UNIVERSITY OF ÇUKUROVA
INSTITUTE OF NATURAL AND APPLIED SCIENCE

MSc THESIS

Fırat KUMRU

REAL TIME MONITORING OF ELECTROCARDIOGRAPHY AND BLOOD VOLUME USING RTAI ON LINUX

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA, 2006

ÇUKUROVA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

REAL TIME MONITORING OF ELECTROCARDIOGRAPHY AND BLOOD VOLUME USING RTAI ON LINUX

Fırat KUMRU

YÜKSEK LİSANS TEZİ

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez / / Tarihinde Aşağıdaki Jüri Üyeleri Tarafından Oybirliği İle Kabul Edilmiştir.

İmza.....……	İmza.....……	İmza.....……
Yrd.Doç. Dr. Sami ARICA	Prof. Dr. Mehmet TÜMAY	Yrd. Doç. Dr. Turgay İBRİKÇİ
DANIŞMAN	ÜYE	ÜYE

Bu tez Enstitümüz Elektrik Elektronik Mühendisliği Anabilim Dalında hazırlanmıştır.

Kod No:

Prof. Dr. Aziz ERTUNÇ
Enstitü Müdürü

Bu çalışma Ç.Ü. Bilimsel Araştırma Projeleri Birimi tarafından desteklenmiştir.

Proje No: MMF2004YL58

ÖZ

YÜKSEK LİSANS TEZİ

REAL TIME LINUX KULLANARAK ELEKTROKARDİYOGRAFİ VE KAN HACMİNİN GERÇEK ZAMANLI İZLENMESİ

Fırat KUMRU

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

FEN BİLİMLERİ ENSTİTÜSÜ

ÇUKUROVA ÜNİVERSİTESİ

Danışman: **Yrd. Doç. Dr. Sami ARICA**
Yıl: **Eylül, 2006 Sayfa: 63**
Jüri: **Yrd. Doç. Dr. Sami ARICA**
Prof. Dr. Mehmet TÜMAY
Yrd. Doç. Dr. Turgay İBRİKÇİ

Bu çalışmada, elektrokardiyografi (EKG) sinyalleri ve Kan Basıncı (KB) sinyalleri alınarak Linux Real Time Application Interface (RTAI) ile okunarak izlenmiştir.

Real Time Application Interface Linux işletim sistemi tarafından sağlanan , Linux' a gerçek zamanlı özellik kazandıran bir uygulamadır. EKG ve KB sinyalleri gürültüden temizlenmek ve yükseltilmek amacıyla yükseltici ve filtre sistemine uygulnmıştır. Temizlenen sinyaller 12 bit 1000 Hz Advantech veri toplama kartı ile bilgisayar ortamına aktarılmıştır.

Geliştirilen yazılım ile sistolik ve diastolik kan basınç değerleri, EKG sinyalinin tepe nokta değerlerinin aralıkları gözlemlenebilmektedir. Yazılım RTAI fonsiyonları ve C fonksiyonlarından oluşturulmuştur.

Anahtar Kelimeler: Gerçek Zamanlı, Veri Toplama, Linux Elektrokardiyogram, Kan Basıncı

ABSTRACT

MSc THESIS

REAL TIME MONITORING OF ELECTROCARDIOGRAPHY AND BLOOD VOLUME USING RTAI ON LINUX

Fırat KUMRU

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

INSTITUTE OF NATURAL AND APPLIED SCIENCES

UNIVERSITY OF ÇUKUROVA

Supervisor: **Assist. Prof. Dr. Sami ARICA**
Year: **September 2006, Pages: 63**
Jury: **Assist. Prof. Dr. Sami ARICA**
Prof. Dr. Mehmet TÜMAY
Assist. Prof. Dr. Turgay İBRİKÇİ

In this study, Real Time reading and monitoring of Electrocardiography (ECG) and Blood Pressure (BP) have been implemented with the Real Time Application Interface (RTAI) on Linux operating system.

The Real Time Application Interface supported by linux is used to provide real time concept. The installation and running of RTAI are also explained.

The ECG and BP signals have been applied to instrumentation amplifiers and filters to clear and to amplify the signal. This clear signal is acquired by an Advantech data acquisition card which has 12 bit resolution at 1000 Hz.

Software has been developed for analyzing signals. The changes of systolic and diastolic pressure and the R points interval of ECG have been observed. The baroreceptor sensitivity (BRS) has been calculated. The RTAI functions and C programming language have been used in the software.

Keywords: Linux, Real Time Application Interface, Electrocardiogram, Blood Pressure, Data Acquisition

ACKNOWLEDGEMENTS

The subject of this thesis was suggested by my supervisor, Assoc. Prof. Dr. Sami ARICA to whom I would like to express my heartfelt thanks for his supervision, guidance, encouragements and extremely useful suggestions throughout this thesis.

I would like to thank Dr. Fırat INCE who has given opinions during the thesis from beginning to the end and also for his supports and friendship. I would like to thank him again especially for introducing me to scientific life.

I would like to express my appreciation to Prof. Dr. Süleyman GÜNGÖR, head of the Department of Electrical and Electronics Engineering, providing materials and study environment.

I would like to thank Prof. Dr. Mehmet TÜMAY, head of the Computer Engineering Department, for his valuable suggestions and endless support.

Finally, I would like to thank my fiance Fatma ÇEKMEZ and my family for their help, moral support and patience.

CONTENTS **PAGE**

LIST OF FIGURES

LIST OF ABBREVIATIONS

RTAI	Real Time Application Inteface
RT	Real Time
RTOS	Real Time Operating System
ECG	Electrocardiogram
BP	Blood Pressure
HR	Heart Rate
BRS	Baroreflex Sensitivity

1. INTRODUCTION

Linux operating system is maintaining to gain popularity in research and student communities, and also in the business world (Sarolathi P., 2001). Linux is a multitasking system, which provides fair, non-preemptive scheduling among several dynamically created and deleted processes. The disadvantage of Linux is that it can not guarantee response times for its processes. However, there are the application areas which require real-time response, such as robotic devices, computers used in health care and military, and various embedded systems used in different kinds of devices.

Real Time (RT) is a software system in which the inputs represent digital data from hardware or other software system's, and the outputs are digital data that control external hardware. The time between the presentation of a set of inputs and the appearance of all the associated outputs is called the **response time.** A RT system is one that must satisfy explicit bounded response time constraints to avoid failure. Briefly, RT provides the response within finite and specified interval.

The Real-Time Application Interface (RTAI) provides hard real-time capabilities in a Linux environment. It adds a small real-time kernel below the standard Linux kernel and treats the Linux kernel as a low priority real-time task. RTAI provides a large selection of inter-process communication mechanisms and other real-time services. Additionally, RTAI provides a LXRT (User Interface Module for RTAI-LINUX) module for easy development of real-time applications in user space.

Real-time Linux provides the capability of running special real-time tasks and interrupt handlers on the same machine as standard Linux. These tasks and interrupt handlers execute whenever they need to, regardless of what Linux is doing. Real-time Linux extends the Standard Linux Programming Environment to real-time problems. Real-time Linux tasks and interrupt handlers can communicate with ordinary Linux processes through a device interface or shared memory. So the exact response time can be obtained for each process. The Real Time Applications for health care are our main concept.

As the population of human grow, the need for health care increases. In recent years, the progress in medical care has been rapid, especially in such fields as cardiology. As the importance of the electronic systems increases, the technology of the integrated measurement systems has gain essential role in studies and industrial areas. The most important criteria are accurate measurement of the signals which are supplied by body and response time characteristics of the system. All of these articles can be achieved by RTAI.

Electrocardiogram (ECG) has vital role to detect any heart health state. The ECG represents the heart beat electrical wave. For this reason, the ECG carries information about the heart.

The Heart Rate (HR) and the Blood Pressure (BP) that are the effect of heart beat are essential parameters for human health. These parameters which are measured must have values in nominal range for healthy life. A heart beat creates electrical signals. The Electrical signal called as **biosignals** at different levels of potentials that human has is acquired by data acquisition systems. The measurement and monitoring these signals can provide information about the patient‘s health state.

The analysis of the HR and BP which are two main parameters provide to obtain Baroreflex Sensitivity (BRS). It controls heart rate to keep the blood pressure in nominal levels for life conditions. It is known that the sensitivity of baroreceptors changes with cardiac pathologies (Ince, 2002). BRS is an important parameter for detection of heart attack.

The measured signals are applied to a biosignal amplifier and filtered to remove noise. Then the clear and amplified signal is applied to computer using a data acquisition card. The acquired signal is captured and analyzed by code blocks under RTAI – Linux operating system environment.

The content of the thesis is arranged as follows: After this introductory chapter, Chapter 2 defines the ECG and BP signals and measurement methods are introduced.

In Chapter 3, the electronic part of the system including amplifier and filters are presented. Also, the amplifying basics and filtering basics are told.

In Chapter 4, the data acquisition part of the system and PCI-818 DAQ card is introduced.

In Chapter 5, the Real Time Application Interface installation and the basics of the RTAI are presented. And also the challenges of the RTAI are explained.

At the end, the conclusion and the future work are explained.

2. ELECTROCARDIOGRAM AND BLOOD PRESSURE

2.1. Electrocardiogram (ECG)

The Heart is the pump of the body. It contains muscle fibers to control the pumping process and cardiac activities. These activities form an electrical potential that is measured by replacing electrodes to the parts of body.

The heart has four chambers that the two upper chambers, left and right atria and the two lower chambers are the ventricles. The right atrium receives the blood from the veins of the body and pumps it in to the right ventricle. The right ventricles pumps the blood trough the lungs, where the blood is cleaned. The oxygenated blood enters to the left atrium that pumps it to the left ventricle. The left ventricle sends the blood into arteries to circulate throughout the body. This cycle is called as *cardiac cycle*.

