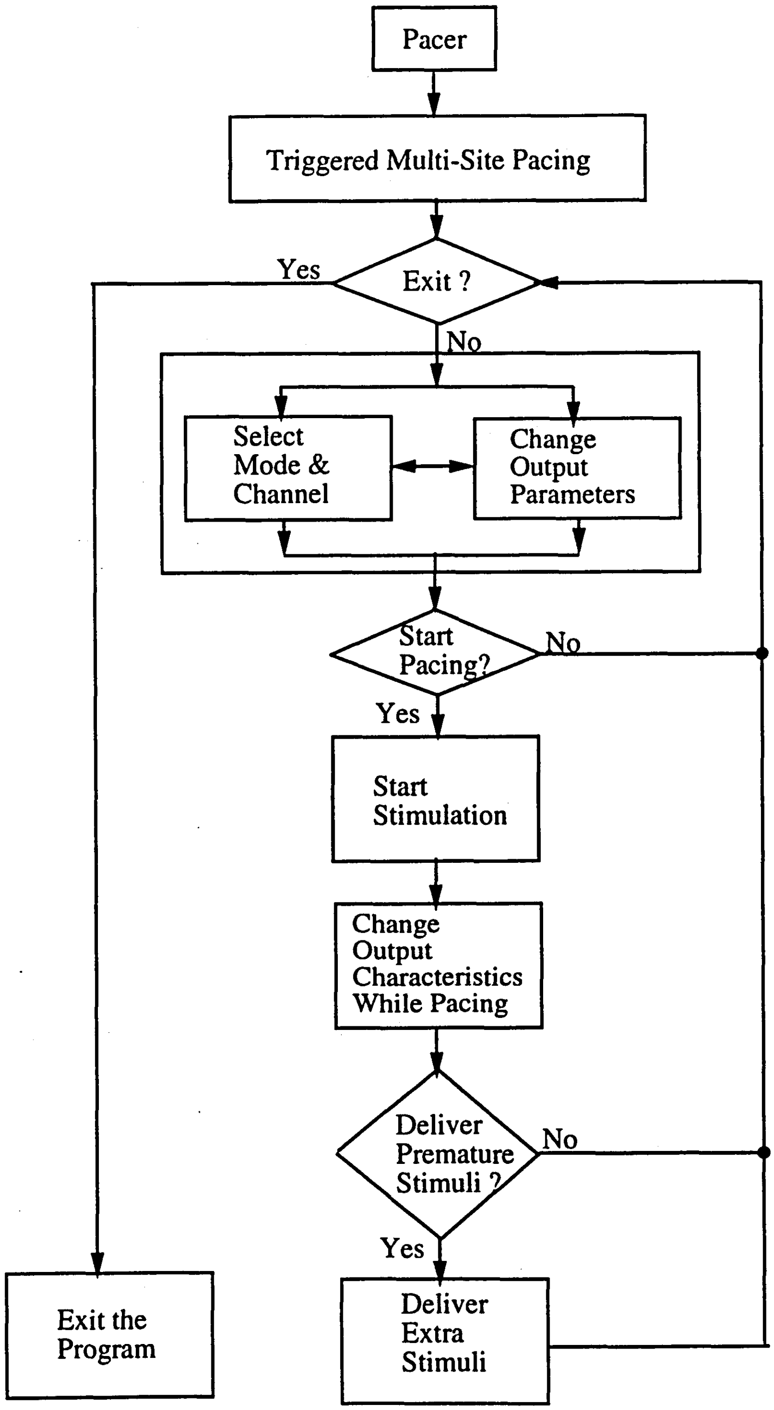

Figure 4-1: Flowchart of the control program

keyboard key may be used to select items from the menu, and *Pacer* automatically initiates the desired task. The control program, *Pacer*, can be divided into two main parts: a) the user interface where stimulation protocol parameters are selected by the investigator, and b) the output controller where actual data buffers are written for the outputs of the digital-to-analog board. This separation of two tasks ease the change of stimulation protocols for a constantly evolving investigation.

4.2 User Interface

Pacer is especially designed to meet the specific needs of investigators in venrticular arrhythmias study. In this section, it will be explained how parameters of different stimulation protocols in the study of ventricular arrhythmias are inputted by the investigator.

4.2.1 Pacing Threshold

For a fixed-rate pacing, a constant current S_1 pacing train is used, as shown in Figure 4-2. Output parameters, such as the S_1 pulse amplitude, S_1 pulse duration, and interstimulus interval (S_1 - S_1), of pacing pulses can be adjusted from default values by using cursor keys or new parameter values can be typed in using keyboard keys. Once these parameters are set, a series of fixed-rate pacing pulses is generated on the command. While stimulation is active, the pulse amplitude can be in-/decremented by the pre-set incremental value using the cursor keys in order to determine the threshold level. On the key pad, number 7 key and number 1 key are used for an increment and decrement of the pacing pulse amplitude, respectively. The interstimulus interval can also be in-/decremented by the cursor keys while stimulation is active. Number 6 key and number 4 key are used for an increment and decrement of the interstimulus interval, respectively.

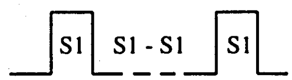

Figure 4-2: Fixed-rate pacing to determine the pacing threshold

4.2.2 Mid-Diastolic Threshold

As discussed in chapter 2, there is a ventricular vulnerable period roughly coinciding in time with the occurrence of the first half of the T wave in the electrocardiogram, as shown in Figure 2-7. Since the ventricles are susceptible to re-entrant rhythm disturbances during the vulnerable period, a mid-diastolic (premature) pacing pulse (S_2) can be used to induce ventricular arrhythmias. S_2 pulse is applied in late diastole of the cardiac cycle, as shown in Figure 4-3. Again, output parameters, such as the S_2 pulse amplitude, S_2 pulse duration, and initial S_1 - S_2 interval (D_1), can be adjusted. After the fixed-rate pacing is initiated, the mid-diastolic pacing pulse (S_2) can be applied on the command to determine if a beat is induced by the mid-diastolic pacing pulse (S_2). The pulse amplitude can be in-/decremented by the pre-set incremental values using the cursor keys and the stimulation can be repeated until the mid-diastolic threshold level is determined. On the key pad, number 9 key and number 3 key are used for an increment and decrement of the pulse amplitude, respectively.

4.2.3 Effective Refractory Period

After the mid-diastolic threshold is determined, the effective refractory period (ERP) is found. During the effective refractory period, there can be no propagated response to the stimuli. The initial S_1 - S_2 interval is decremented until no beat is induced by the mid-diastolic pacing pulse (S_2) and the S_1 - S_2 interval is noted as the effective refractory period.

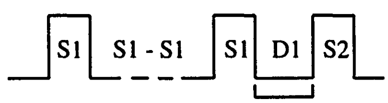

Figure 4-3: Mid-diastole pacing to determine the effective refractory period

4.2.4 Extra Stimuli for Arrhythmias Induction

Since more than one mid-diastolic pacing pulses are usually needed to induce ventricular arrhythmias, another mid-diastolic pacing pulse (S_3) is added in the same manner as S_2 was added. Again, S_3 pulse amplitude, S_3 pulse duration, and initial S_2 - S_3 interval, are adjustable parameters. After the fixed-rate pacing is initiated, the mid-diastolic pacing pulses (S_2 and S_3) are applied on the command to determine if a beat is induced by S_2 and S_3. The S_2 - S_3 interval, D_2, is decremented until no beat is induced by S_3. The effective refractory period of S_2 is noted. Then, same procedure is applied for the determination of effective refractory periods for S_3 and S_4. These premature stimuli delivered during consecutive ventricular vulnerable period are referred to as *extra stimuli*.

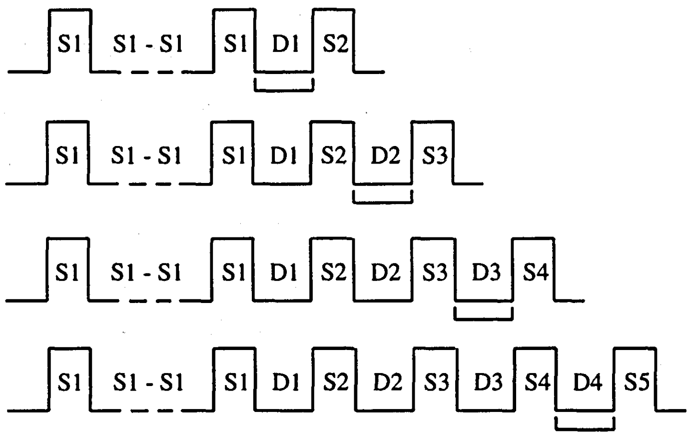

Figure 4-4: Extra stimuli used to induce ventricular arrhythmias

4.2.5 Burst for Arrhythmias Induction

In addition to extra stimuli, burst is another type of stimulation pulses known to induce ventricular arrhythmias. Burst is a group of closely placed current pulses in one vulnerable period (approximately 100 milliseconds), as shown in Figure 4-5. Output parameters, such as S_2 pulse amplitude, S_2 duration, length of vulnerable period, and frequency of pulses ($\frac{1}{\Delta T}$), can be adjusted for the burst.

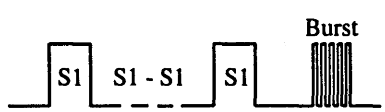

Figure 4-5: Burst used to induce ventricular arrhythmias

4.2.6 Overdrive Pacing for Arrhythmias Prevention

Overdrive stimuli refer to the stimuli which are applied after the delivery of premature stimuli in order to prevent the development of ventricular arrhythmias, as shown in Figure 4-6. Ventricular arrhythmias are caused by the occurrence of re-entrant circuit, and they can be terminated if re-entrant circuit can be interrupted by the depolarization of the ventricular mass.

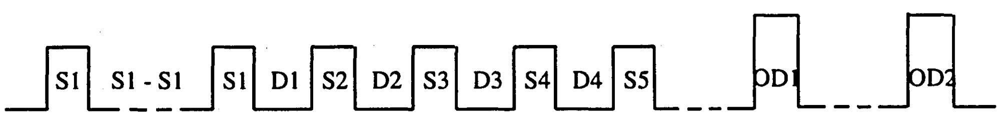

Figure 4-6: Overdrive pacing pulses used for the prevention of the ventricular arrhythmias

Either constant current or voltage pacing train can be used for overdrive stimuli. In addition to the usual output parameters (i.e. the pulse amplitude, pulse duration, and interstimulus interval of overdrive pacing pulses), the number of overdrive stimuli and triggering mode can be adjusted. The overdive pacing stimuli can be continued infinitely or specified in any number of overdrive stimuli.

Triggering mode can be either on or off. When triggering mode is off, the overdrive pacing stimuli are applied at a fixed interval after the last premature stimuli. On the other hand, if triggering mode is on, it allows the overdrive pacing stimuli to be applied synchronously with the external event, such as the ventricular electrical activity. When the triggering mode is on, an external trigger port of the digital-to-analog board is sampled to detect the trigger signal. Triggering capability allows the pacing to be applied when premature ventricular beats are detected after the last premature stimulus.

When used with a triggering mode on, a pacing inhibit time must be specified.

A pacing inhibit time refers to the period after the last premature stimulus, during which detection is disabled in order to prevent being triggered by the artifacts, or transients, of the previous stimuli. The pacing inhibit time is implemented using a computer board's internal clock and it can be controlled with the one millisecond resolution.

4.2.7 Stimulation Modes

To reduce the number of parameters that must be entered for an electrophysiologic study (EPS) of ventricular arrhythmias, individual stimulation modes for this research have been preprogrammed. Currently, six possible stimulation modes are available. They differ in the types of premature stimuli and overdrive stimuli S_2. Different stimulation modes are tabulated in Table 4.1. When each stimulation mode is selected, appropriate premature stimuli and overdrive triggering mode are selected and default parameter values are assigned to the output parameters.

	Extra Stimuli	Burst
No Overdrive Pacing	Mode 1	Mode 4
Fixed Rate Overdrive Pacing	Mode 2	Mode 5
Triggered Overdrive Pacing	Mode 3	Mode 6

Table 4.1: Stimulation Modes

4.3 Output Controller

When the investigator initiates a stimulation run, the output controller portion of the *Pacer* program is entered. The parameters that were setup in the user interface are now used to write output buffers for the registers of the digital-to-analog board. Special board driver program, ATLAB, is used to control the outputs of the digital-to-analog board.

4.3.1 Initialization and Termination

The initialization subroutine AL_INITIALIZE must be called before calling any other subroutine. AL_SELECT_BOARD and AL_RESET are used to reset the currently selected unit. In addition, a first argument of AL_SETUP_DAC specifies an internal clock to be the timing source for D/A conversions and a second argument specifies that two channels are used for writing the data values, one to each channel simultaneously. AL_SET_PERIOD initializes the clock period that affects the conversion rate for D/A operations to one millisecond.

ATLAB provides the termination subroutine AL_TERMINATE which performs the functions of termination. AL_TERMINATE must be called before exiting the program to terminate the DMA operation.

4.3.2 D/A Operation

Fixed-rate pacing can be started by running a PACING function. When PACING function is called in the program, user data buffers are declared by AL_DECLARE_BUFFER and linked to the the digital-to-analog board's Buffer Transfer List by AL_LINK_BUFFER. After a buffer is declared and linked, output buffer is now filled with data based on the parameter values inputted by the investigator. The output buffer contains a twelve bit value which corresponds to the voltage required by the stimulator system to produce the desired current and voltage outputs. The analog output board can produce 4096 steps over a -10 V to +10 V range: i.e., a digital value 0 refers to output of -10 V, 2048 refers to output of 0 V, and 4096 refers to output of 10 V.

After the buffer needed for the stimulation has been written, AL_BURST_DAC is used to produce analog outputs. AL_BURST_DAC allows high speed data transfer using the Direct Memory Access (DMA) technique. AL_WAIT_FOR_COMPLETION function synchronizes user program operation with burst mode I/O operation by returning the control of the program when I/O operation is complete. AL_UNLINK_BUFFER is called to unlink the buffer from the current unit's Buffer Transfer List after the stimulation. AL_UNDECLARE_BUFFER is used to undeclare the buffer.

Arrhythmias induction and prventive overdrive pacing can be started by using a combination of the following functions: EXT_STM, BURST, OVERDRIVE, and TRIGGER. For example, for mode 2 stimulation, functions EXT_STM and OVERDRIVE are called; for mode 3 stimulation, functions EXT_STM and TRIGGER are used. All these functions are written in the same structure as the PACING function.

The actual program, called pacer.c, is attached in appendix. The program is written with comments for future modifications. Also, the ATLAB manual can be consulted for more information on the ATLAB functions used in the program [9].

Chapter 5

Experimental Study

The programmable stimulator system presented in this thesis is tested in the experimental animal study. Ventricular arrhythmias are induced in a small swine, which has a similar heart size and coronary circulation as humans, and triggered pacing is administered to prevent the arrhythmias from developing. There are two days of surgery involved in this experimental study. The first surgery is a survival surgery, during which myocardial infarction is induced by the occlusion of the coronary arteries. The second surgery is an acute procedure for the induction of ventricular arrhythmias and study of triggered multi-site pacing for the prevention of ventricular arrhythmias. (Protocol # 92-005, approved by MIT Committee on Animal Care)

5.1 Survival Surgery

Using aseptic techniques, a pig of female sex (20 $\pm$ 1 kg) is tranquilized with a mixture of ketamine (10 mg/kg) and xylazine (2 mg/kg) intramuscularly. Anesthesia is maintained using isoflurane gas by the vaporizer through the endocardial tude (size 6.5) placed in the trachea. A left thoracotomy at the fourth intercostal space is performed and the paracardium is dissected open.

Myocardial infarction is induced by ligating the second and third diagonal branches from the left anterior descending (LAD) coronary artery, permanently obstructing blood flow. Thirty minutes prior to occlusion, a bolus of lidocaine (1.5 mg/kg) is administered intravenously. The infarct covers approximately 15 - 20 % of the left ventricular myocardium. The pictorial representation of the myocardial infarct is shown in Figure 5-1.