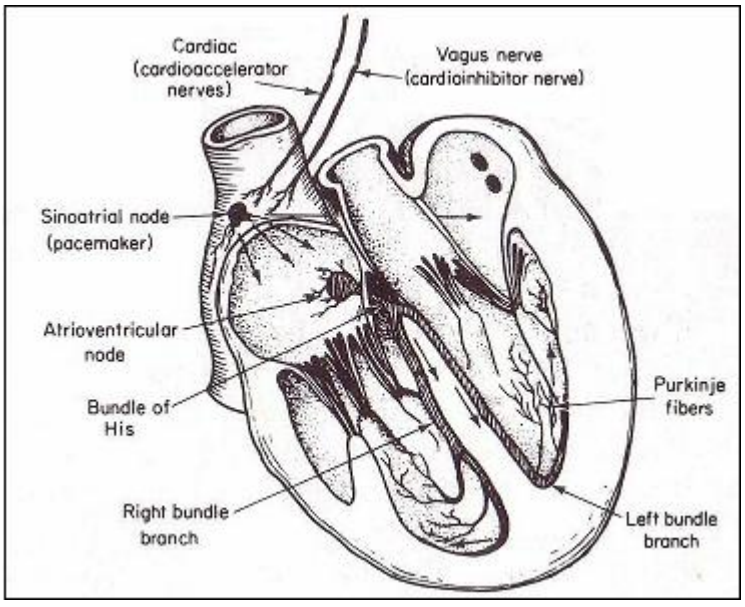

Figure 2.1.a The Heart (Evans. 1971)

The heart has a point that generates action potential regularly. This point is called as *pacemaker* or *sinoatrial* (SA) node. To initiate the heart beat, the action potential is produced by SA and is propagated in all directions. The wave is

terminated at a point near the center of the heart, called *antriventricular* (AV) node. Some special fibers act as a delay line to provide the timing between the action of atria and ventricles. The electrical excitation passes trough the delay line, it is rapidly spread to all parts of both ventricles by *bundle of His* (Cromwell L., Weibell F., Pfeiffer E.A, 1980). The fibers in bundle are called as *Purkinje Fibers,* divides into two branches to initiate action potentials simultaneously in the powerful musculature of the two ventricles.

When the heart pumps, the cell wall offers greater permeability and an excess of sodium is able to flow inside the cell. When the sodium flows into the cell there is no longer a negative potential with respect to the outside. This is known as depolarization. Eventually, when the excitation is completed, the cell depolarizes, and the potential returns to a negative one (Carr, J.J., Brown, J.M., 1981).

The graphic recording or display of the time variant voltages produced by the myocardium during the cardiac cycle is defined as *electrocardiogram*. The P, QRS, and the T waves reflect the rhythmic electrical depolarization and repolarization of myocardium associated with the atria and ventricles. The ECG is used clinically in diagnosing various diseases and conditions of the heart.

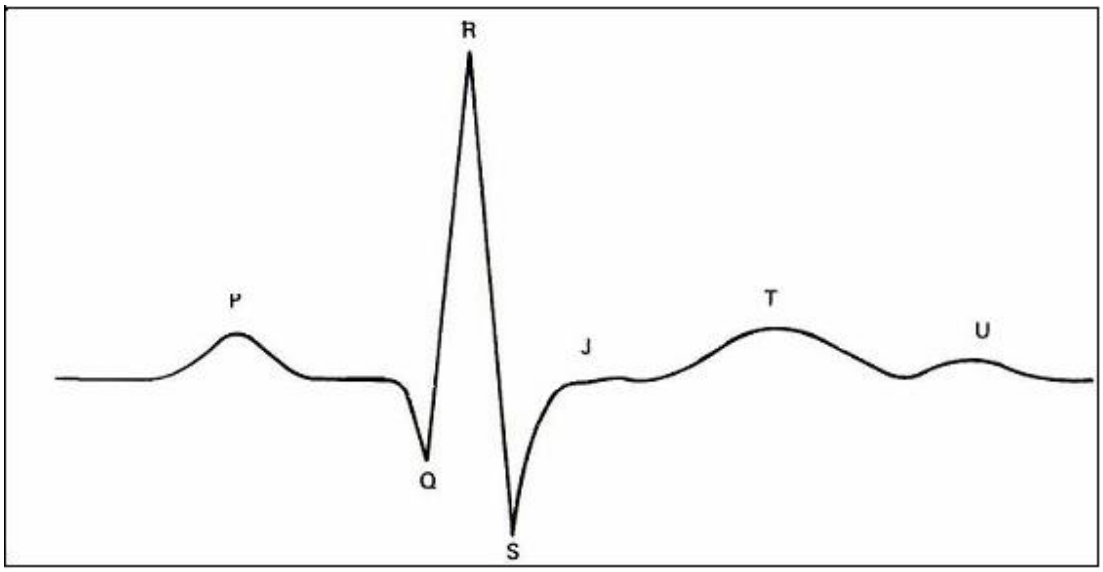

Figure 2.1.b The Electrocardiography wave form (Cromwell, Weibell, Pfeiffer. 1980)

The P wave represents depolarization of the atrial musculature. The QRS complex is combined result of the repolarization of atria and the depolarization of the ventricles which occur almost simultaneously. The T wave is the wave of ventricular repolatization, whereas the U wave, if present, is generally the result of after-potentials in the ventricular muscle.

The voltages changes due to the place of the electrodes that applied to the part of the body. On the heart the QRS complex reaches over 3 mV. Both on the two arms and on the one leg, the QRS reaches 0.2-0.3 mV.

2.2. Blood Pressure (BP)

The blood pressure (BP) is the force applied on a unit area of the blood vessels. In other words, it is defined as the blood flow per unit area. The blood pressure is the result of the cardiac activities of the heart. For this reason, the monitoring the blood pressure gives information about the patient pulse rate. The some cardiac status of the patient is observed with BP.

When the heart pumps the blood into the aorta, the highest pressure called systolic pressure is obtained. When the heart relaxes and is filled with blood, the lowest pressure called as diastolic pressure occurs. Blood pressure is formulized as:

$$BP = \text{Total Resistance} * \text{Cardiac Output} \quad (2.1)$$

$$\text{Cardiac Output} = \text{Heart Rate} * \text{Stroke} \quad (2.2)$$

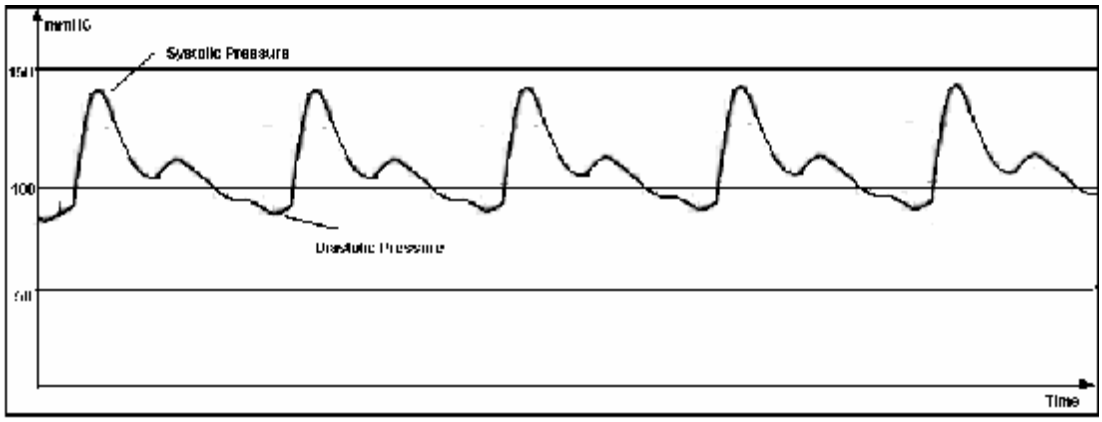

Figure 2.2. Blood Pressure

2.3. Measurement of ECG

In a standard ECG recording, there are five electrodes that connected to the patient: right arm (RA), left arm (LA), left leg (LL), right leg (RL), and chest (C). These electrodes can be connected to the input of a differential buffer amplifier through a lead selector switch.

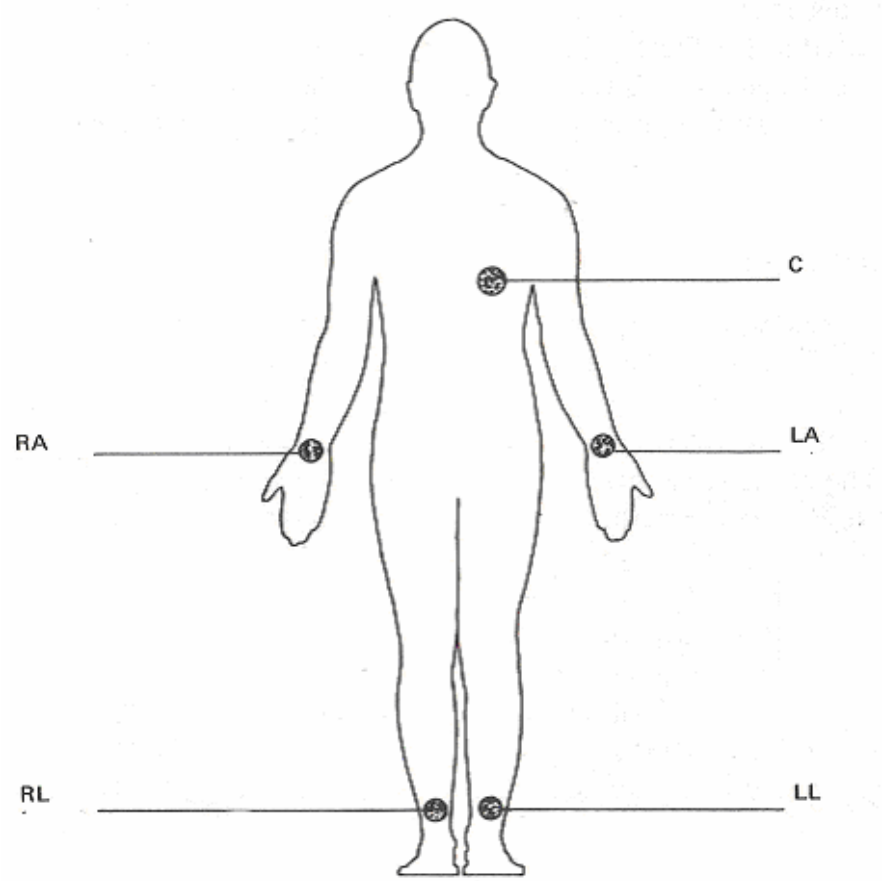

Figure 2.3.a. Normal Connection of Electrodes (Cromwell L., Weibell F., Pfeiffer E.A., 1980)

All of the leads get different signals. Due to the connection of the leads, the result signal is formed. Some of the connections types are showed in Figure 2.4.b.

The Einthoven triangle is the practical measurement concept that was introduced by Einthoven in 1913. Einthoven postulated that the heart was at the center of the en equilateral triangle, the apices of which were the right and left

shoulders and the point where the both legs joined trunk (Geddes L.A., 1995). In early studies, Einthoven used right and left arms and both feet in saline filled bucket as the three electrodes. Thus, he adopted three standard leads: right and left arms and left foot.

The bipolar limb leads are those designated lead I, lead II, and lead III, and from what is called the Einthoven triangle (Figure 2.3.b).

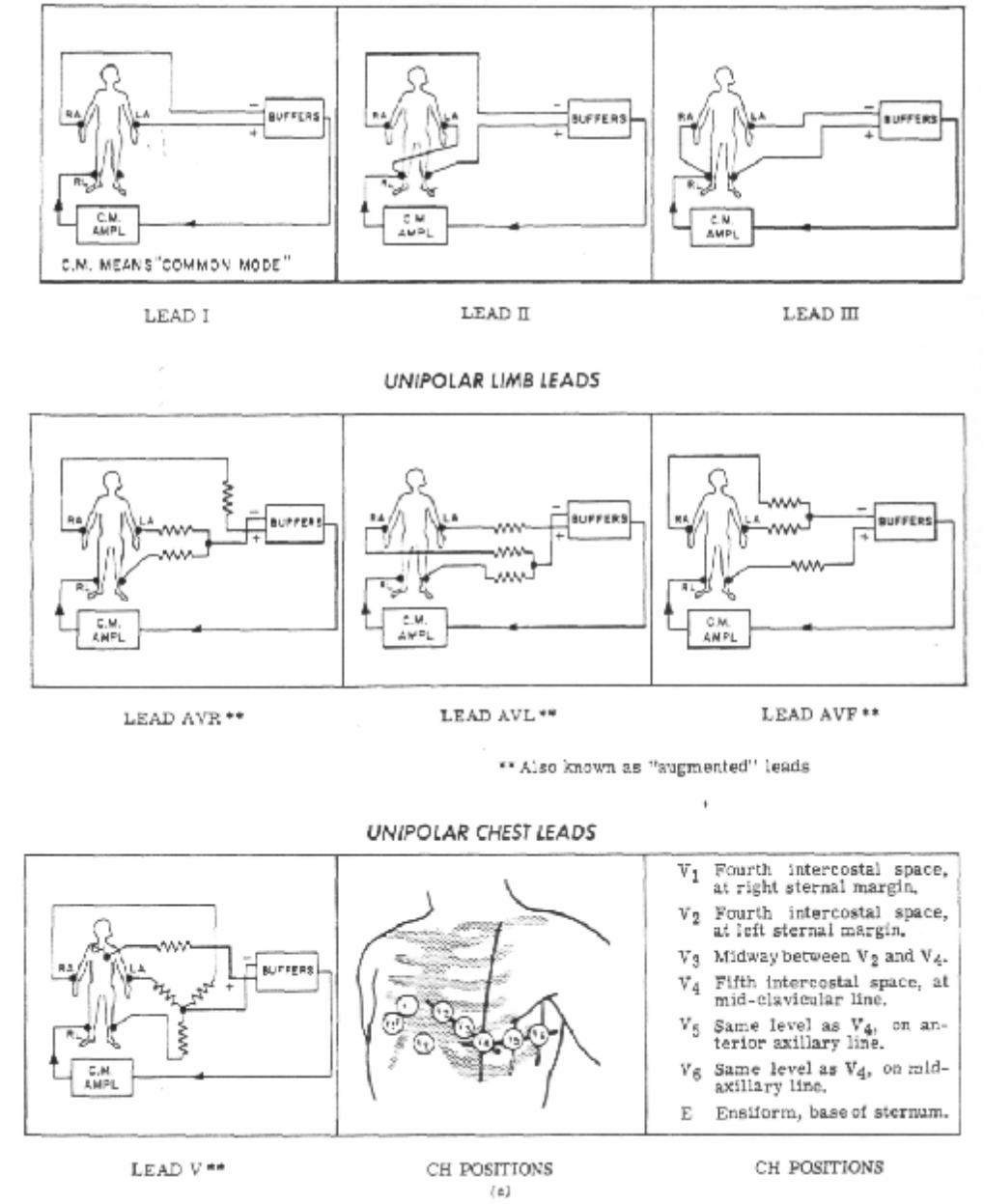

Figure 2.3.b. Standard leads and Einthoven triangle a) Limb and chest leads b) Einthoven triangle

Lead I: LA is connected to the amplifier's noninverting input, while RA is connected to the inverting input.

Lead II: The LL electrode is connected to the amplifier's noninverting input, while the RA is connected to the inverting input (LA is shorted to RL).

Lead III: The LL is connected to the noninverting input, while LA is connected to the inverting input (RA is shorted to RL).

The unipolar limb leads, also known as the augment limb leads, examine the composite potential from all three limbs simultaneously. In all three augmented leads, the signals from two limbs are summed in a resistor network, and then applied to the amplifier's inverting input, while the signal from the remaining limb electrode is applied to the noninverting input.

Lead AVR: RA is connected to the noninverting input, while LA and LL are summed at the inverting input.

Lead AVL: LA is connected to the noninverting input, while RA and LL are summed at the inverting input.

Lead AVF: LL is connected to the noninverting input, while RA and LA are summed at the inverting input.

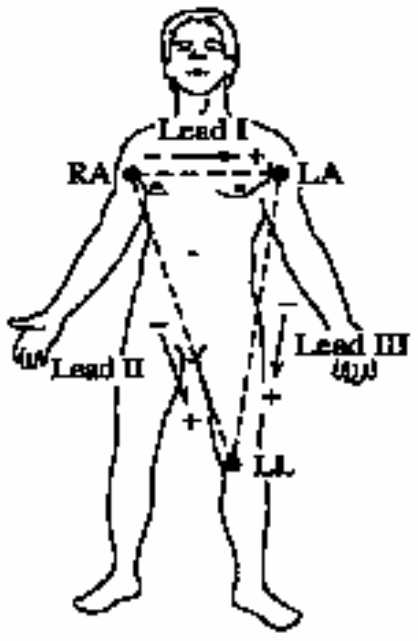

Figure 2.3.c. Einthoven Triangle (Aydın A., 2004)

According to Einthoven triangle law (Figure 2.3.c), if only two bipolar lead potentials are measured and the other bipolar lead can be calculated by using the following equation. On all three lines, the vector sum of the projections is equal to zero.

$$\text{Lead I} + \text{Lead II} + \text{Lead III} = 0 \tag{3.3}$$

2.4. Measurement of BP

At the one of the previous parts of this study, it is declared that the BP is considered a good indicator of the status of the cardiovascular system. The measurement methods of BP are branched as noninvasive and invasive methods. The noninvasive method is explained as the measurement over the skin, no interaction with vessels. But the invasive methods need to reach vessels to observe the flow of the blood.

The methods are used in clinical applications. But in routine clinical tests, the BP is measured by means of an indirect method using *sphygmomanometer.* This is largely used method but it does not continuous recording of pressure variations. This method gives opportunity to get systolic and diastolic pressures, but no other details of the pressure waveform. Furthermore, this method will be failed, if the blood pressure is very low.

All these conditions for measuring by sphygmomanometer are overcame by direct methods or some of the indirect methods with optical transducers. Invasive methods allow more precise and continuous measurement for blood pressure. These methods use gauge sensors, linear variable differential transducers and piezoelectric or capacitance sensors. But such methods are expensive and often applied in intensive care. Non-invasive methods, on the other hand, are easier to implement but less precise.

The pulse oximetry is one of the noninvasive methods that are based on optical permeability of the finger (Flewelling R, 1995). This method is used for the

measurement of oxygen. But this method can be used for measurement of blood volume in vessels of finger tips.