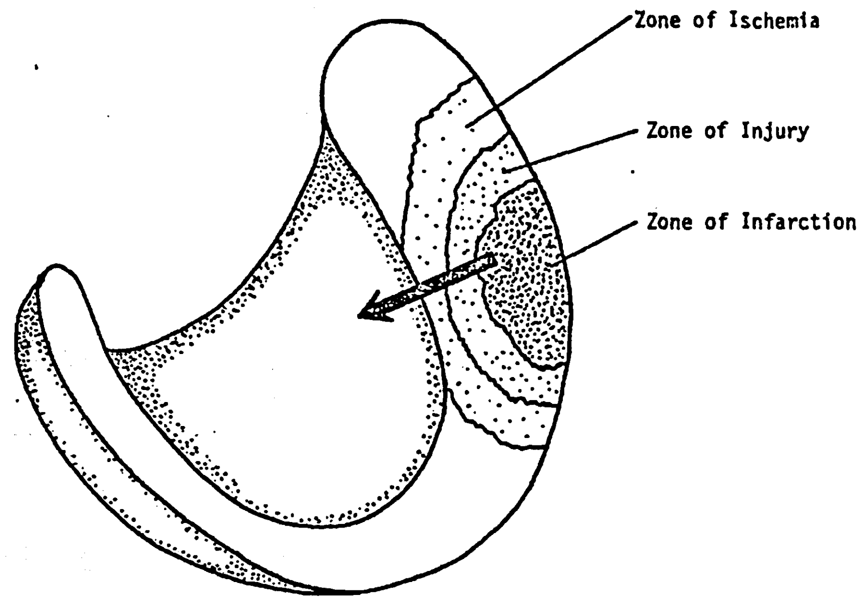

Figure 5-1: Myocardial infarct (Reproduced from Netter FH: Heart, Vol. 5, The CIBA Collection of Medical Illustrations, CIBA, 1981)

Tygon 16-gauge catheter (Norton Plastics, Akron, Ohio) is implanted in the left azygous vein going into the left atrium for post-operational care (see Figure 5-2 for the anatomy of a pig heart). The catheter is externalized on the back of the animal between the shoulder blades. Following the surgery, animal is treated with prophylactic antibiotics (kefflin) and is allowed to recover for seven days before acute surgery.

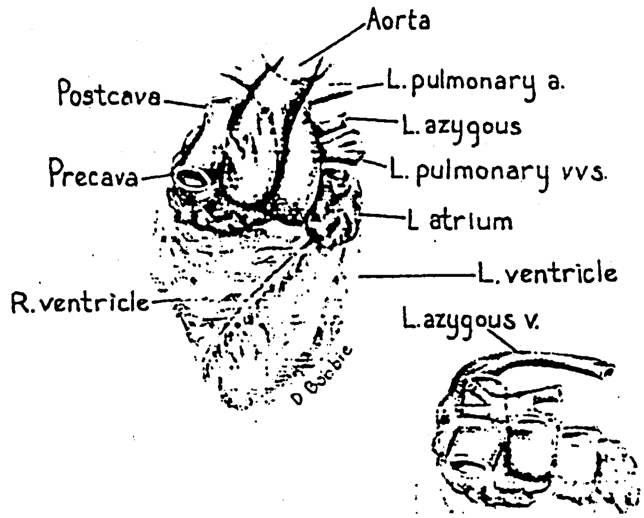

Figure 5-2: Cranial view of the swine heart showing the relationships of the blood vessels at the base of the heart. In the bottom view, the great vessels have been severed to reveal the left azygos vein.

5.2 Acute Surgery

5.2.1 Surgical Preparation

The animals is tranquilized (see last section) and anesthized with sodium pentabarbital (30 mg/kg) intravenously. Tygon 16-gauge catheter is implanted in the femoral vein percutaneously for infusion of 5 % dextrose ringer solution. And, another Tygon 16-gauge catheter is implanted in the femoral artery percutaneously to record the blood pressure. Phasic arterial blood pressure is measured by a pressure transducer connected to the catheter. Lead I electrocardiogram (ECG) is obtained through surface electrode needles. A sterotomy is performed and the heart is exposed.

One catheter with four unipolar pacing electrodes is passed through a jagular vein and positioned in the apex of the right ventricle, as shown in Figure 5-3. Two distal pacing electrodes are used to deliver current pulses for the single-site pacing and the induction of ventricular arrhythmias. Two proximal pacing electrodes are used for recording the intracavitary electrocardogram. In addition, 56-bipolar electrodes sock array is placed around the ventricle for multi-site pacing of the ventricle, as shown in Figure 5-4. The 56-bipolar electrodes sock array of Bard Electrophysiology has the dimensions of 15 centimeters in perimeter and 8.8 centimeters in length.

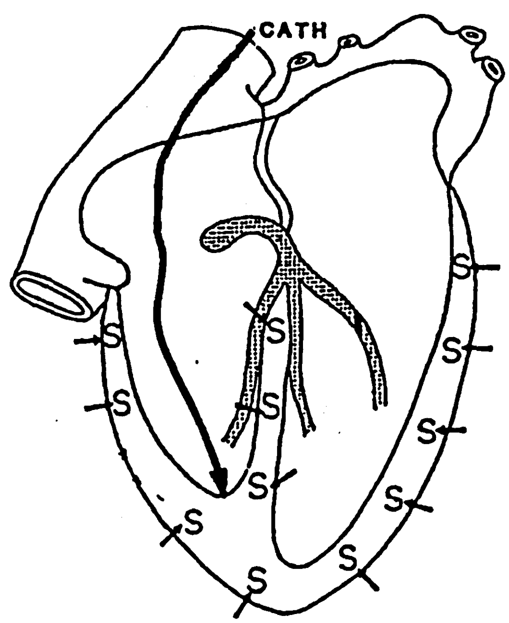

Figure 5-3: Intracavitary lead system used for the experiment

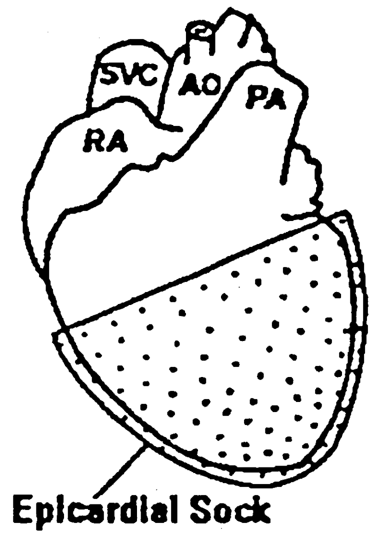

Figure 5-4: 56-bipolar electrodes sock placed around the ventricle

5.2.2 Electrophysiologic Study

Electrophysiologic study is accomplished by the programmable stimulator system presented in this thesis. In the following sections, the electrophysiologic study conducted for the prevention of arrhythmias is presented.

Determination of Single-Site Pacing Threshold

During electrophysiologic study, heart is paced by the distal electrodes of the catheter. The distal pole of the pacing bipolar catheter is made cathodal with reference to the proximal pole. Heart rate is maintained at 150 beats/min using constant current rectangular stimuli of two milliseconds duration. The rate is chosen to be 10 % higher than the intrinsic rate of 135 beats/min. Also, the pulse duration is chosen to be two milliseconds since it is the optimal duration based on the study of the inverse relationship between the pulse duration and pacing threshold [3]. While pacing is being performed at the lowest current amplitude possible, the current amplitude is increased at 1 milliamp interval until a ventricular beat is induced. The pacing threshold is determined to be 1 milliamp. The current amplitude of the pacing is then set at four times the value of the pacing threshold, 4 milliamp. The lead I electrocardiogram recorded during the ventricular pacing is shown in Figure 5-5 with stimulator outputs and arterial blood pressure.

Induction of Ventricular Tachycardia

Extra stimuli are used to induce ventricular arrhythmias in this experiment. While maintaining a 150 beats/min pacing rate at 4 times pacing threshold, a 2 miliseconds square wave stimulus (S_2), producing the first ventricular premature beat, is interposed in late diastole and is moved closer to the T wave of the previous beat in 20 milliseconds intervals until the effective refractory period is reached. S_2 is then set and presented at an interval that is 10 milliseconds longer than the effective refractory period, and a second stimulus (S_3) is added in the same manner. The $S_2 - S_3$ interval is shortened progressively in the same fashion until S_3 fails to evoke a propagated response. A third (S_4) and fourth (S_5) are added similarly until the desired end point is reached. Ventricular fibrillation is induced when the fourth extra stimulus is added. The animal is defibrillated by 10 Joule energy delivered by internal defibrillation paddles. The characteristics of extra stimuli needed to induce ventricular fibrillation is noted. The lead I electrocardiogram during ventricular fibrillation is shown in Figure 5-6 with the stimulator outputs and arterial blood pressure. This protocol is similar to one previously described and utilized in clinical electrophysiologic induction of ventricular arrhythmias [20].

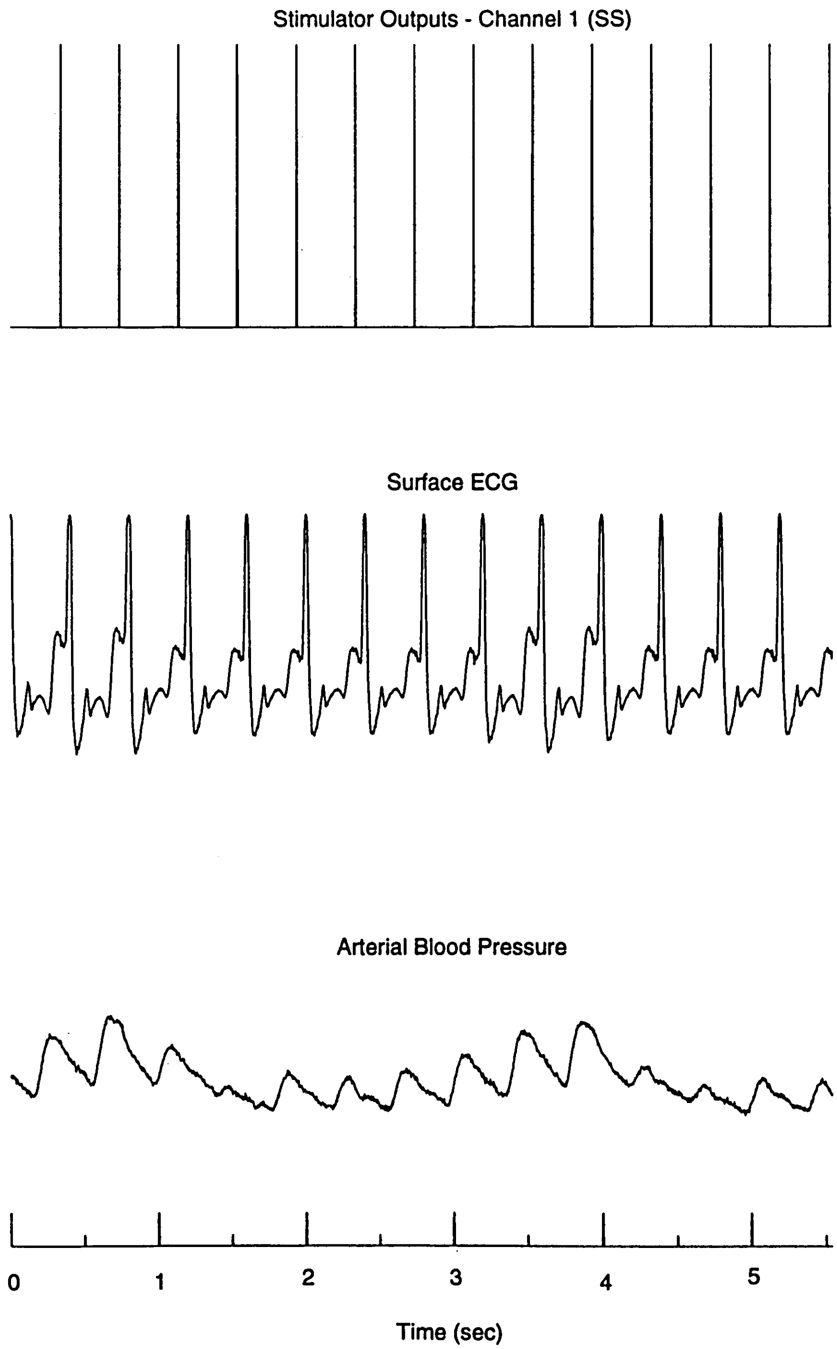

Figure 5-5: Electrocardiogram and arterial blood pressure during single site pacing using current pulses

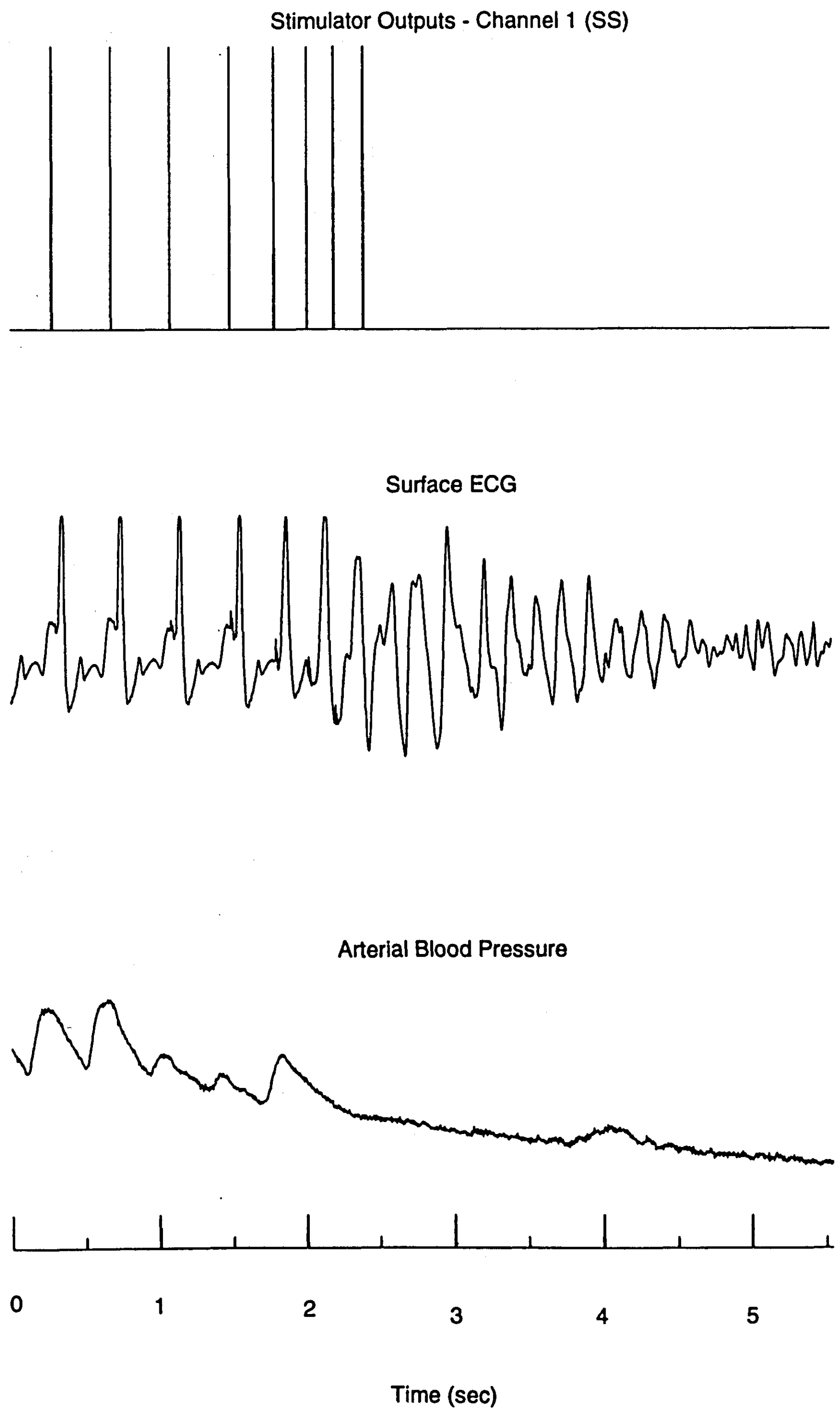

Figure 5-6: Ventricular fibrillation induced by four extra stimuli

Triggered pacing

The extra stimuli of same characteristics is used to induce another ventricularfibrillation, and triggered mulit-site pacing is applied through the mutli-site electrodes sock. The multi-site pacing is applied when two proximal pacing electrodes detect an activity in the right ventricle near the apex. Since the premature activities can be a part of re-entrant circuit, multi-site pacing is applied to prevent arrhythmias by deploarizing the ventricular mass. The multi-site pacing is not applied during the pacing inhibit time, 200 milliseconds period right after the previous pacing beat, in order not to trigger pacing by the transient of the pacing pulse. The lead I electrocardiogram, one right intracavitary electrogram, stimulator outputs, and blood pressure are shown in Figure 5-7.