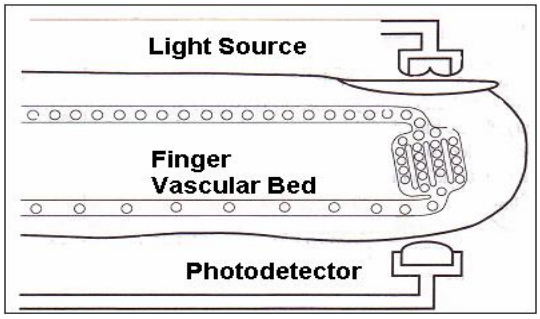

Figure 2.4. Oximetry Method (Flewelling R., 1995)

The passing blood enlarges the volume of elastic vessels or capillary at the tip of fingers. The decreased light force is observed at the photodetector. The transmission of the light is a function of the thickness, light color, structure of the skin, bone, blood, and the other material through which the light passes. This obtained blood volume change has a form of the blood pressure and also gives information as blood pressure measurement.

3. CIRCUITRY OF THE MEASUREMENT SYSTEM

Biosignals that the body produces are low level signals approximately ranges 1 μV to 1 mV. These ranges are very low for the data acquisition systems. The signal requires to be amplified.

The disturbances and noises at the environment also affect the signal. The corrupt signal can cause to get wrong information about patient's health state. This noisy wrong data can cause the doctors make wrong comments. For this reason, the data also must be kept away from noise and such signals. The filtering is the concept that the signal is purified from the noise.

The ECG measurement system has instrumentation amplifier for amplification, high pass filter for neglecting DC levels that was produced by operational amplifiers, low pass filter for neglecting high frequency parts of the signal, and notch filter to neglect the interference produced by electrical devices at the environment and 50 Hz noise.

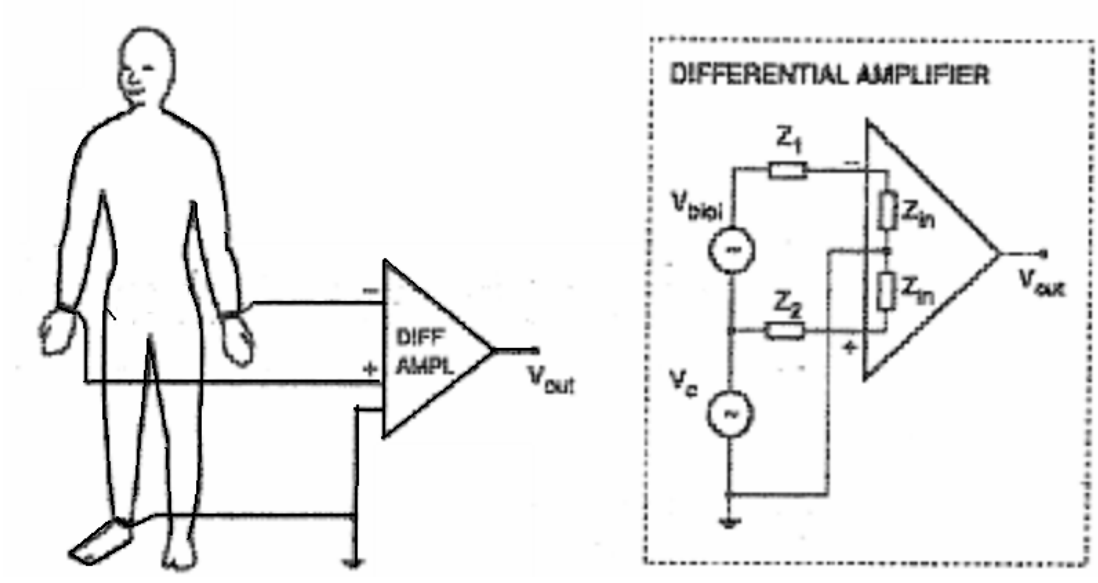

Figure 3.a ECG Measurement (Nagel J.H., 1995)

The blood pressure measurement system has optical to electrical converting part before the amplification stage.

For both of these systems, the amplifier produced by Abdulaziz Aydın is used.

3.1. Circuits of ECG and BP Amplifier

As explained in the previous section, same amplifier is used for ECG which is applied from channel1 and BP which is applied from channel2. The amplifier provides to increase the level of voltage for both signals.

3.1.1. Instrumentation Amplifier

The first stage of both of the circuits are called instrumentation amplifier, which is shown in Figure 3.1.1.a

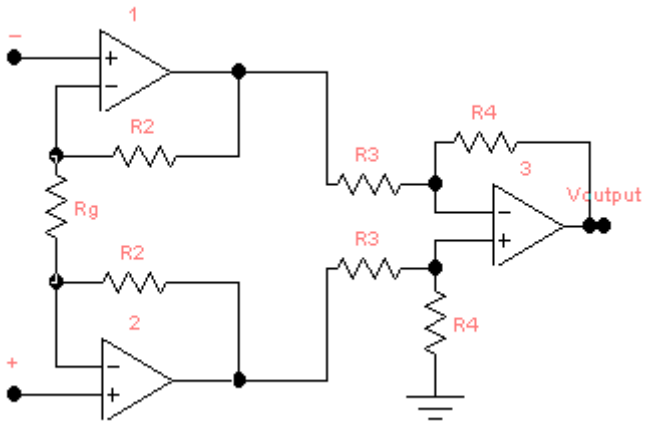

Figure 3.1.1. Instrumentation amplifier

Transfer function of the above stage is;

$$H(s) = \left(1 + \frac{2 * R_2}{R_g}\right)\left(\frac{R_4}{R_3}\right) \qquad (3.1.)$$

Where $R_2 = 100K\Omega$, $R_g = 3.9K\Omega\,\Omega$, $R_4 = 47K\Omega$, $R_3 = 22K\Omega$. All of the resistors have 0.1% tolerance except Rg resistor. To get a good CMRR and cancel interferences, resistors have to be matched perfectly. Total gain of circuit is

52.2 x 2.1=109.

3.1.2. High Pass Filter

In this stage given in Figure 3.1.2., cut off frequency between ECG. At low frequencies, there is important knowledge signal of ECG so cut off frequency was selected 0.495 Hz.

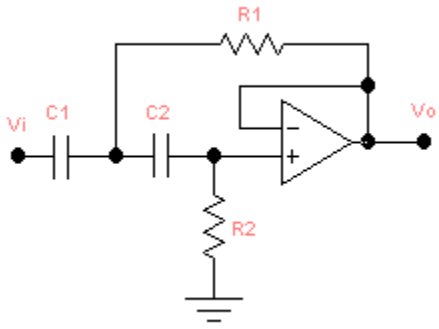

Figure 3.1.2. High pass filter

Transfer function of the high pass filter;

$$H(s) = \frac{s^2 C_1 C_2}{s^2 C_1 C_2 + s\left[(C_1 + C_2)R_2 + (1-K)C_2 R_1\right] + R_1 R_2} \qquad (3.2.)$$

with the cutoff frequency and Q (quality factor);

$$W_O^2 = \frac{1}{R_1 R_2 C_1 C_2} \qquad (3.3.)$$

$$Q = \frac{\sqrt{R_1 R_2}}{R_1 + R_2} \tag{3.4.}$$

The value of components for the ECG circuit is $R_1 = 220K\Omega$, $R_2 = 470K\Omega$, $C_1 = C_2 = 1mF$.

3.1.3. Low Pass Filter

Low pass filters were designed with cutoff frequency at 100 Hz for ECG.

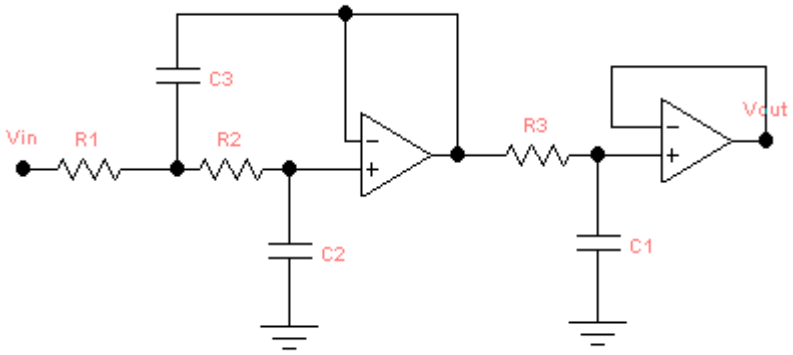

Figure 3.1.3. 3rd low pass filter with 100 Hz cutoff frequency

Transfer function of the filter is;

$$H_1(s) = \frac{1/(R_1 R_2 C_3 C_2)}{s^2 + \left[\frac{1}{R_1 C_3} + \frac{1}{R_2 C_3} + \frac{(1-K)}{R_2 C_2}\right] s + \frac{1}{R_1 R_2 C_3 C_2}} \tag{3.5.}$$

$$H_2(s) = \frac{1}{sC_1R_3 + 1} \tag{3.6.}$$

Thus, total transfer function is $H(s) = H_1(s) * H_2(s)$.

For this filter, the values of the components were selected as $R_1 = R_2 = R_3 = R = 10K\Omega$, and $C_3 = 0.33mF$, $C_2 = 0.82hF$ and $C_1 = 0.15mF$. Cutoff frequency is equal to fc at each stage. That is, for the first stage is

$$W_o = \frac{1}{R\sqrt{C_3C_2}} \tag{3.7.}$$

for the second stage is

$$W_o = \frac{1}{R_3C_1} \tag{3.8.}$$

3.1.4. Patient Protection

The most important problem at biomedical devices is protection of patient from macro shock or leak currents. If electrical equipments are not insulated very well or there is rifled a cable environment of patient, the patient can touch them. Basics of ECG medical devices procedure that reduces or eliminates the resistance of the skin increases possible current flows and makes the patient more vulnerable to macro shock. For example, biopotential electrode paste or jell reduces skin resistance. Thus, in case of touching of patient, leak current can flow through body of patient to ground lead. If the current is adequately large, width and dangerous wounds can expose on body. Both of the ECG circuit is designed with a protection

circuit to prevent this type of problem, as shown in the following schema in Figure 3.1.4.

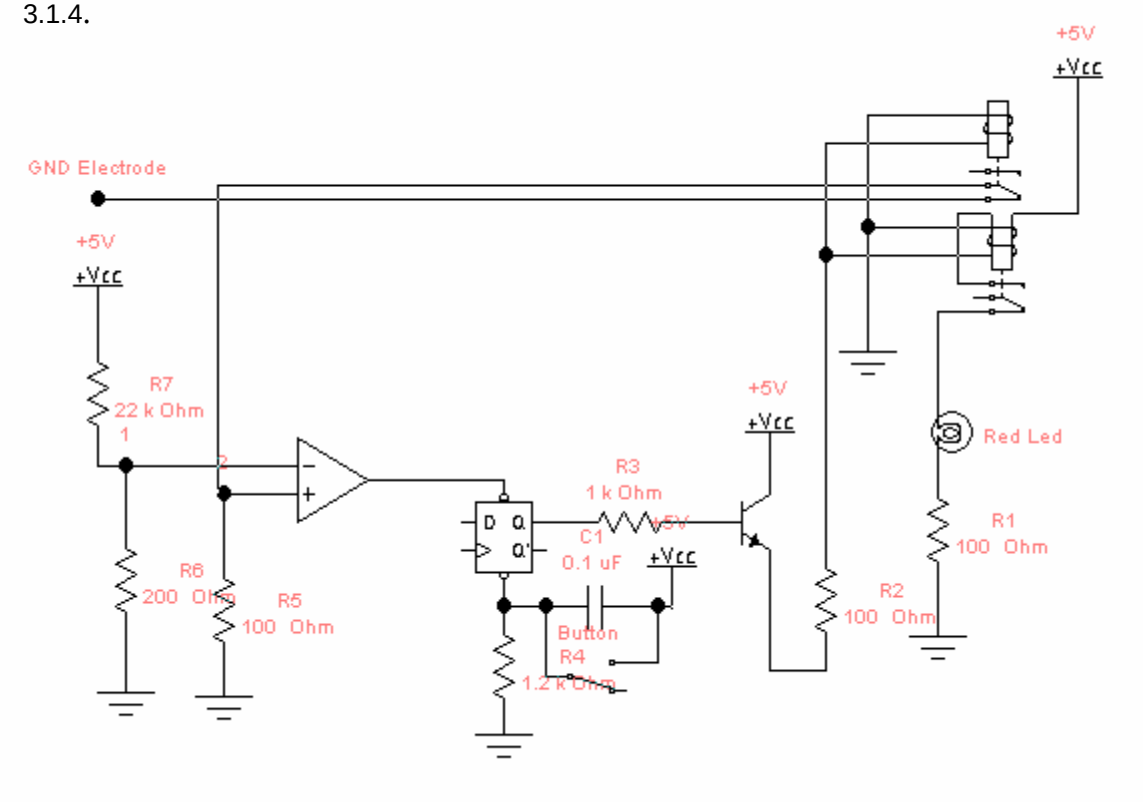

Figure 3.1.4 Patient protection circuit

The protection circuit behaves like a current limiter. When current on the ground electrode reaches to 0.45mA, the circuit cuts connection of ground that come from patient immediately. If any leak current does not flow up to falling below 0.45mA a red led will turn on. In this case, for measurement, it must be pushed to a button. Even if there is still a leak current on the ground electrode, the circuit will disconnect the connection between ground electrode and patient so that the circuit does not permit flowing of leak current. For this purpose, D flip-flop, LM358, transistor and role were used.

In the circuit, ground electrode come from patient was connected to ground with a role and 100Ω resistor. A reference voltage at point 1 that is input of op-amp was selected as;

$$Vref = \left(\frac{V_{cc}}{R_7 + R_6} * R_6 \right), \text{ where Vref=0.45mV} \qquad (3.9.)$$

D flip-flop is state of reset at the beginning and output (Q) is equal to 0V. if current on the ground electrode is lower then 0.45mA, out put of op-amp is 0V. As seen in the circuit schema, output of op-amp was connected to input of SET of D flip-flop. In this case output of the flip-flop is Q (logic-0) and state of transistor is OFF so that role can not change state. However, if current on the ground electrode is higher than 0.45mA, firstly output of op-amp will be +Vcc then output of flip-flop will be +Vcc (logic-1). After when state of transistor is ON, role will change state and red led turn on. Thus, the connection between patient and ground will be disconnected. In this case, even if it was pressed to the button, the flip-flop, transistor, and role does not change state, according to the truth table of CD4013BE. Disconnection will go on to protect patient. If current drops under 0.45mA, in this case out put of opamp will be 0V and connection can be provided by pressing to the button again. Because input of SET is equal 0V and input of RESET will be logic-1 by pressing to the button. The output of flip-flop will equal to logic-0. If output of flip-flop is logik-0, state of transistor will be OFF. Circuit components are LM358, BJT 237 transistor and CD4013BE dual D flip-flop.

3.2. Optical Transducer Circuit of BP Measurement System

This part of the system is constructed with two resistors to adjust the voltage and current on LDR and LED. The finger whom blood volume is to be detected is put between LED and the LDR. The led emits high wavelength red light. The light passing trough the finger is absorbed by LDR according to blood volume in finger. The light intensity on the LDR causes the resistance changes and so voltage changes. This voltage change will affect the output of the system.

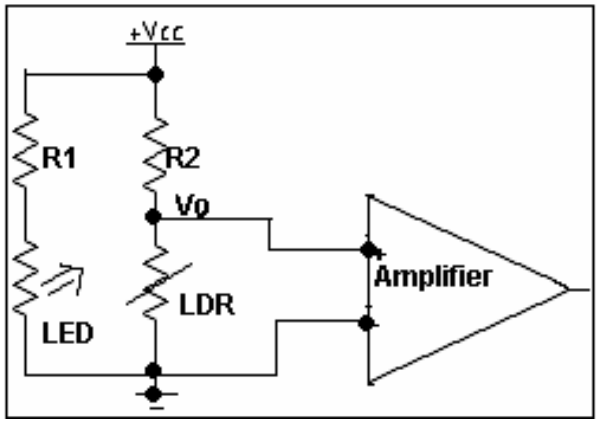

Figure 3.2. Optical Blood Pressure Measurement Card Circuit

The led has wavelength between 600-770 nm (red light) wavelengths. This red light passes easier than the other colors. Infrared LED could be used to provide high light intensity at the LDR surface. But as a result at voltage level, there would be no much difference between red led or infrared led.

4. DATA ACQUISITION OF THE SYSTEM

4.1. Data Acquisition

Data acquisition involves gathering signals from measurement sources and digitizing the signal for storage, analysis, and presentation on a personal computer (PC). Data acquisition (DAQ) systems come in many different PC technology forms for great flexibility when choosing the studied equipment. Data acquisition is one of the essential parts for measurement systems.

This DAQ system contains PCL 818 Advantech Data Acquisition Card. The system is illustrated in figure 4.1.a.