Mutli-site pacing in this experiment did not prevent the development of ventricular fibrillation. The most likely reason for this is that the initial re-entrant site was a very small region and that the multi-site pacing electrodes were not sufficiently close to block the development of re-entry.

5.3 Future Study

To further investigate the preventive methods, the ventricular myocardium where the arrhythmia originate must be depolarized effectively. Other multi-site electrode pacing configurations to achieve this is currently being investigated. The high precision voltage/current amplifier of the current stimlator system can be used for this type of pacing. Also in order to pace only the premature beats, more efficient timing scheme can be programmed into the control program.

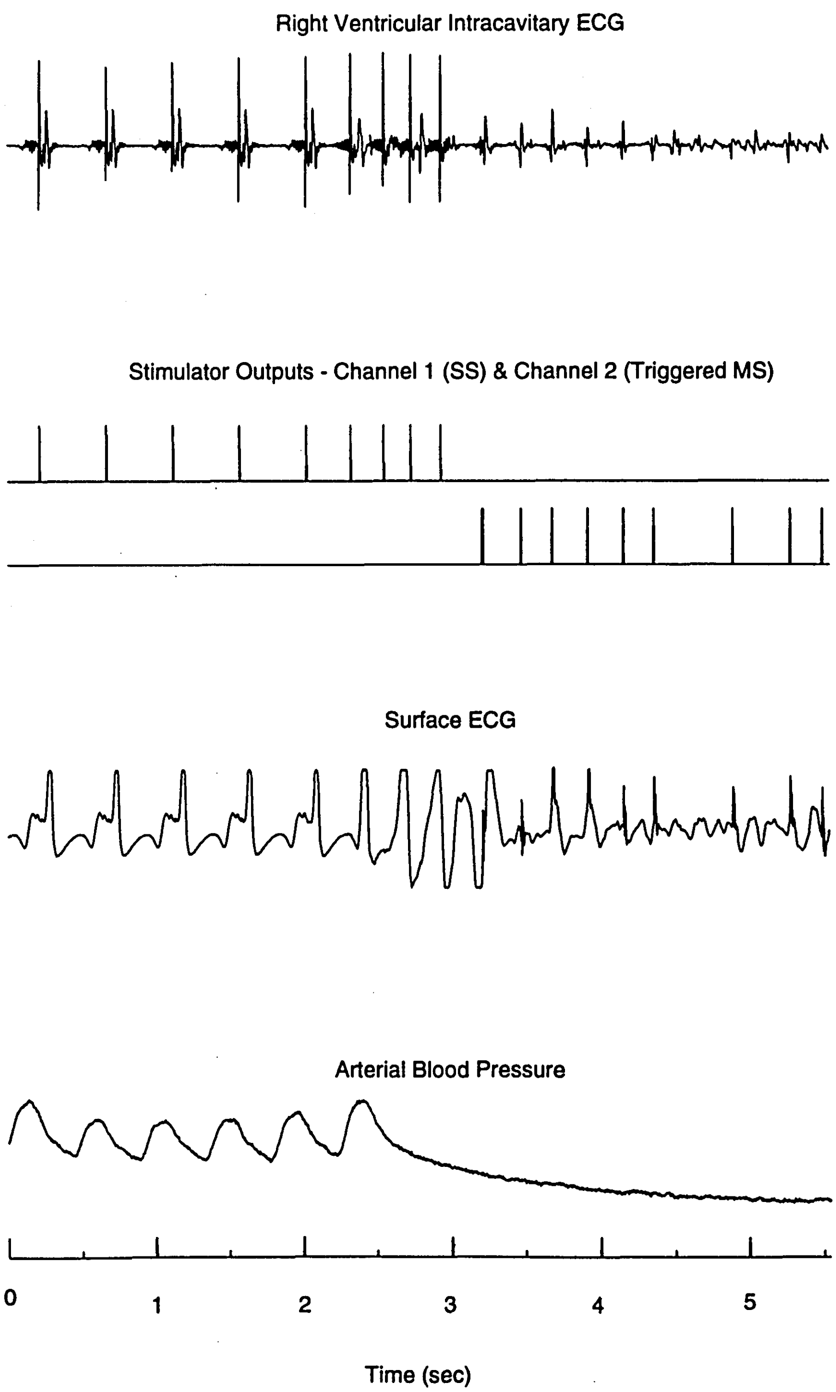

Figure 5-7: Triggerd multi-site pacing to prevent VT/VF

Chapter 6

Conclusion

The programmable stimulator system with the control program has provided our laboratory with an easy to use cardiac stimulator for the electrophysiologic study of the prevention of arrhythmias. As new research protocols are created, it is a relatively simple matter to add them into the control program. Also, new hardware components may be added easily into the current system.

Appendix A

Pacer.c

```
/*****************************************************************************
 *
 * File name --> pacer.c
 * Date --------> 08/22/93
 * Purpose ----> routine to operate DA board as a stimulator
 *
 *****************************************************************************/

#include <stdio.h> /* standard C I/O header file */
#include <bios.h>
#include <conio.h> /* console I/O header file */
#include <dos.h> /* command line */
#include "atldefs.h" /* ATLAB function definition file */
#include "atlerrs.h" /* ATLAB error definition file */
#include "video.h" /* video display function file */

/*****************************************************************************
 *
 * Function prototype
 *
 *****************************************************************************/

void INIT_PARAMETER_SCREEN (); /* initialize parameters on the screen */
void READ_KEYBOARD_INPUT (); /* read keyboard input */
void HIGHLIGHT (char kytmp2);
void FUNCTION_KEY (char kytmp2, int i);
void CURSOR (char kytmp2);
```

```
void KEYBOARD_SCREEN (int k, int l);

void PACING (int, int, int);
void PACING_PVC (int, int);
void PACING_NOPVC (int, int);
void EXT_STM (int, int);
void BURST (int, int);
void OVERDRIVE (int, int, int);
void TRIGGER (int, int, int);

/*****************************************************************************
 *
 * Initial value definitions for parameters
 *
 *****************************************************************************/

/* Note on voltage and current amplitudes

        DABoard:  one unit corresponds to 5 mV

        Ch 1 (Current): Range:  0 to 40 mA, assuming 200 ohm load resistance
                        Resolution:  0.05 mA
                        DA board output of 1000 mV (200 unit) gives 10 mA
                        (1000 mV / 100 ohm = 10 mA)

        Ch 2 (Voltage): Range:    0 to 50000 mV
                        Resolution:  25 mV
                        DA board output of 1000 mV (200 unit) gives 5000 mV
                        (1000 mV is attenuated by a factor of 5/7.2
                        and then amplified by a factor of 7.2 (V/V) )

        Ch 2 (Current): Range:  0 to 600 mA
                        Resolution:  0.3 mA
                        DA board output of 1000 mV (200 unit) gives 60 mA
                        (1000 mV is attenuated by a factor of 1/10
                        and then amplified by a factor of 0.6 (A/V) ) */

/* Pacing parameter initial values */

#define CHP_DE 1  /* 1 if pacing is on channel 1 */
#define OUTP_DE 1  /* 1 if pacing is current pulses */
#define AMPP_DE  200  /* 1 for 25 mV or 1 for 0.05 mA  */
#define IAP_DE 20
#define CYCL_DE 500  /* 500 ms */
#define IC_DE 20  /* 20 ms */
#define WIDTHP_DE 2  /* 2 ms */

/* Extra stimuli parameter initial values */
```

```
#define CHE_DE 1  /* 1 if extra stimuli are on channel 1 */
#define OUTE_DE 1  /* 1 if extra stimuli are current pulses */
#define AMPE_DE 200  /* 10 mA */
#define IAE_DE 20  /* 1 mA */
#define D1_DE 200  /* mode 1, 2, or 3 configuration */
#define D2_DE 0
#define D3_DE 0
#define D4_DE 0
#define ID_DE 10  /* 10 ms */
#define WIDTHE_DE 2  /* 2 ms */
#define EX_ST_DE 1

/* Overdrive pacing parameter initial values */

#define CHO_DE 2
#define OUTO_DE 2  /* 2 if overdrive are voltage pulses */
#define AMPO_DE 200
#define DOD_DE CYCL_DE
#define OD_DE 99
#define WIDTHO_DE 10

/* Trigger parameter initial values */

#define TRIGGER_DE 99
#define TRIG_LOCKE_DE 100
#define TRIG_LOCKO_DE 100
#define TRIG_LOCKB_DE 100
#define TRIG_WINDOW_DE 300
#define TRIG_DEMAND_DE 1000  /* 1000 msec */

/* Other definitions */

#define FALSE 0
#define TRUE 1
#define CURRENT 1
#define VOLTAGE 2

/*****************************************************************************
 *
 *  Global parameter declarations
 *
 *****************************************************************************/

/* parameter declarations for main function */

int ch[3]        = {CHP_DE,CHE_DE,CHO_DE};
int out[3]       = {OUTP_DE,OUTE_DE,OUTO_DE};
```

```
float amp[3]      = {AMPP_DE, AMPE_DE, AMPO_DE}; /* pulse amplitudes */
float ia[2]       = {IAP_DE, IAE_DE}; /* pulse amplitude increments */
int cycl          = CYCL_DE; /* cycle length */
int ic            = IC_DE; /* cycle length increment */
int delays[4]     = {D1_DE, D2_DE, D3_DE, D4_DE}; /* extra stimuli delays */
int id            = ID_DE; /* extra stimuli delay increments */
int width[3]      = {WIDTHP_DE, WIDTHE_DE, WIDTHO_DE}; /* pulse widths */
int ex_st         = EX_ST_DE; /* number of extra stimuli */

int dod           = DOD_DE; /* overdrive delay */
int od          = OD_DE; /* number of overdrive pacing stimuli; 99 for cont */

int trig          = TRIGGER_DE; /* number of tigger stimuli */
int trig_locke = TRIG_LOCKE_DE; /* trigger lockout time for extra */
int trig_locko    = TRIG_LOCKO_DE; /* trigger lockout time for od */
int trig_lockb = TRIG_LOCKB_DE; /* trigger lockout for spontaneous beat */
int trig_window = TRIG_WINDOW_DE;
int trig_demand = TRIG_DEMAND_DE;

/*** parameter declarations for function 'INIT_PARAMETER_SCREEN' ***/
char tmpstr[100];

/*** parameter declarations for function 'READ_KEYBOARD_INPUT' ***/
char kytmp1, kytmp2;

/*** parameter declarations for function 'HIGHLIGHT' ***/
char str[10];

/*** parameter declarations for function 'KEYBOARD_SCREEN' ***/
char kytmp;
float new_value;

/* parameter declarstions for flag */
int again_flg = TRUE;
int ex_st_flg;
int od_flg;
int again_flg2 = TRUE;
int out1_flg = CURRENT;
int out2_flg = VOLTAGE;
int mode_flg = 1;
int in_flg = FALSE;
int start_flg = FALSE;
int write_flg;
int fix_flg;

int i,j,k;

int *pts;
```

```
int accu = 0;

/*** parameter for waiter ***/
int intnumber = 0x15;
union REGS inregs, outregs;
long waitval = 1024;
int clock_tics;
int clock_tics1;

/*** parameter for readport ***/
int addr = 0x240;
unsigned port_read;

main() {

        /*****************************************************************
         *
         *   Parameter assignments
         *
         *****************************************************************/

        /*** parameter assignments for function 'PACING' ***/
        int number_buffer_p;
        int length_buffer_p;
        int number_pacing;

        /*** parameter assignments for function 'PACING_PVC' ***/
        int number_buffer_pvc;
        int length_buffer_pvc;

        /*** parameter assignments for function 'PACING_NOPVC ***/
        int number_buffer_nopvc;
        int length_buffer_nopvc;

        /*** parameter assignments for function 'EXT_STM' ***/
        int number_buffer_e;
        int length_buffer_e;
        /* NOTE: this better be fixed to account for 3 ex or 2 ex */

        /*** parameter assignments for function 'BURST' ***/
        int number_buffer_b;
        int length_buffer_b;

        /*** parameter assignments for function 'OVERDRIVE' ***/
        int number_buffer_op;
        int length_buffer_op;
        int number_overdrive;
```

```
/*** parameter assignments for function 'TRIGGER' ***/
int number_buffer_t;
int length_buffer_t;
int number_trigger;

/**********************************************************************
 *
 * Initialize video screen
 *
 **********************************************************************/

CLEAR_SCREEN ();
SETUP_SCREEN ();
INIT_PARAMETER_SCREEN ();

/**********************************************************************
 *
 * Choose from Main Menu
 *
 *    Protocol 1  f1  Triggered Multi-Site Pacing
 *    Protocol 2  f2
 *    Protocol 3  f3
 *    Protocol 4  f4
 *
 **********************************************************************/

CLEAR_COMMAND (); /* routine to clear command panel */
MAIN_MENU (); /* routine to put protocol menu in command pannel */
READ_KEYBOARD_INPUT (); /* routine to get a keyboard input */

/**********************************************************************
 *
 * Protocol 1  f1  Triggered Multi-Site Pacing
 *
 **********************************************************************/

if ((kytmp1 == 0) && (kytmp2 == 59)) {

    /*************************************************************
     *
     * Flag definitions
     *
     *************************************************************/

    again_flg = 1;      /* do it again when it's 1 */
    ex_st_flg = 1;      /* extra stimuli */
    od_flg = 1; /* overdrive */
    out1_flg = CURRENT;
```

```
out2_flg = CURRENT;

/************************************************************
 *
 * Begin protocol 1 loop
 *
 ************************************************************/

while(again_flg) {

        /****************************************************
         *
         * Choose from Protocol 1 Menu:
         *
         *   Level 1.1   f1    select mode & channel
         *   Level 1.2   f2    change output characteristics
         *   Level 1.3   cursor change output characteristics
         *   Level 1.4   f10   start pacing
         *       Level 1.4.1   cursor  change output characteristics
         *       Level 1.4.2   f1      stop pacing & exit to protocol 1 menu
         *       Level 1.4.3   f10     introduce extra stimuli & overdrive
         *   Level 1.5   esc   exit to the main menu
         *
         ****************************************************/

        CLEAR_COMMAND ();
        PROTOCOL1_MENU ();
        READ_KEYBOARD_INPUT ();

        /****************************************************
         *
         * Level 1.1  f1  select mode & channel
         *
         ****************************************************/

        if ((kytmp1 == 0) && (kytmp2 == 59)) { /* f1 */

                /****************************************
                 *
                 * Choose from Level 1.1 Menu:
                 *
                 * Level 1.1.1   f1   select mode
                 * Level 1.1.2   f2   select channel
                 * Level 1.1.3   esc  exit this level
                 *
                 ****************************************/

                CLEAR_COMMAND ();
```

```
PROTOCOL1_1_MENU ();
READ_KEYBOARD_INPUT ();

/*******************************************
 *
 * Level 1.1.1  f1  select mode
 *
 *******************************************/

if ((kytmp1 == 0) && (kytmp2 == 59)) { /* f1 */
        CLEAR_COMMAND ();
        PROTOCOL1_1_1_MENU ();
        READ_KEYBOARD_INPUT ();

        if ((kytmp1 == 0) && (kytmp2 == 59)) {
                mode_flg  = 1;
                ex_st_flg = 1;
                od_flg    = 1;
        }

        else if ((kytmp1 == 0) && (kytmp2 == 60)) {
                mode_flg  = 2;
                ex_st_flg = 1;
                od_flg    = 2;
        }

        else if ((kytmp1 == 0) && (kytmp2 == 61)) {
                mode_flg  = 3;
                ex_st_flg = 1;
                od_flg    = 3;
        }

        else if ((kytmp1 == 0) && (kytmp2 == 62)) {
                mode_flg  = 4;
                ex_st_flg = 2;
                od_flg    = 1;
        }

        else if ((kytmp1 == 0) && (kytmp2 == 63)) {
                mode_flg  = 5;
                ex_st_flg = 2;
                od_flg    = 2;
        }

        else if ((kytmp1 == 0) && (kytmp2 == 64)) {
                mode_flg  = 6;
                ex_st_flg = 2;
                od_flg    = 3;
```

```
        }

        else if ((kytmp1 == 0) && (kytmp2 == 65)) {
                mode_flg  = 7;
        }

        if (ex_st_flg == 1) {
                delays[0] = 200;
                delays[1] = 0;
                delays[2] = 0;
                delays[3] = 0;
                sprintf(tmpstr,"%4d ms",delays[0]);
                outtext(EXTRAx,D1y,tmpstr,0);
                sprintf(tmpstr,"%4d ms",delays[1]);
                outtext(EXTRAx,D2y,tmpstr,0);
                sprintf(tmpstr,"%4d ms",delays[2]);
                outtext(EXTRAx,D3y,tmpstr,0);
                sprintf(tmpstr,"%4d ms",delays[3]);
                outtext(EXTRAx,D4y,tmpstr,0);
        }

        else if (ex_st_flg == 2) {
                delays[0] = 200; /* delay before burst train */
                delays[1] = 100; /* period for which burst train is on */
                delays[2] = 8; /* delay between burst train */
                delays[3] = 0;
                sprintf(tmpstr,"%4d ms",delays[0]);
                outtext(EXTRAx,D1y,tmpstr,0);
                sprintf(tmpstr,"%4d ms",delays[1]);
                outtext(EXTRAx,D2y,tmpstr,0);
                sprintf(tmpstr,"%4d ms",delays[2]);
                outtext(EXTRAx,D3y,tmpstr,0);
                sprintf(tmpstr,"%4d ms",delays[3]);
                outtext(EXTRAx,D4y,tmpstr,0);
        }
}

/******************************************
 *
 * Level 1.1.2  f2  select channel
 *
 ******************************************/

else if ((kytmp1 == 0) && (kytmp2 == 60)) { /* f2 */
        CLEAR_COMMAND ();
        PROTOCOL1_1_2_MENU ();
        READ_KEYBOARD_INPUT ();
```

```
if ((kytmp1 == 0) && (kytmp2 == 59)) {
        CLEAR_COMMAND ();
        PROTOCOL1_1_2_1_MENU ();
        READ_KEYBOARD_INPUT ();

        if ((kytmp1 == 0) && (kytmp2 == 59)) {
                ch[0] = 1;
                sprintf(str,"%4d",ch[0]);
                outtext(PACINGx,CHPy,str,0);
        }

        else if ((kytmp1 == 0) && (kytmp2 == 60)) {
                ch[0] = 2;
                sprintf(str,"%4d",ch[0]);
                outtext(PACINGx,CHPy,str,0);
        }
}

else if ((kytmp1 == 0) && (kytmp2 == 60)) {
        CLEAR_COMMAND ();
        PROTOCOL1_1_2_1_MENU ();
        READ_KEYBOARD_INPUT ();

        if ((kytmp1 == 0) && (kytmp2 == 59)) {
                ch[1] = 1;
                sprintf(str,"%4d",ch[1]);
                outtext(EXTRAx,CHEy,str,0);
        }

        else if ((kytmp1 == 0) && (kytmp2 == 60)) {
                ch[1] = 2;
                sprintf(str,"%4d",ch[1]);
                outtext(EXTRAx,CHEy,str,0);
        }

}

else if ((kytmp1 == 0) && (kytmp2 == 61)) { /* f3 */
        CLEAR_COMMAND ();
        PROTOCOL1_1_2_1_MENU ();
        READ_KEYBOARD_INPUT ();

        if ((kytmp1 == 0) && (kytmp2 == 59)) {
                ch[2] = 1;
                sprintf(str,"%4d",ch[2]);
                outtext(Ox,CHOy,str,0);
        }
```

```
                    else if ((kytmp1 == 0) && (kytmp2 == 60)) {
                            ch[2] = 2;
                            sprintf(str,"%4d",ch[2]);
                            outtext(Ox,CHOy,str,0);
                    }

            }

        } /* end of level 1.1.2  f2  select channel */

        /*******************************************
         *
         * Level 1.1.3  esc  exit this level
         *
         *******************************************/

        else if (kytmp1 == 27) {

                /* don't do anything;
                   it will automatically go to the protocol 1 menu */

        } /* end of level 1.1.4  esc exit this level */

    } /* end of level 1.1 f1 (select channel, extra stimuli & overdrive) */

    /*********************************************
     *
     * Level 1.2  f2  change output characteristics
     *
     *********************************************/

    else if ((kytmp1 == 0) && (kytmp2 == 60)) { /* f2 */

        /****************************************
         *
         * Begin level 1.2 loop
         *
         ****************************************/

        while (1) {

CLEAR_COMMAND();
outtext(2,19,"Which output characteristics do you want to change?",0);
outtext(2,20,"f1 AMPP  f2 IAP  f3 WP    f4 CYCL f5 IC           ESC Exit",0);
outtext(2,21,"f6 AMPO  f7 OD   f8 DOD  f9 L_E  f10 L_O  #7 L_B  #8 WINDOW  #9 WO",0);
        outtext(2,22,"F1 AMPE  F2 IAE  F3 WE   F4 D1   F5  D2  F6  D3   F7  D4  F8 ID  F9 EX_ST",0);
READ_KEYBOARD_INPUT ();
```

```
                    if (kytmp1 == 27) break; /* Exit level 1.2 loop */
                    if (kytmp1 == 0) {
                            CLEAR_COMMAND ();
                            outtext(2,19,"Type in new value for ",0);
                            HIGHLIGHT(kytmp2);
                            outtext(28,19,", followed by RETURN key:",0);
                            KEYBOARD_SCREEN(54,19);
                                    /* program to print what's being typed */
                            FUNCTION_KEY(kytmp2,i);
                            /* AL_TERMINATE (); */
                    }

        }

} /* end of level 1.2 f2 (change output characteristics) */

/*************************************************
 *
 * Level 1.3  cursor  change output characteristics
 *
 *************************************************/

else if ((kytmp1 == 0) && (kytmp2 > 70) && (kytmp2 < 82)) { /* cursor */

        CURSOR(kytmp2);

} /* end of level 1.3 cursor (change output characteristics) */

/****************************************************
 *
 * Level 1.4  f10  begin pacing
 *
 ****************************************************/

else if ((kytmp1 == 0) && (kytmp2 == 68)) { /* f10 */

        /********************************************
         *
         * Initialize ATLAB for DA operation
         *
         ********************************************/

    /* initialize the ATLAB subroutines */
        AL_INITIALIZE ();
        /* select board 1, the first unit */
        AL_SELECT_BOARD (1);
        /* perform a reset on the device */
        AL_RESET ();
```

```
/* 0 – internal clock, software start
   –1 – output two values, one to each DAC
   channel simultanelously */
AL_SETUP_DAC (0,–1);
/* set sample point interval at 1 msec */
AL_SET_PERIOD (0.001);

CLEAR_COMMAND ();
PROTOCOL1_4_MENU (mode_flg);

again_flg2 = 1;

/**************************************
 *
 * Begin protocol 1.4 loop
 *
 **************************************/

while (again_flg2) {

/**********************************************
 *
 * Pacing
 *
 **********************************************/

number_buffer_p = 0;
length_buffer_p = cycl;
number_pacing   = 99;

PACING (number_buffer_p, length_buffer_p, number_pacing);

/*********************************************
 *
 * Choose from Level 1.4 Menu:
 *
 * Level 1.4.1  cursor  change output characteristics
 * Level 1.4.2  f1      stop pacing & exit to protocol 1 menu
 * Level 1.4.3  f10     introduce extra stimuli & overdrive
 *
 *********************************************/

READ_KEYBOARD_INPUT();

/*****************************************
 *
 * Level 1.4.1  cursor  change amp and cycle
```

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

if ((kytmp2 > 70) && (kytmp2 < 82)) {
        CURSOR(kytmp2);
}

/****************************************
 *
 * Level 1.4.2   f10   stop pacing
 *
 ****************************************/

else if ((kytmp1 == 0) && (kytmp2 == 68)) {

        again_flg2 = 0;  /* end while loop */

        CLEAR_COMMAND ();
        PROTOCOL1_MENU ();
}

/****************************************
 *
 * Level 1.4.3   f1   extra stimuli & overdrive
 *
 ****************************************/

else if ((kytmp1 == 0) && (kytmp2 == 59)) {

        /********************************
         *
         * Mode 1:   Extra stimuli only
         *
         ********************************/

      if (mode_flg == 1) {

                /************************
                 *
                 * ES for VT induction
                 *
                 ************************/

                number_buffer_e = 0;
                /* last pacing pulse width */
                length_buffer_e = width[0];
                for (i=0; i<ex_st; i++)
                        length_buffer_e
```

```
                = length_buffer_e + delays[i] + width[1];
        /* reset the value to 0 so to avoid the DC shock */
        length_buffer_e = length_buffer_e + 2;

        EXT_STM (number_buffer_e, length_buffer_e);

        AL_TERMINATE ();
}

/*************************************
 *
 * Mode 2:  Extra stimuli & fixed rate overdrive pacing
 *
 *************************************/

else if (mode_flg == 2) {       /* ES & OD */

        /*****************************
         *
         * ES for VT induction
         *
         *****************************/

        number_buffer_e = 0;
        length_buffer_e = width[0];
        for (i=0; i<ex_st; i++)
                length_buffer_e
                = length_buffer_e + delays[i] + width[1];
        length_buffer_e = length_buffer_e + 2;
        EXT_STM (number_buffer_e, length_buffer_e);

        /*****************************
         *
         * Fixed rate overdrive pacing
         *
         *****************************/

        number_buffer_op = 0;
        length_buffer_op = dod;
        number_overdrive = od;
        OVERDRIVE (number_buffer_op,
                  length_buffer_op, number_overdrive);