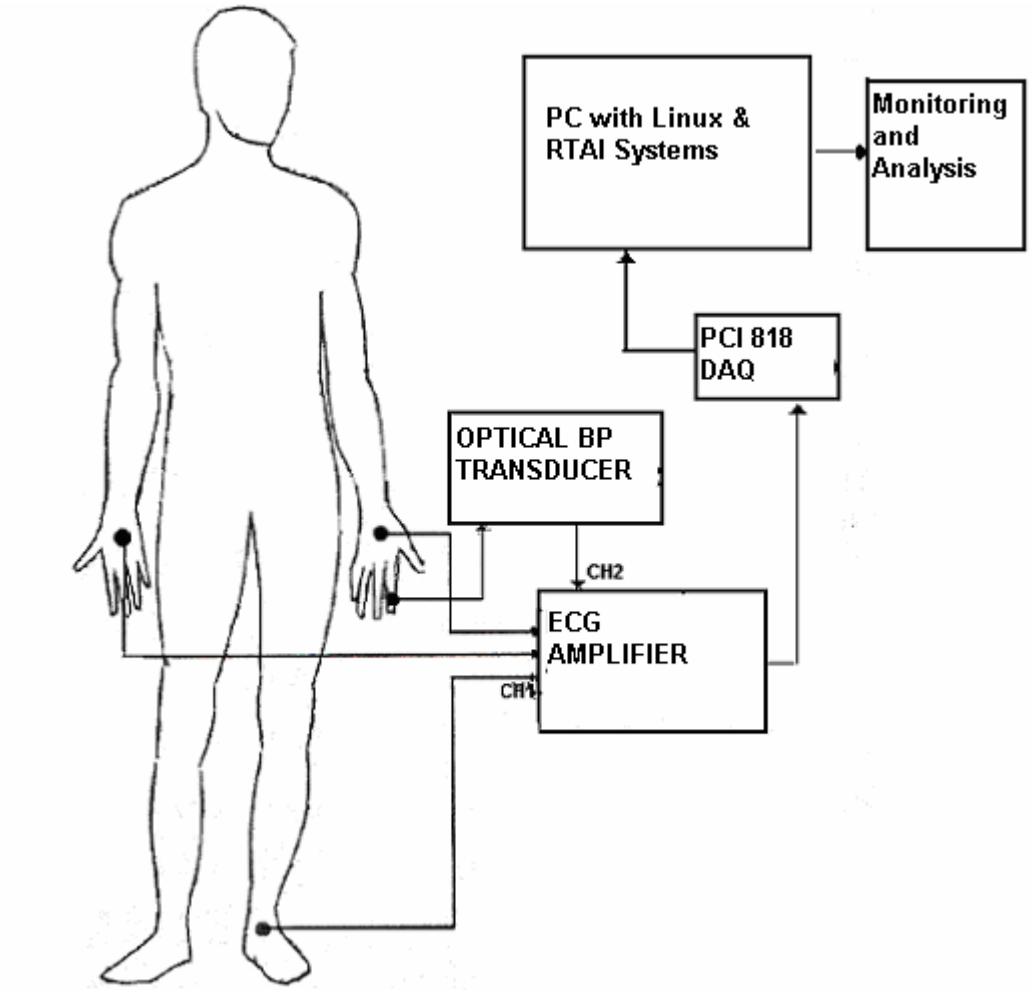

Figure 4.1.a Block Diagram of the System

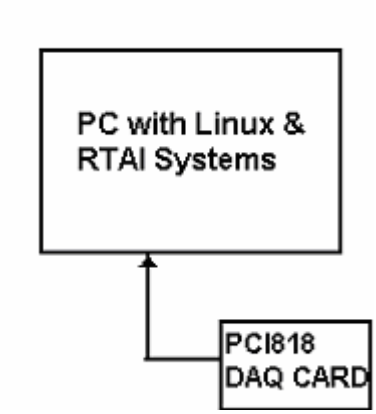

Figure 4.1.b Block Diagram of Data Acquisition Part

The PCL-818 is a high performance multi-function data acquisition card for computers. It offers the five most desired measurement and control functions: 12-bit A/D conversion, D/A conversion, digital input, digital output and timer/counter (Advantech, 1994). Automatic channel scanning circuitry and on-board SRAM let you perform multiple-channel A/D conversion with DMA and individual channel gains.

The specifications of the card which are supplied by Advantech Cooperation are explained in Appendix A.

4.1.1. Settings of the PCI 818 Data Acquisition Card

The card has one function switch and seven jumper settings. These settings must be done carefully due to the needs. These settings provide your signal to be acquired correctly due to your frequency choice, gain tune, clock choice. The sampling rate will be changed.

4.1.1.1. Base Address

We control the card operations by reading or writing data to PC's I/O port address. S1 switch is used to set up the base address. Valid base address must be

detected from Hex 000 to Hex 3F0. Some of these addresses may be used by some other devices. The switch set is shown in figure.

Range (hex)	Switch position					
	1	2	3	4	5	6
000 - 00F	●	●	●	●	●	●
010 - 01F	●	●	●	●	●	○
200 - 20F	○	●	●	●	●	●
210 - 21F	○	●	●	●	●	○
* 300 - 30F	○	○	●	●	●	●
3F0 - 3FF	○	○	○	○	○	○

○ = Off ● = On * = default

Figure 4.1.1.1. Base Address Selection Map (Advantech, 1994)

We adjusted the base address to H0210 by setting the switches SW1 and SW6 are OFF. Via this selection, the status address is H0216 and the control address is H0219.

4.1.1.2. Timer Clock Selection

The jumper 2 (JP2) controls the input clock frequency for the timer. Two choices are support by the card. These are 10MHz and 1MHz. The pacer rate is calculated as

$$\text{Pacer Rate} = \text{Fclk} / (\text{Div1}*\text{Div2}) \quad (4.2)$$

where Fclk is clock frequency at 1MHz or 10 MHz, Div1 and Div2 are the dividers set in counter1 and counter 2. Our selection is 1 MHz.

4.1.1.3. Input Voltage Range

The jumper 7 selects the input voltage range for A/D converter. When we set JP7 to ±5, the maximum input voltage range is ±5 V. When we set JP7 to ±5, the maximum input voltage range is ±10 V.

4.1.1.4. Channel Configuration

The PCL-818L offers 16 single-ended or eight differential analog input channels. Jumper JP6 switches the channels between single ended or differential input, as shown below:

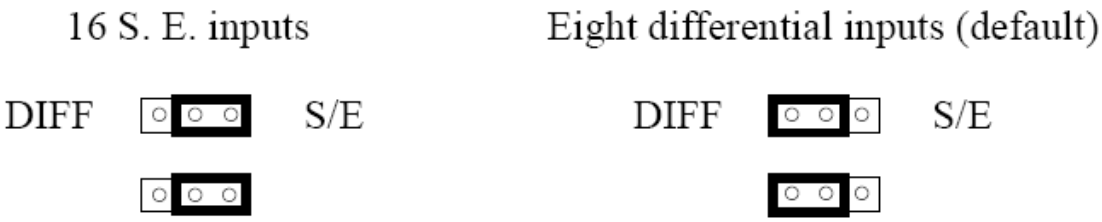

Figure 4.1.1.4. Channel Selection Diagram (Advantech, 1994)

We set the system channels to single ended form.

4.1.1.5. D/A Reference Voltage

Jumper JP4 selects reference voltage source for the PCL-818L's D/A converters. You can use the card's internal reference or supply an external reference.

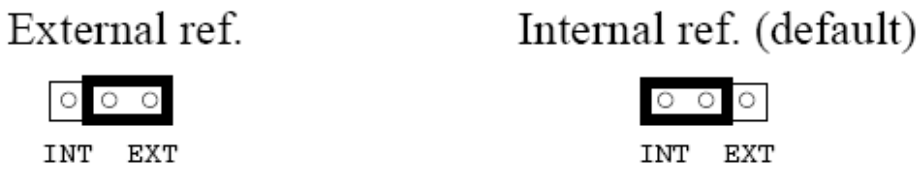

Figure 4.1.1.5. Voltage Selection Diagram (Advantech, 1994)

Setting the JP4 to INT, the D/A converter takes its reference voltage input from the card's on-board reference. Jumper JP5 sets the on-board reference to either -5 V or -10 V. With JP4 set to INT, the D/A channel has an output range of 0 V to +5 V or 0 V to +10 V. When you set JP4 to EXT, the D/A converter takes its reference voltage input from pin 31 of connector CN3. We apply any voltage between -10 V and +10 V to this pin to function as the external reference. The reference input can be either DC or AC (<100 KHz).

Using an external reference with voltage Vref provides to program the D/A channel to output from 0 V to -Vref. The D/A converter act as a programmable attenuator. The attenuation factor between reference input and analog output is:

$$\text{Attenuation factor} = G / 4095 \qquad (4.1.1.5)$$

Where G is a value you write to the D/A registers between 0 and 4095. For example, if you set G to 2048, then the attenuation factor is 0.5. A sine wave of 10 V amplitude applied to the reference input will generate a sine wave of 5V amplitude on the analog output.

4.1.1.6. Internal Voltage Reference

Using an internal reference voltage (set with JP4) forces the PCL-818L to select a choice of DC internal reference voltage sources: -5 V and - 10 V. JP5 selects the source, as shown below:

Figure 4.1.1.6. Internal Voltage Reference Selection Diagram (Advantech, 1994)

5. REAL TIME APPLICATION INTERFACE

5.1. Real Time Systems

The word meaning of “Real Time” is occurring immediately. We can describe real-time system that responds the inputs immediately.

A real-time system means that it performs its functions and responds to external, asynchronous events within a specified amount of time. So a real-time operating system can be defined as a system capable of guaranteeing timing requirements of the processes under its control (Stankovic J.A., Ramamritham K., 1993). In another words, a real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period.

The systems can face some difficulties and time delays while making computations, multiprocessing, task changes. This causes the needed operation to be delayed. For this reason, real time systems are used. The real time systems can take priority at CPU due to the scheduled tasks.

A real-time operating system (RTOS) schedules the tasks to be performed according to a set of established priorities. Under "normal" conditions, tasks follow a predictable schedule of execution. The ability to respond to environmental inputs in a priority-based manner allows a real-time operating system to respond almost instantaneously to events as they occur. This makes it the ideal control system for mission-critical applications - such as medical monitoring devices, flight consoles, automated assembly lines, telecom hubs, or off-planet vehicles.

Real-time systems must perform computations according to deadlines. By definition, if a hard real-time system misses a deadline, something catastrophic happens. The system fails. The computed results are useless. In the worst-case scenario, lives are lost.

The system response must be *fast and predictable*. *Fast* means that it has a low latency, i.e. it responds to external, asynchronous events in a short time. The lower the latency, the better the system will respond to events which require

immediate attention. *Predictable* means that it is able to determine task's completion time with certainty.

Typically a real time system represents the *computer controlling system* that manages and coordinates the activities of a controlled system, that can be viewed as the environment with which the computer interacts. The interaction is bidirectional, says through various sensors (environment -> computer) and actuators (computer -> environment), and is characterized by *timing correctness constraints*.

It is desirable that *time-critical* and *non time-critical activities* coexist in a real time system. Both are called tasks and a task with a timeliness requirement is called a *real time task*. Typically real time tasks have the following types of requirements and/or constraints.

Timing constraints: The most common types are either *periodic* or *aperiodic*. An aperiodic task has a deadline by which it must finish or start, or it may have a constraint on both start and finish times. A periodic task has to be repeated once per period. Most sensory processing is periodic, while aperiodic requirements can arise from dynamic events.

Resource requirements: A real time task may require access to certain resources such as I/O devices, data structures, files and databases.

Communication requirements: Tasks should be allowed to communicate with messages.

Concurrency constraints: Tasks should be allowed concurrent access to common resources providing the consistency of the resource is not violated.

Criticalness: Depending on the functionality of a task, meeting the deadline of one task may be considered more critical than another. For example, a task that reacts to an emergency situation, such as fire on the factory floor, probably will be more critical than the task that controls the movements of a robot under normal operating conditions.