        AL_TERMINATE ();
}

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

```
 * Mode 3:  Extra stimuli & triggered pacing
 *
 *************************************/

else if (mode_flg == 3) {   /* ES &T & OD */

        /*****************************
         *
         * Extra stimuli for VT induction
         *
         *****************************/

        number_buffer_e = 0;
        length_buffer_e = width[0];
        for (i=0; i<ex_st; i++)
                length_buffer_e
                = length_buffer_e + delays[i] + width[1];
        length_buffer_e = length_buffer_e + 2;
        EXT_STM (number_buffer_e, length_buffer_e);

        /*****************************
         *
         * Trigger
         *
         *****************************/

        number_buffer_t = 0;
        length_buffer_t = width[0] + 2;
        number_trigger = trig;
        TRIGGER
        (number_buffer_t, length_buffer_t, number_trigger);

        AL_TERMINATE ();
}

/*************************************
 *
 * Mode 7:  Extra stimuli & PVC triggered pacing
 *
 *************************************/

else if (mode_flg == 7) {   /* ES &T & OD */

        /*****************************
         *
         * Extra stimuli to induce arrhythmias
         *
         *****************************/
```

```
            number_buffer_e = 0;
            length_buffer_e = width[0];
            for (i=0; i<ex_st; i++)
                    length_buffer_e
                    = length_buffer_e + delays[i] + width[1];
            length_buffer_e = length_buffer_e + 2;
            EXT_STM (number_buffer_e, length_buffer_e);

            CLEAR_COMMAND ();
            outtext(2,19,"Press the space bar
              to stop PVC triggered pacing",1);

            /*****************************
             * Wait the pacing inhibit time
             * to avoid sensing pacing pulse
             * transients
             *****************************/

            /* Initialization for waiter */
            inregs.h.ah = 0x86;
            inregs.x.cx = waitval >> 16;
            inregs.x.dx = waitval && 0xffff;

            /* 'clock_tics' are used to
               convert the 1.02 KHz clock
               to 1 KHz clock */
            clock_tics = (int) (((long)trig_locke *
                        (long)1024)  / (long)1000);
            for (i=0; i<clock_tics; i++)
                    int86 (intnumber, &inregs, &outregs);

/*****************************
 *
 * PVC Trigger
 *
 *****************************/

/* Set up ports 0 & 1 of DT2821 interface board to read in words */
outport (addr+0x6,0x00);
clock_tics = (int) (((long)(trig_window-trig_locke) *
              (long)1024) /(long)1000);

/* Start monitoring trigger */

do {
   for (i=0; i< clock_tics; i++) {
        port_read = inport(addr+0xa);
```

```
        if (port_read == 0xfeff) {

            /* output triggered pacing pulse */
            number_buffer_pvc = 0;
            length_buffer_pvc = width[2] + 2;
            PACING_PVC(number_buffer_pvc, length_buffer_pvc);

            /* wait trig lockout time of overdrive */
            clock_tics1 = (int) (((long)trig_locko *
                          (long)1024)  / (long)1000);
            for (i=0; i<clock_tics1; i++)
              int86 (intnumber, &inregs, &outregs);

            /* new initialization */
            clock_tics = (int) (((long)(trig_window−trig_locko) *
                              (long)1024) / (long)1000);

            /* get out of 'for' loop */
            break;
         }

    /* if the end of triggering window is reached */

        else if (i == clock_tics − 1) {

            /* wait for the spontaneous beat */

            clock_tics1 = (int) (((long)(trig_demand − trig_window) *
                          (long)1024) /(long)1000);
            j = 0;

            do {
               int86(intnumber,&inregs, &outregs);
               j = j + 1;
               port_read = inport (addr+0xa);
            } while ( port_read == 0xffff && j < clock_tics1 );

            if (j < clock_tics1) {

              /* wait trigger lockout time of spontaneous beat */
              clock_tics1 = (int) (((long)trig_lockb * (long)1024) /
                              (long)1000);
              for (j=0; j<clock_tics1; j++)
                int86 (intnumber, &inregs, &outregs);

              /* new initialization */
              clock_tics = (int) (((long)(trig_window−trig_lockb) *
```

```
                                (long)1024) / (long)1000);

          /* get out of 'for' loop */
          break;
        }

        else {
          number_buffer_nopvc = 0;
          length_buffer_nopvc = width[0] + 2;
          PACING_NOPVC (number_buffer_nopvc, length_buffer_nopvc);

          /* wait trigger lockout time of channel 1 (or ES)  */
          clock_tics1 = (int) (((long)trig_locke * (long)1024)  /
                           (long)1000);
          for (j=0; j<clock_tics1; j++)
                int86 (intnumber, &inregs, &outregs);

          /* new initialization */
          clock_tics = (int) (((long)(trig_window-trig_locke) *
                           (long)1024) / (long)1000);

          /* get out of 'for' loop */
          break;
        }
    }

       else {
         int86 (intnumber, &inregs, &outregs);
       }

                                              } /* end of for loop */

                                  } while ( !kbhit() );

                                  /* kbhit() puts the key hit in register,
                                     so need to be removed */

                                  getch ();

                                  AL_TERMINATE ();

                        } /* end of mode 7 */

                        /***************************************
                         *
                         * Mode 4:  Burst only
                         *
                         ***************************************/
```

```
else if (mode_flg == 4) {    /* Burst */

        number_buffer_b = 0;
        length_buffer_b = delays[0] + delays[1];
        BURST(number_buffer_b, length_buffer_b);

        AL_TERMINATE ();
}

/*************************************
 *
 * Mode 5:   Burst & fixed rate overdrive pacing
 *
 *************************************/

else if (mode_flg == 5) {    /* Burst & OD */

        number_buffer_b = 0;
        length_buffer_b = delays[0] + delays[1];
        BURST(number_buffer_b, length_buffer_b);

        number_buffer_op = 0;
        length_buffer_op = cycl;
        number_overdrive = od;
        OVERDRIVE
        (number_buffer_op, length_buffer_op, number_overdrive);

        AL_TERMINATE ();
}

/*******************************************
 *
 * Mode 6:   Burst & triggered OD
 *
 *******************************************/

else if (mode_flg == 6) {    /* B & T */

        number_buffer_b = 0;
        length_buffer_b = delays[0] + delays[1];
        BURST(number_buffer_b, length_buffer_b);

        /*****************************
         *
         * Trigger
         *
         *****************************/
```

```
                        number_buffer_t = 0;
                        length_buffer_t = width[0] + 2;
                        number_trigger = trig;
                        TRIGGER (number_buffer_t, length_buffer_t, number_trigger)

                        AL_TERMINATE ();
                    }

                    again_flg2 = 0;
                    CLEAR_COMMAND ();
                    PROTOCOL1_MENU ();

                }  /* different extra stimuli loop */

            }  /* while loop */

        }  /* end of level 1.4 */

        /**************************************************
         *
         * Level 1.5:   esc (exit)
         *
         **************************************************/

        else if (kytmp1 == 27) { /* esc */

            again_flg = 0;

        } /* end of lelve 1.5 esc (exit) */

    } /* end of while loop */

    /*** put in 'EXIT_HANDLER ()' which resets all outputs to avoid
    accidential outputs ***/
  }                                           /* end of protocol 1 */
}                                             /* end of main.c */

/**************************************************************************
 *
 * Function --------> INIT_PARAMETER_SCREEN()
 *
 * Return Type ----> void
 *
 * Description ----> routine to put initial parameter values on the screen
 *
 **************************************************************************/
```

```
void INIT_PARAMETER_SCREEN () {

        /*** PACING panel ***/

        sprintf(tmpstr,"%4d",ch[0]);
        outtext(PACINGx,CHPy,tmpstr,0);
        sprintf(tmpstr,"%4.0f mA",amp[0]/20);
        outtext(PACINGx,AMPPy,tmpstr,0);
        sprintf(tmpstr,"%4.0f mA",ia[0]/20);
        outtext(PACINGx,IAPy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",width[0]);
        outtext(PACINGx,WIDTHPy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",cycl);
        outtext(PACINGx,CYCLy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",ic);
        outtext(PACINGx,ICy,tmpstr,0);

        /*** EXTRA STIMULI panel ***/

        sprintf(tmpstr,"%4d",ch[1]);
        outtext(EXTRAx,CHEy,tmpstr,0);
        sprintf(tmpstr,"%4.0f mA",amp[1]/20);
        outtext(EXTRAx,AMPEy,tmpstr,0);
        sprintf(tmpstr,"%4.0f mA",ia[1]/20);
        outtext(EXTRAx,IAEy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",width[1]);
        outtext(EXTRAx,WIDTHEy,tmpstr,0);
        sprintf(tmpstr,"%4d",ex_st);
        outtext(EXTRAx,EX_STy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",delays[0]);
        outtext(EXTRAx,D1y,tmpstr,0);
        sprintf(tmpstr,"%4d ms",delays[1]);
        outtext(EXTRAx,D2y,tmpstr,0);
        sprintf(tmpstr,"%4d ms",delays[2]);
        outtext(EXTRAx,D3y,tmpstr,0);
        sprintf(tmpstr,"%4d ms",delays[3]);
        outtext(EXTRAx,D4y,tmpstr,0);
        sprintf(tmpstr,"%4d ms",id);
        outtext(EXTRAx,IDy,tmpstr,0);

        /*** OVERDRIVE panel ***/

        sprintf(tmpstr,"%4d",ch[2]);
        outtext(Ox,CHOy,tmpstr,0);
        sprintf(tmpstr,"%4.0f mV",amp[2]*25);
        outtext(Ox,AMPOy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",width[2]);
```

```
        outtext(Ox,WIDTHOy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",dod);
        outtext(Ox,DODy,tmpstr,0);
        sprintf(tmpstr,"%4d",od);
        outtext(Ox,ODy,tmpstr,0);
        sprintf(tmpstr,"%4d",trig);
        outtext(Ox,TRIGy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",trig_locko);
        outtext(Ox,LOCKy,tmpstr,0);

        /*** TRIGGER panel ***/

        sprintf(tmpstr,"%4d ms",trig_locke);
        outtext(Tx,LOCK_Ey,tmpstr,0);
        sprintf(tmpstr,"%4d ms",trig_locko);
        outtext(Tx,LOCK_Oy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",trig_lockb);
        outtext(Tx,LOCK_By,tmpstr,0);
        sprintf(tmpstr,"%4d ms",trig_window);
        outtext(Tx,WINDOWy,tmpstr,0);
        sprintf(tmpstr,"%4d ms",trig_demand);
        outtext(Tx,DEMANDy,tmpstr,0);
}

/***************************************************************************
 *
 * Function --------> READ_KEYBOARD_INPUT()
 *
 * Return Type -----> void
 *
 * Description -----> routine to read keyboard input
 *
 ***************************************************************************/

void READ_KEYBOARD_INPUT() {

        kytmp1 = getch();
        if (kytmp1 == 0)
                kytmp2 = getch();         /* ESC, cursors, function keys */
}

/***************************************************************************
 *
 * Function --------> HIGHLIGHT(char kytmp2)
 *
 * Return Type -----> void
 *
 * Description -----> routine to highlight
```

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

void HIGHLIGHT(char kytmp2){

        int posi_x = 24, posi_y = 19;

        switch (kytmp2) {
                case 59: outtext(posi_x,posi_y,"AMPP",0); /* f1 */
                         if (out[0] == VOLTAGE)
                                 sprintf(str,"%4.0f",amp[0]*25);
                         else if (out[0] == CURRENT)
                                 sprintf(str,"%4.0f",amp[0]/20);
                         outtext(PACINGx,AMPPy,str,1);
                         break;
                case 60: outtext(posi_x,posi_y,"IAP",0); /* f2 */
                         if (out[0] == VOLTAGE)
                                 sprintf(str,"%4.0f",ia[0]*25);
                         else if (out[0] == CURRENT)
                                 sprintf(str,"%4.0f",ia[0]/20);
                         outtext(PACINGx,IAPy,str,1);
                         break;
                case 61: outtext(posi_x,posi_y,"WP",0); /* f3 */
                         sprintf(str,"%4d",width[0]);
                         outtext(PACINGx,WIDTHPy,str,1);
                         break;
                case 62: outtext(posi_x,posi_y,"CYCL",0); /* f4 */
                         sprintf(str,"%4d",cycl);
                         outtext(PACINGx,CYCLy,str,1);
                         break;
                case 63: outtext(posi_x,posi_y,"IC",0); /* f5 */
                         sprintf(str,"%4d",ic);
                         outtext(PACINGx,ICy,str,1);
                         break;
                case 64: outtext(posi_x,posi_y,"AMPO",0); /* f6 */
                         if (out[2] == VOLTAGE)
                                 sprintf(str,"%4.0f",amp[2]*25);
                         else if (out[2] == CURRENT)
                                 sprintf(str,"%4.0f",amp[2]*0.3);
                         outtext(Ox,AMPOy,str,1);
                         break;
                case 65: outtext(posi_x,posi_y,"OD",0); /* f7 */
                         sprintf(str,"%4d",od);
                         outtext(Ox,ODy,str,1);
                         break;
                case 66: outtext(posi_x,posi_y,"DOD",0); /* f8 */
                         sprintf(str,"%4d",dod);
                         outtext(Ox,DODy,str,1);
```

```
            break;

    case 67: outtext(posi_x,posi_y,"LOCK_E",0); /* f9 */
            sprintf(str,"%4d",trig_locke);
            outtext(Tx,LOCK_Ey,str,1);
            break;
    case 68: outtext(posi_x,posi_y,"LOCK_O",0); /* f10 */
            sprintf(str,"%4d",trig_locko);
            outtext(Ox,LOCKy,str,1);
            outtext(Tx,LOCK_Oy,str,1);
            break;
    case 71: outtext(posi_x,posi_y,"LOCK_B",0); /* #7 */
            sprintf(str,"%4d",trig_lockb);
            outtext(Tx,LOCK_By,str,1);
            break;
    case 72: outtext(posi_x,posi_y,"WINDOW",0); /* #8 */
            sprintf(str,"%4d",trig_window);
            outtext(Tx,WINDOWy,str,1);
            break;
    case 73: outtext(posi_x,posi_y,"WO",0); /* #9 */
            sprintf(str,"%4d",width[2]);
            outtext(Ox,WIDTHOy,str,1);
            break;

    case 84: outtext(posi_x,posi_y,"AMPE",0); /* F1 */
            if (out[1] == VOLTAGE)
                    sprintf(str,"%4.0f",amp[1]*25);
            else if (out[1] == CURRENT)
                    sprintf(str,"%4.0f",amp[1]/20);
            outtext(EXTRAx,AMPEy,str,1);
            break;
    case 85: outtext(posi_x,posi_y,"IAE",0); /* F2 */
            if (out[1] == VOLTAGE)
                    sprintf(str,"%4.0f",ia[1]*25);
            else if (out[1] == CURRENT)
                    sprintf(str,"%4.0f",ia[1]/20);
            outtext(EXTRAx,IAEy,str,1);
            break;
    case 86: outtext(posi_x,posi_y,"WE",0); /* F3 */
            sprintf(str,"%4d",width[1]);
            outtext(EXTRAx,WIDTHEy,str,1);
            break;
    case 87: outtext(posi_x,posi_y,"D1",0); /* F4 */
            sprintf(str,"%4d",delays[0]);
            outtext(EXTRAx,D1y,str,1);
            break;
    case 88: outtext(posi_x,posi_y,"D2",0); /* F5 */
            sprintf(str,"%4d",delays[1]);
```

```
                        outtext(EXTRAx,D2y,str,1);
                        break;
                case 89: outtext(posi_x,posi_y,"D3",0); /* F6 */
                        sprintf(str,"%4d",delays[2]);
                        outtext(EXTRAx,D3y,str,1);
                        break;
                case 90: outtext(posi_x,posi_y,"D4",0); /* F7 */
                        sprintf(str,"%4d",delays[3]);
                        outtext(EXTRAx,D4y,str,1);
                        break;
                case 91: outtext(posi_x,posi_y,"ID",0); /* F8 */
                        sprintf(str,"%4d",id);
                        outtext(EXTRAx,IDy,str,1);
                        break;
                case 92: outtext(posi_x,posi_y,"EX_ST",0); /* F9 */
                        sprintf(str,"%4d",ex_st);
                        outtext(EXTRAx,EX_STy,str,1);
                        break;

        }
}

/*****************************************************************************
 *
 * Function --------> KEYBOARD_SCREEN (int k, int l)
 *
 * Return Type ----> void
 *
 * Description ----> routine to put keyboard input onto the screen
 *
 *****************************************************************************/

void KEYBOARD_SCREEN (int k, int l) {

        i=0, j=0;

        while ((kytmp = getch()) != 13){            /* implement delete key */
                tmpstr[0]=kytmp;
                tmpstr[1]='\0';
                str[i++]=kytmp;
                outtext(k+j,l,tmpstr,0);
                j=j+1;
        }

        str[i] = '\0';                          /* End of line */
        new_value = atoi(str);
/*      outtext(1,l,tmpstr2,0); */
}
```

```
/**************************************************************************
 *
 * Function --------> FUNCTION_KEY (char kytmp2, int i)
 *
 * Return Type -----> void
 *
 * Description -----> routine to read function key input and do the following
 *
 **************************************************************************/

void FUNCTION_KEY(char kytmp2, int i){

        switch (kytmp2) {

                case 59: if (str[i-1] == 'V') {                         /* f1 */
                                out[0] = VOLTAGE;
                                amp[0] = new_value / 25;
                                for (i=0; i<strlen(str); i++) {
                                        if (str[i] == 32) { /* blank space ? */
                                                str[i] = '\0'; /* put in end of line */
                                                break;
                                        }
                                }
                                sprintf(str,"%4.0f mV ",new_value);
                                outtext(PACINGx,AMPPy,str,0);
                                sprintf(str,"%4.0f mV ",ia[0]*25);
                                outtext(PACINGx,IAPy,str,0);
                        }

                    else if (str[i-1] == 'A') {
                                out[0] = CURRENT;
                                amp[0] = new_value * 20;
                                for (i=0; i<strlen(str); i++) {
                                        if (str[i] == 32) {
                                                str[i] = '\0';
                                                break;
                                        }
                                }
                                sprintf(str,"%4.0f mA ",new_value);
                                outtext(PACINGx,AMPPy,str,0);
                                sprintf(str,"%4.0f mA ",ia[0]/20);
                                outtext(PACINGx,IAPy,str,0);
                        }

                        else {
                                if (out[0] == VOLTAGE) {
                                        amp[0] = new_value / 25;
```

```
                    sprintf(str,"%4.0f mV ",new_value);
                    outtext(PACINGx,AMPPy,str,0);
                }
                else if (out[0] == CURRENT) {
                    amp[0] = new_value * 20;
                    sprintf(str,"%4.0f mA ",new_value);
                    outtext(PACINGx,AMPPy,str,0);
                }
        }

            break;

case 60: if (out[0] == VOLTAGE) {                          /* f2 */
                ia[0] = new_value / 25;
                sprintf(str,"%4.0f mV ",new_value);
            }
            else if (out[0] == CURRENT) {
                ia[0] = new_value * 20;
                sprintf(str,"%4.0f mA",new_value);
            }
            outtext(PACINGx,IAPy,str,0);
            break;
case 61: width[0] = new_value;                     /* f3 */
            sprintf(str,"%4.0f ms ",new_value);
            outtext(PACINGx,WIDTHPy,str,0);
            break;
case 62: cycl = new_value;                         /* f4 */
            sprintf(str,"%4.0f ms ",new_value);
            outtext(PACINGx,CYCLy,str,0);
        fix_flg = 2;
            break;
case 63: ic = new_value;                           /* f5 */
            sprintf(str,"%4.0f ms ",new_value);
            outtext(PACINGx,ICy,str,0);
            break;
case 64: if (str[i-1] == 'V') {                            /* f6 */
                out[2] = VOLTAGE;
                amp[2] = new_value / 25;
                for (i=0; i<strlen(str); i++) {
                    if (str[i] == 32) { /* blank space ? */
                        str[i] = '\0'; /* put in end of line */
                        break;
                    }
                }
                sprintf(str,"%4.0f mV ",new_value);
                outtext(Ox,AMPOy,str,0);
        }
```