Precedence relationships: A complex task (for example, one requiring access to many resources) is better handled by breaking it up into multiple subtasks related by precedence constraints and each requiring a subset of the resources.

5.1.1. Hard Real Time Systems

A hard real-time system is a system that requires a guaranteed response to specific events within a defined time period. The failure of a hard real-time system to meet these requirements typically results in a severe failure of the system. It is absolutely imperative that responses occur within the required deadline.

5.1.2. Soft Real Time Systems

Soft real time is a property of the timeliness of a computation where the value diminishes according to its tardiness. A soft real time system can tolerate some late answers to soft real time computations, as long as the value hasn't diminished to zero. A soft real time system will often carry meta requirements such as a stochastic model of acceptable frequency of late computations.

Soft real time is often improperly applied to operating systems that don't satisfy the necessary conditions for guaranteeing that computations can be completed on time. In this type of real time, deadlines are important but which will still function correctly if deadlines are occasionally missed.

5.2. Real Time Application Interface (RTAI)

Real-Time Application Interface (RTAI) is a real-time Linux implementation based on RT Linux. RTAI is not a real time operating system. It adds a small real-time kernel below the standard Linux kernel and treats the Linux kernel as a low priority real-time task. RTAI provides a large selection of inter-process communication mechanisms and other real-time services.

RTAI treats the conventional Linux kernel as a low-priority real-time task, which may do its normal operations whenever there are no higher priority real-time tasks running. In the basic RTAI operation, the real-time tasks are implemented as Linux kernel modules, similarly to RTLinux. RTAI handles the interrupts from peripherals and dispatches the interrupts to Linux kernel after handling the possible real-time actions triggered by the interrupts.

Figure 5.2. shows the basic architecture of RTAI, which is rather similar to RTLinux architecture. There are interrupts originating from processor and peripherals, of which processor originated interrupts (mainly error signals such as division error) are still handled by the standard Linux kernel but the interrupts from the peripherals (e.g. timer) are handled by RTAI's *Interrupt dispatcher*. The RTAI forwards the interrupts to the standard Linux kernel handlers when there are no active real-time tasks. The interrupt disabling and enabling instructions in Linux kernel are replaced by macros that forward the instructions to RTAI. When interrupts are disabled in the standard kernel, RTAI queues the interrupts to be delivered after the Linux kernel has enabled the interrupts again.

When an interrupt occurs, the real time kernel intercepts the interrupt and decides what to dispatch. If there is a real time handler for the interrupt, the appropriate handler is invoked. If there is no real time interrupt handler, or if the handler indicates that it wants to share the interrupt with Linux, then the interrupt is marked as pending. If Linux has requested that interrupts be enabled, any pending interrupts are enabled, and the appropriate Linux interrupt handler invoked - with hardware interrupts re-enabled.

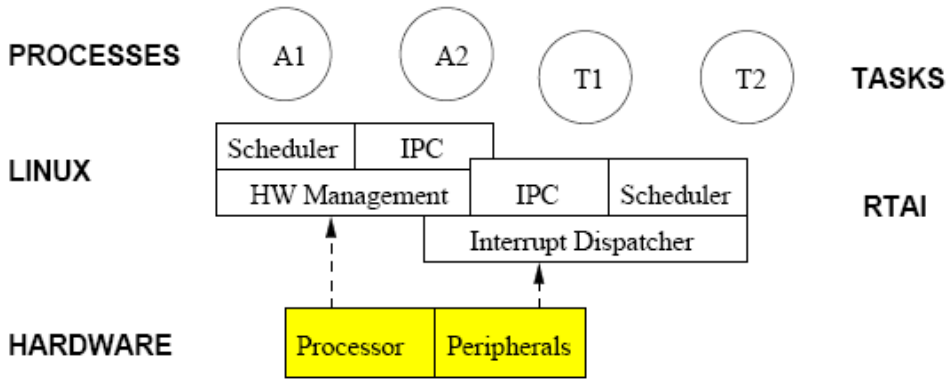

Figure 5.2. Architecture of RTAI (Sarolahti P.,2001)

For RTAI all interrupts are initially handled by the real time kernel and are passed to Linux only when there are no active real time tasks. Changes to the Linux kernel are minimized by providing the kernel with a software emulation of the interrupt control hardware. Thus, when Linux has disabled interrupts, the emulation software will queue interrupts that have been passed on by the real time kernel.

5.2.1. Hardware Abstraction Layer

RTAI developers introduce the concept of *Real-Time Hardware Abstraction Layer (RTHAL)* which is used for intercepting the hardware interrupts and processing them. RTHAL is a structure installed in the Linux kernel which gathers the pointers to the internal hardware related kernel data and functions needed by the RTAI to operate. The purpose of RTHAL is to minimize the number of changes needed to make to the kernel code and thereby improve the maintainability of RTAI and Linux kernel code. With RTHAL, the different operations (e.g. the interrupt handlers) are easy to be changed or modified without having to interfere with the Linux implementation. For example, the RTHAL structure contains the interrupt handler table, which lists the functions that are called for handling different interrupts.

When RTHAL is installed on Linux, the RTSAI function calls, and data structures related to hardware interaction are replaced by pointers to the RTHAL structure. Initially, the structure contains the pointers to the original functions and

data of the Linux implementation, but when RTAI is enabled, the needed replacements are made to the pointers in the RTHAL table.

Briefly, RTHAL intercepts all hardware interrupts and routes them to either standard Linux or to real-time tasks depending on the requirements of the RTAI schedulers.

Advantages and Disadvantages of the RTHAL

The main advantages of using a RTHAL approach compared to a relatively kernel intrusive:

- The changes needed to the standard Linux kernel are minimal, a few lines in eleven source files plus configuration additions to three files in the build structure, (Makefile, configuration files etc). This lower intrusion on the standard Linux kernel improves the code maintainability, and makes it easier to keep the real time modifications up-to-date with the latest release of the Linux kernel (DIAPM, Lineo Inc., 2002).
- The real time extensions can easily be removed by replacing the interrupt function pointers with the original Linux routines. This is especially useful in certain debugging situations when it is necessary to remove the extensions, and when verifying the performance of standard Linux with and without the real time extensions (DIAPM, Lineo Inc., 2002).

The Linux kernel suffers a slight, but essentially negligible, performance loss when RTAI is added due to the indirection through pointers to the interrupt mask, unmask and flag functions.

In consideration of both strengths and weaknesses, this technique has shown itself to be both efficient and flexible, because it removes none of the capability of standard Linux, yet it provides guaranteed scheduling and response time for critical tasks.

5.2.2. Schedules

The scheduling units of RTAI are called *tasks*. There is always at least one task, namely the Linux kernel which is run as a low-priority task. When real time tasks are added, the scheduler gives them priority over the Linux kernel. The scheduler provides services such as *suspend, resume, yield, make periodic, wait until*, which are used in various real-time operating systems.

The scheduler is implemented as a dedicated kernel module (again similarly to RTLinux), which makes it easy to implement alternative schedulers if necessary. There are three different types of schedulers depending on the machine type. *Uniprocessor (UP)* scheduler is intended to be used on uniprocessor platforms, and it can not be used with multiprocessor machines. *Symmetric Multiprocessor (SMP)* scheduler is designed for SMP machines and it provides an interface for the applications to select the processor or the pool of processors on which a given task is run. If the user does not specify any processor for the task, SMP selects the processor based on the processor load status. *Multi-uniprocessor (MUP)* scheduler can be used with both multi and uniprocessor machines. Unlike with the SMP scheduler, the tasks must be bound to specific processor when MUP scheduler is used. On positive side, MUP scheduler allows more flexible timer mechanisms for the tasks than SMP or UP scheduler

5.2.3. Difference of RTOS Due to the Conventional Operating System

A conventional OS, such as Linux, attempts to use a “fairness” policy when scheduling threads and processes to the CPU (Furr S., 2002). This gives all applications in the system a chance to make progress, but does not establish the supremacy of realtime threads in the system or preserve their relative priorities, as is required to guarantee that they finish on time. Likewise, all priority information is usually lost when a system service, usually performed in a kernel call, is executing on behalf of the client thread. These results in unpredictable delays and, thus prevent an activity from completing on time. By contrast, the micro kernel architecture used

in the RTOS is designed to deal directly with all of these requirements (Furr S., 2002). The microkernel itself simply manages processes - and threads - within the system, and allows them to communicate with each other. Scheduling is always performed at the thread level, and threads are always scheduled according to their fixed priority - or, in the case of priority inversion, by the priority as adjusted by the microkernel to compensate for priority inversions.

Consequently, a high-priority thread that becomes ready to run can preempt a lower-priority thread. Within this framework all device drivers and operating system services apart from basic scheduling and interprocess communication (IPC) exist as separate processes within the system. All services are accessed through a synchronous message-passing IPC mechanism that allows the receiver to inherit the priority of the client. This priority-inheritance scheme allows Operating System Requirements (OSR) 5 to be met by carrying the priority of the original real-time activity into all service requests and subsequent device-driver requests. Using this model, an operating service or device driver can be swapped out in favor of a realtime version that satisfies these requirements.

5.3. RTAI Installation

Rtai offers the same services of Linux kernel core, adding the features of an industrial real time operating system. The Hal provides few dependencies to Linux Kernel.

Rtai works with set of kernel versions which are provided trough internet. In the following subjects in this chapter are about installation of RTAI step by step.

Our system works with the kernel version of linux-2.6.7 and Rtai-3.1 on Fedora 2 which is one of the Linux operating system versions.