```
            else if (str[i-1] == 'A') {
                out[2] = CURRENT;
                amp[2] = new_value / 0.3;
                for (i=0; i<strlen(str); i++) {
                    if (str[i] == 32) {
                        str[i] = '\0';
                        break;
                    }
                }
                sprintf(str,"%4.0f mA ",new_value);
                outtext(Ox,AMPOy,str,0);
        }

        else {
                if (out[2] == VOLTAGE) {
                    amp[2] = new_value / 25;
                    sprintf(str,"%4.0f mV ",new_value);
                    outtext(Ox,AMPOy,str,0);
                }
                else if (out[2] == CURRENT) {
                    amp[2] = new_value / 0.3;
                    sprintf(str,"%4.0f mA ",new_value);
                    outtext(Ox,AMPOy,str,0);
                }
        }
            break;
case 65: od = new_value;                        /* f7 */
        sprintf(str,"%4.0f ",new_value);
            outtext(Ox,ODy,str,0);
            break;
  case 66: dod = new_value;                        /* f8 */
            sprintf(str,"%4.0f ms ",new_value);
            outtext(Ox,DODy,str,0);
            break;
  case 67: trig_locke = new_value;                    /* f9 */
            sprintf(str,"%4.0f ",new_value);
            outtext(Tx,LOCK_Ey,str,0);
            break;
  case 68: trig_locko = new_value;              /* f10 */
        sprintf(str,"%4.0f ",new_value);
            outtext(Ox,LOCKy,str,0);
            outtext(Tx,LOCK_Oy,str,0);
            break;
  case 71: trig_lockb = new_value;                    /* #7 */
            sprintf(str,"%4.0f ",new_value);
            outtext(Tx,LOCK_By,str,0);
            break;
  case 72: trig_window = new_value;                   /* #8 */
```

```
            sprintf(str,"%4.0f ",new_value);
            outtext(Tx,WINDOWy,str,0);
            break;
case 73: width[2] = new_value;                    /* #9 */
            sprintf(str,"%4.0f ms ",new_value);
            outtext(Ox,WIDTHOy,str,0);
            break;

case 84: if (str[i-1] == 'V') {                           /* F1 */
                    out[1] = VOLTAGE;
                    amp[1] = new_value / 25;
                    for (i=0; i<strlen(str); i++) {
                            if (str[i] == 32) { /* blank space ? */
                                    str[i] = '\0'; /* put in end of line */
                                    break;
                            }
                    }
                    sprintf(str,"%4.0f mV ",new_value);
                    outtext(EXTRAx,AMPEy,str,0);
                    sprintf(str,"%4.0f mV ",ia[1]*25);
                    outtext(EXTRAx,IAEy,str,0);
      }

         else if (str[i-1] == 'A') {
                    out[1] = CURRENT;
                    amp[1] = new_value * 20;
                    for (i=0; i<strlen(str); i++) {
                            if (str[i] == 32) {
                                    str[i] = '\0';
                                    break;
                            }
                    }
                    sprintf(str,"%4.0f mA ",new_value);
                    outtext(EXTRAx,AMPEy,str,0);
                    sprintf(str,"%4.0f mA ",ia[1]/20);
                    outtext(EXTRAx,IAEy,str,0);
      }

      else {
                    if (out[1] == VOLTAGE) {
                            amp[1] = new_value / 25;
                            sprintf(str,"%4.0f mV ",new_value);
                            outtext(EXTRAx,AMPEy,str,0);
                    }
                    else if (out[1] == CURRENT) {
                            amp[1] = new_value * 20;
                            sprintf(str,"%4.0f mA ",new_value);
                            outtext(EXTRAx,AMPEy,str,0);
```

```
                        }
                }

                  break;

        case 85: if (out[1] == VOLTAGE) {                    /* F2 */
                        ia[1] = new_value / 25;
                        sprintf(str,"%4.0f mV ",new_value);
                  }
                  else if (out[1] == CURRENT) {
                        ia[1] = new_value * 20;
                        sprintf(str,"%4.0f mA ",new_value);
                }
                  outtext(EXTRAx,IAEy,str,0);
                  break;
        case 86: width[1] = new_value;                    /* F3 */
                  sprintf(str,"%4.0f ms ",new_value);
                  outtext(EXTRAx,WIDTHEy,str,0);
                  break;
        case 87: delays[0] = new_value;                   /* F4 */
                  sprintf(str,"%4.0f ms ",new_value);
                  outtext(EXTRAx,D1y,str,0);
                  break;
        case 88: delays[1] = new_value;                   /* F5 */
                  sprintf(str,"%4.0f ms ",new_value);
                  outtext(EXTRAx,D2y,str,0);
                  break;
        case 89: delays[2] = new_value;                      /* F6 */
                  sprintf(str,"%4.0f ms ",new_value);
                  outtext(EXTRAx,D3y,str,0);
                  break;
        case 90: delays[3] = new_value;                   /* F7 */
                  sprintf(str,"%4.0f ms ",new_value);
                  outtext(EXTRAx,D4y,str,0);
                  break;
        case 91: id = new_value;                          /* F8 */
                  sprintf(str,"%4.0f",new_value);
                  outtext(EXTRAx,IDy,str,0);
                  break;
        case 92: ex_st = new_value;                          /* F9 */
                  sprintf(str,"%4.0f",new_value);
                  outtext(EXTRAx,EX_STy,str,0);
                  break;

        }
}

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

```
*
* Function --------> CURSOR(char kytmp2)
*
* Return Type ----> void
*
* Description ----> routine to do the following actions upon cursor input
*
****************************************************************************/

void CURSOR (char kytmp2){

        switch(kytmp2){
                case 71: amp[0] = amp[0] + ia[0];                      /* home */
                         if (out[0] == VOLTAGE) {
                                 sprintf(tmpstr,"%4.0f mV ",amp[0]*25);
                         }
                         else if (out[0] == CURRENT) {
                                 sprintf(tmpstr,"%4.0f mA ",amp[0]/20);
                         }
                         outtext(PACINGx,AMPPy,tmpstr,0);
                         fix_flg = 1;
                         break;
                case 72: amp[1] = amp[1] + ia[1];              /* up arrow */
                         if (out[1] == VOLTAGE) {
                                 sprintf(tmpstr,"%4.0f mV ",amp[0]*25);
                         }
                         else if (out[1] == CURRENT) {
                                 sprintf(tmpstr,"%4.0f mA ",amp[1]/20);
                         }
                         outtext(EXTRAx,AMPEy,tmpstr,0);
                         fix_flg = 1;
                         break;
                case 73: delays[ex_st-1] = delays[ex_st-1] + id;   /* pg up */
                         sprintf(tmpstr,"%4d ms ",delays[ex_st-1]);
                         if (ex_st == 1) outtext(EXTRAx,D1y,tmpstr,0);
                         else if (ex_st == 2)  outtext(EXTRAx,D2y,tmpstr,0);
                         else if (ex_st == 3)  outtext(EXTRAx,D3y,tmpstr,0);
                         else if (ex_st == 4)  outtext(EXTRAx,D4y,tmpstr,0);
                         break;
                case 75: cycl = cycl - ic;                     /* left arrow */
                         sprintf(tmpstr,"%4d ms ",cycl);
                         outtext(PACINGx,CYCLy,tmpstr,0);
                         fix_flg = 2;
                         break;
                case 77: cycl = cycl + ic;                     /* right arrow */
                         sprintf(tmpstr,"%4d ms ",cycl);
                         outtext(PACINGx,CYCLy,tmpstr,0);
                         fix_flg = 2;
```

```
                        break;
                case 79: amp[0] = amp[0] - ia[0];                    /* end */
                        if (out[0] == VOLTAGE) {
                                sprintf(tmpstr,"%4.0f mV ",amp[0]*25);
                        }
                        else if (out[0] == CURRENT) {
                             sprintf(tmpstr,"%4.0f mA ",amp[0]/20);
                        }
                        outtext(PACINGx,AMPPy,tmpstr,0);
                        fix_flg = 1;
                        break;
                case 80: amp[1] = amp[1] - ia[1];               /* down arrow */
                        if (out[1] == VOLTAGE) {
                                sprintf(tmpstr,"%4.0f mV ",amp[1]*25);
                        }
                        else if (out[1] == CURRENT) {
                                sprintf(tmpstr,"%4.0f mA ",amp[1]/20);
                        }
                        outtext(EXTRAx,AMPEy,tmpstr,0);
                        fix_flg = 1;
                        break;
                case 81: delays[ex_st-1] = delays[ex_st-1] - id; /* pg down */
                        sprintf(tmpstr,"%4d ms ",delays[ex_st-1]);
                        if (ex_st == 1) outtext(EXTRAx,D1y,tmpstr,0);
                        else if (ex_st == 2)  outtext(EXTRAx,D2y,tmpstr,0);
                        else if (ex_st == 3)  outtext(EXTRAx,D3y,tmpstr,0);
                        else if (ex_st == 4)  outtext(EXTRAx,D4y,tmpstr,0);
                        break;
        }
}

/*************************************************************************
 *
 * Function --------> PACING (int number_buffer_p, int length_buffer_p, int number_pacing)
 *
 * Return Type ----> void
 *
 * Description ----> routine to produce pacing buffer
 *
 *************************************************************************/

void PACING (int number_buffer_p, int length_buffer_p, int number_pacing) {

        int BUFFER_P[2000];

        /*************************************/
        /*** declare BUFFER_P  for pacing ***/
        /*************************************/
```

```
AL_DECLARE_BUFFER (number_buffer_p, BUFFER_P, length_buffer_p*2);
AL_LINK_BUFFER (number_buffer_p);

/***************************************/
/*** fill BUFFER_P with pacing data ***/
/***************************************/

for (i=0; i<length_buffer_p*2; i++)          /* initialize BUFFER_P */
        BUFFER_P[i]=2048;

for (i=ch[0]-1; i<width[0]*2; i=i+2)
        BUFFER_P[i]=2048+amp[0];

/***************************************/
/*** set up output                  ***/
/***************************************/

AL_SETUP_DAC (0,-1);                    /* 0 - internal clock
                                           -1 - output two values to channels 1 & 2 */

AL_SET_PERIOD (0.001);

/***************************************/
/*** output the pacing stimuli      ***/
/***************************************/

if (number_pacing != 99) {
        for (i=0; i<number_pacing; i++) {

                AL_BURST_DAC ();

                outtext(53,19," **    ** ",2);
                outtext(53,20,"**** ****",2);
                outtext(53,21," ******* ",2);
                outtext(53,22,"  *****  ",2);
                outtext(53,23,"    *    ",2);

                AL_WAIT_FOR_COMPLETION (number_buffer_p);

        }
}

else if (number_pacing == 99) {
        while ( !kbhit() ) {          /* while key is not hit, repeat */

                AL_BURST_DAC ();
```

```
                    outtext(53,19," **   ** ",2);
                    outtext(53,20,"**** ****",2);
                    outtext(53,21," ******* ",2);
                    outtext(53,22,"  *****  ",2);
                    outtext(53,23,"    *    ",2);

                    AL_WAIT_FOR_COMPLETION (number_buffer_p);

            }
        }

        outtext(53,19,"          ",0);    /* to erase the heart */
        outtext(53,20,"          ",0);
        outtext(53,21,"          ",0);
        outtext(53,22,"          ",0);
        outtext(53,23,"          ",0);

/*      for (i=0; i<cycl*2; i++)
                BUFFER_P[i] = 2048; */  /* is this necessary? */

        AL_UNLINK_BUFFERS ();
        AL_UNDECLARE_BUFFER (number_buffer_p);
}

/***************************************************************************
 *
 * Function --------> EXT_STM (int number_buffer_e, int length_buffer_e)
 *
 * Return Type ----> void
 *
 * Description ----> routine to produce extra stimuli for VT induction
 *
 ***************************************************************************/

void EXT_STM (int number_buffer_e, int length_buffer_e) {

     int BUFFER_E[5000];

        /********************************************/
        /*** declare BUFFER_E for extra stimuli ***/
        /********************************************/

        AL_DECLARE_BUFFER (number_buffer_e, BUFFER_E, length_buffer_e*2);
        AL_LINK_BUFFER (number_buffer_e);

        /**********************************************/
        /*** fill BUFFER_E with extra stimuli data ***/
        /**********************************************/
```

```
        for (i=0; i < (length_buffer_e*2); i++)  /* initialize BUFFER_E */
                BUFFER_E[i] = 2048;

        for (j=ch[0]-1; j < (width[0]*2); j=j+2)  /* last pacing pulse before es */
                BUFFER_E[j] = 2048 + amp[0];

        accu = width[0] + delays[0];

        for (i=0; i < ex_st - 1; i++) {                  /* first three extra stimuli */
                for (j=(ch[1]-1)+(accu*2); j < (accu+width[1])*2; j=j+2)
                        BUFFER_E[j] = 2048 + amp[1];

                accu = accu + width[1] + delays[i+1];
        }

        for (j=(ch[1]-1)+(accu*2); j < (accu+width[1])*2; j=j+2)  /* last es */
                BUFFER_E[j] = 2048 + amp[1];

        /****************************************/
        /*** set up output                   ***/
        /****************************************/

        AL_SETUP_DAC (0,-1);                   /* 0 - internal clock
                                                  -1 - output two values to channels 1 & 2 */

        AL_SET_PERIOD (0.001);

        /*********************************/
        /*** output the extra stimuli ***/
        /*********************************/

        AL_BURST_DAC ();
        AL_WAIT_FOR_COMPLETION (number_buffer_e);
        AL_UNLINK_BUFFERS ();
        AL_UNDECLARE_BUFFER (number_buffer_e);
}

/****************************************************************************
 *
 * Function --------> OVERDRIVE(int number_buffer_op, int length_buffer_op,
 *                              int number_overdrive)
 *
 * Return Type -----> void
 *
 * Description -----> routine to do overdrive pacing
 *
 ****************************************************************************/
```

```
void OVERDRIVE(int number_buffer_op, int length_buffer_op,
                                int number_overdrive) {

        int BUFFER_OP[2000];

        /**********************************************/
        /*** declare BUFFER_OP for overdrive pacing ***/
        /**********************************************/

        AL_DECLARE_BUFFER (number_buffer_op, BUFFER_OP, length_buffer_op*2);
        AL_LINK_BUFFER (number_buffer_op);

        /************************************************/
        /*** fill BUFFER_OP with overdrive pacing data ***/
        /************************************************/

        for (i=0; i<length_buffer_op*2; i++)        /* initialize BUFFER_OP */
                BUFFER_OP[i]=2048;

        for (i = (ch[2]-1)+((length_buffer_op - width[0] - 2)*2);
            /* pulse near the end of the cycle length */
                i < (length_buffer_op - 2)*2; i=i+2)
                BUFFER_OP[i]=2048 + amp[2];

        /******************************************/
        /*** output the overdrive pacing stimuli ***/
        /******************************************/

        if (number_overdrive != 99) {
                CLEAR_COMMAND ();
                outtext(2,19,"Press the space bar to stop the overdrive pacing anytime",1);
                for (i=0; i<number_overdrive; i++) {
                        if ( kbhit() ) break; /* stop overdrive pacing if any key is hit */
                        AL_BURST_DAC ();

                        outtext(65,19," **   ** ",2);
                        outtext(65,20,"**** ****",2);
                        outtext(65,21," ******* ",2);
                        outtext(65,22,"  *****  ",2);
                        outtext(65,23,"    *    ",2);

                        AL_WAIT_FOR_COMPLETION (number_buffer_op);

                        }
        }

        else if (number_overdrive == 99) {
```

```
            CLEAR_COMMAND ();
            outtext(2,19,"Press the space bar to stop overdrive pacing anytime",1);
            while ( !kbhit() ) {
                  AL_BURST_DAC ();

                  outtext(65,19," **    ** ",2);
                  outtext(65,20,"**** ****",2);
                  outtext(65,21," ******* ",2);
                  outtext(65,22,"  *****  ",2);
                  outtext(65,23,"    *    ",2);

                  AL_WAIT_FOR_COMPLETION (number_buffer_op);

            }
      }

      outtext(65,19,"         ",0);
      outtext(65,20,"         ",0);
      outtext(65,21,"         ",0);
      outtext(65,22,"         ",0);
      outtext(65,23,"         ",0);

      AL_UNLINK_BUFFERS ();
      AL_UNDECLARE_BUFFER (number_buffer_op);
}

/****************************************************************************
 *
 * Function --------> TRIGGER(int number_buffer_op, int length_buffer_op
 *                        int number_trigger)
 *
 * Return Type ----> void
 *
 * Description ----> routine to do trigger pacing right after ES or burst
 *
 ****************************************************************************/

void TRIGGER(int number_buffer_t, int length_buffer_t, int number_trigger) {

    int BUFFER_T[100]; /* limits the width of trigger pulse to 50 msec */

      /*** parameter assignments for timer.c   ***/

/**   struct time timep;
      long tmp=0, otmp=0; **/

      /*********************************************/
      /*** declare BUFFER_T for triggered pacing ***/
```

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

        AL_DECLARE_BUFFER (number_buffer_t, BUFFER_T, length_buffer_t*2);
        AL_LINK_BUFFER (number_buffer_t);

/*      AL_SETUP_DAC (2,-1);
        AL_SET_PERIOD (0.001);
*/
        /*******************************************************/
        /*** fill BUFFER_T with triggered pacing stimuli data ***/
        /*******************************************************/

        for (i=0; i<length_buffer_t*2; i++)          /* initialize BUFFER_T */
                BUFFER_T[i]=2048;

        for (i=ch[2]-1; i<(width[0])*2; i=i+2)
                BUFFER_T[i]=2048 + amp[2];

        /*************************************/
        /*** set up output               ***/
        /*************************************/


        AL_SETUP_DAC (2,-1);             /* 2 - external trigger
                                            -1 - output one value to channel 2 */

        AL_SET_PERIOD (0.001);

        /**********************************/
        /*** output the triggered stimuli ***/
        /**********************************/

        if (number_trigger != 99) {
                for (k=0; k<number_trigger; k++) {

                        if ( kbhit() ) break; /* stop overdrive pacing if any key is hit */

                        /*** timer to count 100 msec for trigger lockout.
                             this timer increments in the amount 18.2 Hz (60 msec) ***/

/**                     gettime(&timep);
                        otmp = (timep.ti_sec * 1000) + (timep.ti_hund * 10);
                        while(1) {
                                gettime(&timep);
                                tmp = timep.ti_sec * 1000 + timep.ti_hund * 10;
                                if ((tmp-otmp) >= trig_locko) {
                                        break;
                                }
```

```
                } **/

                inregs.h.ah = 0x86;
                inregs.x.cx = waitval >> 16;
                inregs.x.dx = waitval && 0xffff;
                clock_tics = (int) (((long)trig_locko*(long)1024)/(long)1000);
                for (i=0; i<clock_tics; i++)
                        int86(intnumber,&inregs,&outregs);

                AL_SET_TIMEOUT (0);
                AL_BURST_DAC (number_buffer_t);

                outtext(65,19,"*********",2);
                outtext(65,20,"*********",2);
                outtext(65,21,"    ***    ",2);
                outtext(65,22,"    ***    ",2);
                outtext(65,23,"    ***    ",2);

                AL_WAIT_FOR_COMPLETION (number_buffer_t);

        }
}

else if (number_trigger == 99) {
        while ( !kbhit() ) {

                /*** timer to count 100 msec for trigger lockout.
                   this timer increments in the amount 18.2 Hz (60 msec) ***/

/**             gettime(&timep);
                otmp = (timep.ti_sec * 1000) + (timep.ti_hund * 10); **/

        /*      printf("%d %d %d %d\n", timep.ti_hour,timep.ti_min,
                        timep.ti_sec,timep.ti_hund); */

/**             while(1) {
                        gettime(&timep);
                        tmp = timep.ti_sec * 1000 + timep.ti_hund * 10;
                        if ((tmp-otmp) >= trig_lockout) {
                                break;
                        }
                } **/

        /*      printf("%d %d %d %d\n", timep.ti_hour,timep.ti_min,
                        timep.ti_sec,timep.ti_hund); */

                inregs.h.ah = 0x86;
                inregs.x.cx = waitval >> 16;
```

```
                    inregs.x.dx = waitval && 0xffff;
                    clock_tics = (int) (((long)trig_locko*(long)1024)/(long)1000);
                    for (i=0; i<clock_tics; i++)
                            int86(intnumber,&inregs,&outregs);

                    /* A timeout period 0 causes ATLAB to wait indefinitely */
                    AL_SET_TIMEOUT (0);
                    AL_BURST_DAC (number_buffer_t);

                    outtext(65,19,"*********",2);
                    outtext(65,20,"*********",2);
                    outtext(65,21,"   ***   ",2);
                    outtext(65,22,"   ***   ",2);
                    outtext(65,23,"   ***   ",2);

                    AL_WAIT_FOR_COMPLETION (number_buffer_t);

            }
        }

        outtext(65,19,"         ",0);
        outtext(65,20,"         ",0);
        outtext(65,21,"         ",0);
        outtext(65,22,"         ",0);
        outtext(65,23,"         ",0);

        AL_UNLINK_BUFFERS ();
        AL_UNDECLARE_BUFFER (number_buffer_t);

        /*** at this point, either there was a intracardiac
        triggered output or timeout triggered output  ***/
}

/**************************************************************************
 *
 * Function --------> PACING_PVC (int number_buffer_pvc, int length_buffer_pvc)
 *
 * Return Type -----> void
 *
 * Description ----> routine to produce PVC triggered pacing buffer
 *
 **************************************************************************/

void PACING_PVC (int number_buffer_pvc, int length_buffer_pvc) {

        int BUFFER_PVC[50];

        AL_DECLARE_BUFFER (number_buffer_pvc, BUFFER_PVC, length_buffer_pvc*2);
```

```
        AL_LINK_BUFFER (number_buffer_pvc);

        /*****************************************/
        /*** fill BUFFER_PVC with pacing data ***/
        /*****************************************/

        for (i=0; i<length_buffer_pvc*2; i++)           /* initialize BUFFER_PVC */
                BUFFER_PVC[i]=2048;

        for (i=ch[2]-1; i<width[2]*2; i=i+2)
                BUFFER_PVC[i]=2048+amp[2];

        /***************************************/
        /*** set up output                 ***/
        /***************************************/

        AL_SETUP_DAC (0,-1);            /* 0 - internal clock
                                           -1 - output two values to channels 1 & 2 */

        AL_SET_PERIOD (0.001);

        /***************************************/
        /*** output the pacing stimuli       ***/
        /***************************************/

        AL_BURST_DAC ();

        outtext(53,19,"********  ",2);
        outtext(53,20,"**      ** ",2);
        outtext(53,21,"********  ",2);
        outtext(53,22,"**         ",2);
        outtext(53,23,"**         ",2);

        AL_WAIT_FOR_COMPLETION (number_buffer_pvc);

        outtext(53,19,"           ",0);        /* to erase the P */
        outtext(53,20,"           ",0);
        outtext(53,21,"           ",0);
        outtext(53,22,"           ",0);
        outtext(53,23,"           ",0);

        AL_UNLINK_BUFFERS ();
        AL_UNDECLARE_BUFFER (number_buffer_pvc);
}

/**************************************************************************
 *
 * Function -------> PACING_NOPVC (int number_buffer_nopvc, int length_buffer_nopvc)
```

```
 *
 * Return Type ----> void
 *
 * Description ----> routine to produce no PVC pacing buffer
 *
 ****************************************************************************/

void PACING_NOPVC (int number_buffer_nopvc, int length_buffer_nopvc) {

        int BUFFER_NOPVC[100];

        AL_DECLARE_BUFFER (number_buffer_nopvc, BUFFER_NOPVC, length_buffer_nopvc*2);
        AL_LINK_BUFFER (number_buffer_nopvc);

        /*******************************************/
        /*** fill BUFFER_NOPVC with pacing data ***/
        /*******************************************/

        for (i=0; i<length_buffer_nopvc*2; i++)      /* initialize BUFFER_PVC */
                BUFFER_NOPVC[i]=2048;

        for (i=ch[0]-1; i<width[0]*2; i=i+2)
                BUFFER_NOPVC[i]=2048+amp[0];

        /**************************************/
        /*** set up output                 ***/
        /**************************************/

        AL_SETUP_DAC (0,-1);          /* 0 - internal clock
                                         -1 - output two values to channels 1 & 2 */

        AL_SET_PERIOD (0.001);

        /**************************************/
        /*** output the pacing stimuli      ***/
        /**************************************/

        AL_BURST_DAC ();

        outtext(53,19,"**    *** ",2);
        outtext(53,20,"**   **** ",2);
        outtext(53,21,"** ** ** ",2);
        outtext(53,22,"****  ** ",2);
        outtext(53,23,"***    ** ",2);

        AL_WAIT_FOR_COMPLETION (number_buffer_nopvc);

        outtext(53,19,"           ",0);      /* to erase the N */
```

```
        outtext(53,20,"            ",0);
        outtext(53,21,"            ",0);
        outtext(53,22,"            ",0);
        outtext(53,23,"            ",0);

        AL_UNLINK_BUFFERS ();
        AL_UNDECLARE_BUFFER (number_buffer_nopvc);
}


/****************************************************************************
 *
 * Function --------> BURST(int number_buffer_b, int length_buffer_b)
 *
 * Return Type -----> void
 *
 * Description -----> routine to produce burst train for VT induction
 *
 ****************************************************************************/

void BURST(int number_buffer_b, int length_buffer_b) {

        int BUFFER_E[5000];

        /********************************************/
        /*** declare BUFFER_E for extra stimuli ***/
        /********************************************/

        AL_DECLARE_BUFFER (number_buffer_b, BUFFER_E, length_buffer_b*2);
        AL_LINK_BUFFER (number_buffer_b);

        /**********************************************/
        /*** fill BUFFER_E with extra stimuli data ***/
        /**********************************************/

        for (i=0; i < (length_buffer_b*2); i++)  /* initialize BUFFER_E */
                BUFFER_E[i] = 2048;

        for (j=(ch[1]-1); j < (width[0]*2); j=j+2)  /* pacing pulse before es */
                BUFFER_E[j] = 2048 + amp[1];

        accu = width[0] + delays[0];

        while (accu < (delays[0] + delays[1])) {              /* burst train */
                for (j = (ch[1]-1) + (accu*2); j < (accu+width[1])*2; j=j+2)
                        BUFFER_E[j] = 2048 + amp[1];
                accu = accu + width[1] + delays[2];
        }
```

```
        for (j=(length_buffer_b-2)*2; j<length_buffer_b*2; j=j+1) /* to avoid DC shock */
                BUFFER_E[j] = 2048;

        /**********************************/
        /*** output the extra stimuli ***/
        /**********************************/

        AL_BURST_DAC ();

        AL_WAIT_FOR_COMPLETION (number_buffer_b);

        AL_UNLINK_BUFFERS ();
        AL_UNDECLARE_BUFFER (number_buffer_b);
}
```

Appendix B

Video.c

```
/*************************************************************************
 *
 * File name --> video.c
 * Date --------> 8/23/93
 * Purpose -----> Routine to do video opearations for the stimulator
 *
 *************************************************************************/

#include "video.h"

/*************************************************************************
 *
 *   Global parameter declarations
 *
 *************************************************************************/

unsigned char far *screenptr = (unsigned char far *)(SCREEN_SEG * 0x10000);
                                        /* pointer to text screen memory */

/*************************************************************************
 *
 * Function --------> CLEAR_SCREEN()
 *
 * Return Type -----> void
 *
 * Description -----> routine to initialize screen attributes
 *
 *************************************************************************/

void CLEAR_SCREEN() { /* Initializes the screen attributes */
```

```
        int i=0;
        while(i < (MAXX*MAXY<<1)) {
                screenptr[i++] = ' ';
                screenptr[i++] = FOREGROUND_WHITE|BACKGROUND_BLACK;
        }
}

/***************************************************************************
 *
 * Function --------> SETUP_SCREEN()
 *
 * Return Type ----> void
 *
 * Description ----> routine to draw boundaries
 *
 ***************************************************************************/

void SETUP_SCREEN() { /* outside boundary */

        int j;
        char tmpstr[2];
        char *tstr;

        /*** four corners ***/
        tmpstr[1]=0;
        tmpstr[0]=218;
        outtext(0,0,tmpstr,0);
        tmpstr[0]=192;
        outtext(0,24,tmpstr,0);
        tmpstr[0]=217;
        outtext(79,24,tmpstr,0);
        tmpstr[0]=191;
        outtext(79,0,tmpstr,0);

        /*** horizontal lines at y = 0,2,18,24 ***/
        tmpstr[0]=196;
        for (j=1; j<79; j++) {
                outtext(j,0,tmpstr,0);
                outtext(j,2,tmpstr,0);
                outtext(j,18,tmpstr,0);
                outtext(j,24,tmpstr,0);
        }
/*      for (j=35; j<79; j++) {
                outtext(j,3,tmpstr,0);
                outtext(j,10,tmpstr,0);
        } */
        for (j=1; j<34; j++)
                outtext(j,14,tmpstr,0);
```

```
	/*** vertical lines at x = 0,17,34,79 ***/
	tmpstr[0]=179;
	for (j=1; j<24; j++) {
		outtext(0,j,tmpstr,0);
		outtext(79,j,tmpstr,0);
	}
	for (j=3; j<14; j++)
		outtext(17,j,tmpstr,0);
	for (j=3; j<18; j++)
		outtext(34,j,tmpstr,0);

	tmpstr[0]=195;	/* |- */
	outtext(0,2,tmpstr,0);
	outtext(0,14,tmpstr,0);
	outtext(0,18,tmpstr,0);
/*	outtext(34,3,tmpstr,0);
	outtext(34,10,tmpstr,0); */
	outtext(34,14,tmpstr,0);

	tmpstr[0]=180;	/* -| */
	outtext(79,2,tmpstr,0);
/*	outtext(79,3,tmpstr,0);
	outtext(79,10,tmpstr,0); */
	outtext(79,18,tmpstr,0);
	outtext(34,14,tmpstr,0);

	tmpstr[0]=194;	/* T */
	outtext(17,2,tmpstr,0);
	outtext(34,2,tmpstr,0);

	tmpstr[0]=193;	/* reverse T */
	outtext(17,14,tmpstr,0);
	outtext(34,18,tmpstr,0);

	/*** headings ***/
    tstr="P R O G R A M M A B L E   S T I M U L A T O R   S Y S T E M";
	outtext(12,0,tstr,1);
	tstr="PACING";
	outtext(2,2,tstr,1);
	tstr="PREMATURE";
	outtext(19,2,tstr,1);
	tstr="OVERDRIVE";
	outtext(36,2,tstr,1);
/*	tstr="WAVEFORM";
	outtext(52,2,tstr,1);
	tstr="CH 1";
	outtext(36,3,tstr,1);
```

```
tstr="CH 2";
outtext(36,10,tstr,1); */
tstr="COMMAND";
outtext(36,18,tstr,1);

/*** PACING panel ***/
tstr="CH   =";
outtext(2,3,tstr,0);
tstr="AMP  =";
outtext(2,4,tstr,0);
tstr="STEP =";
outtext(2,5,tstr,0);
tstr="WIDTH=";
outtext(2,6,tstr,0);
tstr="CYCL =";
outtext(2,9,tstr,0);
tstr="STEP =";
outtext(2,10,tstr,0);

/*** PREMATURE panel ***/
tstr="CH   =";
outtext(19,3,tstr,0);
tstr="AMP  =";
outtext(19,4,tstr,0);
tstr="STEP =";
outtext(19,5,tstr,0);
tstr="WIDTH=";
outtext(19,6,tstr,0);
tstr="EX_ST=";
outtext(19,8,tstr,0);
tstr="D1   =";
outtext(19,9,tstr,0);
tstr="D2   =";
outtext(19,10,tstr,0);
tstr="D3   =";
outtext(19,11,tstr,0);
tstr="D4   =";
outtext(19,12,tstr,0);
tstr="STEP =" ;
outtext(19,13,tstr,0);

/*** OVERDRIVE panel ***/

tstr="CH   =";
outtext(36,3,tstr,0);
tstr="AMP  =";
outtext(36,4,tstr,0);
tstr="WIDTH=";
```

```
        outtext(36,6,tstr,0);

        tstr="MODE 2 & 5";
        outtext(36,8,tstr,1);
        tstr="OD    =";
        outtext(36,9,tstr,0);
        tstr="DOD   =";
        outtext(36,10,tstr,0);

        tstr="MODE 3 & 6";
        outtext(36,11,tstr,1);
        tstr="TRIG =";
        outtext(36,12,tstr,0);
        tstr="LOCKO=";
        outtext(36,13,tstr,0);

        tstr="MODE 7";
        outtext(53,8,tstr,1);
        tstr="LOCK_E =";
        outtext(53,9,tstr,0);
        tstr="LOCK_O =";
        outtext(53,10,tstr,0);
        tstr="LOCK_B =";
        outtext(53,11,tstr,0);
        tstr="WINDOW =";
        outtext(53,12,tstr,0);
        tstr="DEMAND =";
        outtext(53,13,tstr,0);
}

/***************************************************************************
 *
 * Function --------> CLEAR_COMMAND()
 *
 * Return Type -----> void
 *
 * Description -----> routine to clear command panel
 *
 ***************************************************************************/

void CLEAR_COMMAND() {

        char *tstr;

        tstr="                                                                              ";
        outtext(2,19,tstr,0);
        outtext(2,20,tstr,0);
        outtext(2,21,tstr,0);
```

```
        outtext(2,22,tstr,0);
        outtext(2,23,tstr,0);
}

/****************************************************************************
 *
 * Function --------> MAIN_MENU()
 *
 * Return Type -----> void
 *
 * Description -----> routine to put protocols menu in command panel
 *
 ****************************************************************************/

void MAIN_MENU() {

        char *tstr;

     tstr="Select the protocol:     ";
        outtext(2,19,tstr,0);
        tstr="f1  Triggered Multi-Site Pacing";
        outtext(2,20,tstr,0);
     tstr="f2 ";
        outtext(2,21,tstr,0);
     tstr="f3 ";
        outtext(2,22,tstr,0);
     tstr="f4 ";
        outtext(2,23,tstr,0);
}

/****************************************************************************
 *
 * Function --------> PROTOCOL1_MENU()
 *
 * Return Type -----> void
 *
 * Description -----> routine to put protocol 1 menu in command panel
 *
 ****************************************************************************/

void PROTOCOL1_MENU() {

        char *tstr;

        tstr="f1      select mode and channel";
        outtext(2,19,tstr,0);
     tstr="f2      change output characteristics";
        outtext(2,20,tstr,0);
```

```
    tstr="cursor change output characteristics";
      outtext(2,21,tstr,0);
    tstr="f10    start pacing";
      outtext(2,22,tstr,0);
    tstr="esc    end the program";
      outtext(2,23,tstr,0);
}

/****************************************************************************
 *
 * Function --------> PROTOCOL1_1_MENU()
 *
 * Return Type ----> void
 *
 * Description ----> routine to put protocol 1.1 menu in command panel
 *
 ****************************************************************************/

void PROTOCOL1_1_MENU() {

        char *tstr;

        tstr="f1     select mode";
        outtext(2,19,tstr,0);
        tstr="f2     select channel";
        outtext(2,20,tstr,0);
/*      tstr="f3     select premature stimuli";
        outtext(2,21,tstr,0);
        tstr="f4     select overdrive";
        outtext(2,22,tstr,0); */
        tstr="esc    exit this level";
        outtext(2,21,tstr,0);
}

/****************************************************************************
 *
 * Function --------> PROTOCOL1_1_1_MENU()
 *
 * Return Type ----> void
 *
 * Description ----> routine to put protocol 1.1.1 menu in command panel
 *
 ****************************************************************************/

void PROTOCOL1_1_1_MENU () {

        char *tstr;
```

```
        tstr="f1  Mode1 (ES & NO)             f4  Mode4 (BURST & NO)    ";
        outtext(2,19,tstr,0);
        tstr="f2  Mode2 (ES & FIXED)          f5  Mode5 (BURST & FIXED)";
        outtext(2,20,tstr,0);
        tstr="f3  Mode3 (ES & TRIG)           f6  Mode6 (BURST & TRIG) ";
        outtext(2,21,tstr,0);
        tstr="f7  Mode7 (ES & PVC TRIG)                                ";
        outtext(2,22,tstr,0);
}
/*************************************************************************
 *
 * Function --------> PROTOCOL1_1_2_MENU()
 *
 * Return Type -----> void
 *
 * Description -----> routine to put protocol 1.1.2 menu in command panel
 *
 *************************************************************************/

void PROTOCOL1_1_2_MENU () {

        char *tstr;

        tstr="f1      Pacing channel";
        outtext(2,19,tstr,0);
        tstr="f2      Extra stimuli channel";
        outtext(2,20,tstr,0);
        tstr="f3      Overdrive channel";
        outtext(2,21,tstr,0);
}

/*************************************************************************
 *
 * Function --------> PROTOCOL1_1_2_1_MENU()
 *
 * Return Type -----> void
 *
 * Description -----> routine to put protocol 1.1.2.1 menu in command panel
 *
 *************************************************************************/

void PROTOCOL1_1_2_1_MENU () {

        char *tstr;

        tstr="f1      Channel 1";
        outtext(2,19,tstr,0);
        tstr="f2      Channel 2";
```

```
	outtext(2,20,tstr,0);
}

/* PROTOCOL1_1_3_MENU () and PROTOCOL1_1_4_MENU () are obsolete */

/*************************************************************************
 *
 * Function --------> PROTOCOL1_1_3_MENU()
 *
 * Return Type ----> void
 *
 * Description ----> routine to put protocol 1.1.3 menu in command panel
 *
 *************************************************************************/

void PROTOCOL1_1_3_MENU() {

	char *tstr;

/*	tstr="f1	None";
	outtext(2,19,tstr,0);
*/	tstr="f1	Extra Stimuli";
	outtext(2,19,tstr,0);
	tstr="f2	Burst";
	outtext(2,20,tstr,0);
}

/*************************************************************************
 *
 * Function --------> PROTOCOL1_1_4_MENU()
 *
 * Return Type ----> void
 *
 * Description ----> routine to put protocol 1.1.4 menu in command panel
 *
 *************************************************************************/

void PROTOCOL1_1_4_MENU() {

	char *tstr;

	tstr="f1	None";
	outtext(2,19,tstr,0);
	tstr="f2	Fixed rate";
	outtext(2,20,tstr,0);
	tstr="f3	Triggered";
	outtext(2,21,tstr,0);
}
```

```
/**************************************************************************
 *
 * Function --------> PROTOCOL1_4_MENU (int mode)
 *
 * Return Type -----> void
 *
 * Description -----> routine to put protocol 1.4 menu in command panel
 *
 **************************************************************************/

void PROTOCOL1_4_MENU (int mode) {

        char *tstr;

        tstr="f1     introduce extra stimuli & overdrive";
        outtext(2,19,tstr,0);
        tstr="cursor change output characteristics";
        outtext(2,20,tstr,0);
        tstr="f10    stop pacing & exit to protocol 1";
        outtext(2,21,tstr,0);

        if (mode == 1) {
                tstr="Mode 1:    Extra stimuli only";
                outtext(2,23,tstr,0);
        }
        else if (mode == 2) {
                tstr="Mode 2:    Extra stimuli and fixed rate overdrive";
                outtext(2,23,tstr,0);
        }
        else if (mode == 3) {
                tstr="Mode 3:    Extra stimuli and triggered pacing";
                outtext(2,23,tstr,0);
        }
        else if (mode == 4) {
                tstr="Mode 4:    Burst only";
                outtext(2,23,tstr,0);
        }
        else if (mode == 5) {
                tstr="Mode 5:    Burst and fixed rate overdrive";
                outtext(2,23,tstr,0);
        }
        else if (mode == 6) {
                tstr="Mode 6:    Burst and triggered pacing";
                outtext(2,23,tstr,0);
        }
        else if (mode == 7) {
                tstr="Mode 7:    Extra Stimuli and PVC triggered pacing";
```

```
                outtext(2,23,tstr,0);
        }
}

/*************************************************************************
 *
 * Function --------> PROTOCOL2_MENU()
 *
 * Return Type -----> void
 *
 * Description -----> routine to put protocol 2 menu in command panel
 *
 *************************************************************************/

void PROTOCOL2_MENU() {

        char *tstr;

        tstr="f1 induce VT by extra stimuli w/ no overdrive pacing";
        outtext(2,19,tstr,0);
        tstr="f2 induce VT by extra stimuli w/ fixed overdrive pacing";
        outtext(2,20,tstr,0);
        tstr="f3 induce VT by extra stimuli w/ triggered overdrive pacing";
        outtext(2,21,tstr,0);
     /* need to put induce VT by burst w/ no overdrive pacing */
        tstr="f4 induce VT by burst w/ fixed overdrive pacing";
        outtext(2,22,tstr,0);
        tstr="f5 induce VT by burst w/ fixed overdrive pacing";
        outtext(2,23,tstr,0);
}

/*************************************************************************
 *
 * Function --------> outtext(int x_coord, int y_coord, char *str, int video_flg)
 *
 * Return Type -----> void
 *
 * Description -----> routine to write characters
 *
 *************************************************************************/

void outtext(int x_coord, int y_coord, char *str, int video_flg) {

        int off=offset(x_coord, y_coord);

        if (video_flg == 0) {
                while (*str != '\0'){
                        screenptr[off+1]=
```

```
                        FOREGROUND_GREEN|BACKGROUND_BLACK;
                screenptr[off] = *str++;
                off += 2;
            }
        }

        else if (video_flg == 1) {
            while (*str != '\0') {
                screenptr[off+1]=
                        FOREGROUND_BLACK|BACKGROUND_GREEN;
                screenptr[off] = *str++;
                off += 2;
            }
        }

        else if (video_flg ==2) {
            while (*str != '\0') {
                screenptr[off+1]=
                        FOREGROUND_RED|BACKGROUND_BLACK|BLINKING;
                screenptr[off] = *str++;
                off += 2;
            }
        }
}

/* This main program is used for testing.

main() {
        int x, y;
        init_screen();
        for(y=0;y<MAXY;y++) {
            for(x=0;x<MAXX;x++) {
                if ((x==10) && (y==0)) {
                        screenptr[offset(x,y)+1] =
                        BLINKING|FOREGROUND_BLACK|BACKGROUND_WHITE;
                }
                screenptr[offset(x,y)] = 'B';
            }
        }
}

*/
```

Bibliography

[1] F. Al-Nasser. Tables speed design of low-pass active filters. *EDN*, pages 23–32, 1971.

[2] N. Barkakati. *The Waite Group's Essential Guide to Turbo C.* Howard W. Sams and Company, Indianapolis, first edition, 1989.

[3] S. S. Barold and D. P. Zipes. Cardiac pacemakers and antiarrhythmic devices. In E. Braunwald, editor, *Heart Diseases: A Textbook of Cardiovascular Medicine*, chapter 25. W.B. Saunders Company, Philadelphia, 1992.

[4] R. M. Berne and M. N. Levy. *Cardiovascular Physiology.* Mosby-Year Book, Inc., St. Louis, sixth edition, 1992.

[5] H. B. Burchell and J. Merideth. Management of cardiac tachyarrhythmias with cardiac pacemakers. *Annals of the New York Academy of Sciences*, 167:546–556, October 1969.

[6] M. N. Collins and G. E. Billman. Autonomic response to coronary occlusion in animals susceptible to ventricular fibrillation. *American Journal of Physiology*, 257:H1886–94, 1989.

[7] P. F. Cranefield and R. S. Aronson. *Cardiac Arrhythmias: The Role of Triggered Activity and Other Mechanisms.* Futura Publishing Company, Inc, Mount Kisco, New York, 1988.

[8] B. P. Damiano and M. R. Rosen. Effects of pacing on triggered activity induced by early afterdepolarizations. *Circulation*, 69:1013–1025, 1984.

[9] Data Translation, Inc. *User Manual for ATLAB*, 1988.

[10] Data Translation, Inc. *User Manual for DT2821 Series*, ninth edition, 1989.

[11] D. S. Echt, J. C. Griffin, A. J. Ford, J. W. Knutti, R. C. Feldman, and J. W. Mason. Nature of inducible ventricular tachyarrhythmias with different electrophysiologic characteristics and different mechanisms in the infarcted canine heart. *American Journal of Cardiology*, 52:1127–1132, 1983.

[12] N. El-Sherif, M. Restivo, W. Craelius, R. Mehra, R. Henkin, E. B. Caref, and G. Kelen. The high-resolution electrocardiogram: Technical and basic aspects. In El-Sherif N and Samet P, editors, *Cardiac Pacing and Electrophysiology*, chapter 19. W. B. Saunders Company, Philadelphia, 1991.

[13] R. A. Freedman and J. W. Mason. Sustained ventricular tachycardia: Clinical aspects. In El-Sherif N and Samet P, editors, *Cardiac Pacing and Electrophysiology*, chapter 13. W. B. Saunders Company, Philadelphia, 1991.

[14] S. Furman and D. J. W. Escher. *Principles and Techniques of Cardiac Pacing.* Harper and Row, Publishers, Inc., Mount Kisco, New York, fifth edition, 1970.

[15] A. H. Harken. Surgical treatment of cardiac arrhythmias. *Scientific American*, pages 68–74, July 1993.

[16] H.. Hirche and F. M. McDonald. New aspects on the pathogenesis of ischemia-induced ventricular arrhythmias. In D. W. Behrenbeck, E. Sowton, G. Fontaine, and U. J. Winter, editors, *Cardiac Pacemakers*, pages 170–177. Steinkopff Verlag Darmstadt, 1985.

[17] M. E. Josephson and C. D. Gottlieb. Ventricular tachycardia associated with coronary artery disease. In Zipes E, editor, *From Cell to Bedside*, chapter 63. W.B. Saunders Company, Philadelphia, 1989.

[18] W. Kaltenbrunner, R. Cardinal, M. Dubuc, M. Shenasa, R. Nadeau, G. Tremblay, M. Vermeulen, P. Savard, and P. Page. Epicardial and endocardial mapping of ventricular tachycardia in patients with myocardial infarction. *Circulation*, 84(3):1058–1071, 1991.

[19] Kepco. *Bipolar Operational Power Supplies Instruction Manual*, 1979.

[20] D. A. Kirby, S. Hottinger, S. Ravid, and B. Lown. Inducible monomorphic sustained ventricular tachycardia in the conscious pig. *American Heart Journal*, 119(5):1042–1049, 1990.

[21] K.-H. Kuck, K.-P. Kunze, M. Schuter, and W. Bleifeld. Single-beat stimulation and train of stimuli method for prevention of reentrant tachycardia. In D. W.

Behrenbeck, E. Sowton, G. Fontaine, and U. J. Winter, editors, *Cardiac Pacemaker*, pages 282–290. Steinfopff Verlag Darmstadt, Springer-Verlag New York, 1985.

[22] J. W. Mason, K. P. Anderson, and R. A. Freedman. Techniques and criteria in electrophysiologic study of ventricular tachycardia. *Circulation*, 75:III125, 1987.

[23] E. N. Moore and J. F. Spear. Electrophysiologic studies on the initiation, prevention, and termination of ventricular fibrillation. In *Cardiac Electrophysiology and Arrhythmias*, chapter 35. Grune & Stratton, 1985.

[24] R. J. Myerburg and A. Castellanos. Cardiac arrest and sudden cardiac death. In E. Braunwald, editor, *Heart Diseases: A Textbook of Cardiovascular Medicine*, chapter 26. W.B. Saunders Company, Philadelphia, 1992.

[25] F. H. Netter. *Heart, The CIBA Collection of Medical Illustrations*, volume 5. CIBA, 1981.

[26] S. L. Robbins, R. S. Cotran, and V. Kumar. *Pathologic Basis of Disease*. W. B. Saunders Company, Philadelphia, third edition, 1984.

[27] J. F. Spear, E. M. Moore, and L. N. Horowitz. Effect of current pulses delivered during the ventricular vulnerable period upon the ventricular fibrillation threshold. *American Journal of Cardiology*, 32:814–822, 1973.