5.3.1. Downloading Kernel

The kernel source is downloaded from the web site address given below.

ftp://ftp.kernel.org/pub/linux/kernel/v.2.6/linux-2.6.7.tar.gz

The downloaded source is our source but some versions of this source may need a patch which is got from the web site address below.

ftp://ftp.kernel.org/pub/linux/kernel/patches

We have to download the patch version 2.6.7 as patch-2.6.7.bz2

5.3.2. Downloading RTAI Source

In the same manner, the rtai source is downloaded from web site address given below.

http://www.aero.polimi.it/RTAI/rtai-3.1.tar.bz2

5.3.3. Extracting the Sources

Both of the kernel and the rtai sources are extracted into the /usr/src/ folder.

5.3.4. Symbolic Links

The Linux OS provides symbolic links to specified folders, to reach the folders easily. It is like a short cut.

ln –s /usr/src/linux-2.6.7 linux

ln –s /usr/src/rtai-3.1 rtai

This command produces virtual links to the source folders. It is not necessary, but it is useful while compiling kernel and compiling rtai.

5.3.5. Patching the Kernel with RTAI

The kernel must be patched with the appropriate rtai patch. This provides the kernel to be introduced to rtai. This introduction process loads the kernel with rtai specifications. The c source files in kernel are replaced with rtai c source files.

cd/usr/src/linux

patch –p1</usr/src/rtai/rtai-core/arch/i386/patches/hal6-2.6.7.patch

Some patches related with rtai may be found in same folder. That patch can not be compatible. It notices you while you are patching. It says that the versions are uncompatible.

5.3.6. Compiling the Kernel

The most important part of the compiling kernel is to create .config file. Every Linux program is an executable file holding the list of opcodes the CPU executes to accomplish specific operations. For instance, the ls command is provided by the file /bin/ls, which holds the list of machine instructions needed to display the list of files in the current directory onto the screen. The behaviors of almost every program can be customized to your preferences or needs by modifying its configuration files. Shortly, the configuration file is a file created by an application program that stores the choices you make when you install (or configure) the program so that they're available the next time you start the program.

You must copy the configuration file under boot folder to linux source folder. The default linux version was Linux-2.6.9. We decided to change it to Linux 2.6.7 to work with the rtai-3.1 source to provide stable work. The copying process is done with the command below.

```
cp /boot/config-2.6.9-1.667 /usr/src/linux/
```

Then the configuration selection can be done running the command below. This command provides selection menu for configuration. Some important changes or selections must be done as following.

- Adeos is selected (Adeos Support)
- “Loadable Kernel Module Support- Module Versioning Support” is disabled.
- “Kernel Hacking-Compile the kernel with frame pointers” is disabled.

Adeos is a resource virtualization layer available as a Linux kernel patch. Adeos enables multiple entities called *domains* to exist simultaneously on the same machine. These domains do not necessarily see each other, but all of them see Adeos. A domain is most probably a complete OS, but there is no assumption being made

regarding the sophistication of what is in a domain. However, all domains are likely to compete for processing external events (e.g. interrupts) or internal ones (e.g. traps, exceptions), according to the system wide priority they have been given. Adeos support enables rtai modules to run.

Kernel Module Support provides to insert a module to kernel or remove a module from kernel.

The following command provides to run configuration program.

```
make menuconfig
```

After choosing and then closing the configuration menu, running the "bzImage" command identifies the dependencies and the "make" command forms the objects of kernel which will be loaded. For identifying the dependencies of the installation "make dep" command is ran before "make bzImage".

```
make dep

make bzImage

make

make modules_install

make install
```

These command sets provides you to install linux-2.6.7 kernel on your pc.

You can use following instruction format while compiling the kernel. The "&&" characters are used to run commands with one command line.

```
make bzImage && make && make modules_install && make install
```

5.3.7. Compiling the RTAI

Compiling the RTAI is approximately same as compiling RTAI. Firstly, change the folder to rtai source code.

```
cd /usr/src/rtai
```

The configuration is made by the command “make menuconfig”. Some options are chosen due to your needs. For example, if you want to write a rtai application by using parallel ports, you must enable the configuration of the ports with “Y” choice. In this step, the important issue is selecting the path to linux source tree. Write the path to linux source tree as /usr/src/linux.

Then you select all your choices or you can select defaults. After selecting the configuration options, close and save the configuration file. The “make” command is run.

```
make
```

This command forms the object files for installation. “make install” command will installs the rtai.

These instruction sets modify the kernel. The grub.conf which is boot loader file is automatically modified.

After rebooting your system the RTAI-Linux is ready to write applications. At the beginning, you have to test your RTAI using test modules or written applications.

5.3.8. Testing Your RTAI

If all the parts of installation has done properly, the test operations could be done.

For testing, you must run the test program which is about latency test under realtime/testsuite directory.

```
cd /usr/realtime/testsuite/kern/latency

./run
```

You can see the output of latency test as maximum, minimum and average values.

5.3.9. RTAI Modules

RTAI is very much module oriented. Linux allows dynamically loading or unloading components of the operating system. The way to manually insert a module into the kernel is using the “insmod” command. Modules are removed by “rmmod” command. A kernel module has to have at least two functions: “init_module()”, which is called when module is inserted into kernel, and “cleanup_module()”, which is called just before it is removed.

To use RTAI, you load the modules that implement whatever RTAI capabilities you need. There are three core modules: rtai module (rtai.o), scheduler module (rtl_sched.o) and the module that implements rt-fifos (rt_fifo.o). The application process is written as a kernel module, and this kernel module is compiled. The result of compilation is load with the kernel core modules.

```
insmod rtai.o
insmod rtl_fifo.o
insmod rt_sched.o
```

insmod /home/deneme/surucu.o

The last command provides application modules to be ran as a part of system's kernel. The module executes the function in the init_module. To stop the application following commands are executed.

rmmod surucu
rmmod rt_sched
rmmod rtl_fifo
rmmod rtai

5.3.10. Compiling RTAI Modules

To compile any kernel module, gcc command is used:

gcc –c –D__KERNEL__ -DMODULE_ -o ornek ornek.c

This command set produces ornek.o module as an application. For the files more than one, this command set is not very useful. Instead of this command set, using *makefile* for compilation is very advantageous.

The makefile has a vital role to produce driver and executable file. The "make" command is important tool for linux systems. This command looks for Makefile to run. A Makefile is a file that instructs the program how to be compiled and to be linked to a program. A Makefile can include several source files for compilation. A makefile example:

all: ornek.o ornek
RTLINUX:-I /usr/src/linux/include –I/usr/src/realtime/include
CFLAGS: -O2 –Wall –DMODULE –D__KERNEL__
ornek.o:ornek.c
gcc $(RTLINUX) $(CFLAGS) –c ornek.c

5.3.11. Building RTAI Example

RTAI needs executable rtai file, rtai code, related makefile, optinal script which is not necessary to load rtai modules such as rtai.o, rtai_hal.o, and sometimes run.info file.

The existence of these files , “make” command is ran to form object files due to the codes in makefile. This command produces “.ko” extension files that are related with driver.

The first example is sine wave. The code is saved as sine.c

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <asm/io.h>
#include <math.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>


#define TICK_PERIOD 1000000
#define TASK_PRIORITY 1
#define STACK_SIZE 10000
#define FIFO 0

static RT_TASK rt_task;

static void fun(int t)
{
        int counter = 0;
        float sin_value;
        while (1) {
```

```
                sin_value = sin(2*M_PI*1*rt_get_cpu_time_ns()/1E9);
                //sin_value = rt_get_cpu_time_ns()/1E9;
                rtf_put(FIFO, &counter, sizeof(counter));
                rtf_put(FIFO, &sin_value, sizeof(sin_value));
                counter++;
                rt_task_wait_period();
        }
    }

    int init_module(void)
    {
        RTIME tick_period;
        rt_set_periodic_mode();
        rt_task_init(&rt_task, fun, 1, STACK_SIZE, TASK_PRIORITY, 1, 0);
        rtf_create(FIFO, 8000);
        tick_period = start_rt_timer(nano2count(TICK_PERIOD));
        rt_task_make_periodic(&rt_task, rt_get_time() + tick_period,
tick_period);
        return 0;
    }

    void cleanup_module(void)
    {
        stop_rt_timer();
        rtf_destroy(FIFO);
        rt_task_delete(&rt_task);
        return;
    }
    MODULE_LICENSE("GPL");
```

This code provides realtime sine wave pulse due to the priority at kernel space. To see the output of these program you need a user space program called as kullanici.c to communicate with kernel space program.

```
include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>

static int end;
static void endme(int dummy)
{
        end=1;
}

int main(int argc, char * argv[])
{
        int fifo, counter;
        float sin_value;
        if ((fifo = open("/dev/rtf0", O_RDONLY)) < 0) {
                fprintf(stderr, "Hata /dev/rtf0\n");
                exit(1);
        }
        signal(SIGINT, endme);
        while (!end) {
                read(fifo, &counter, sizeof(counter));
                read(fifo, &sin_value, sizeof(sin_value));
                printf(" Sayac : %d Deger : %f \n", counter, sin_value);
```

```
        }
        return 0;
}
```

The makefile has a uniqe name as “Makefile” in all linux systems. For 2.6.x kernel versions, makefile format is like:

```
obj-m  := sine.o
KDIR  := /lib/modules/$(shell uname -r)/build
PWD          := $(shell pwd)
EXTRA_CFLAGS  :=  -I/usr/realtime/include  -I/usr/include/  -ffast-math  -mhard-float
default:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
        gcc -o kullanici kullanici.c
```

A run script is used to load rtai_load module.

```
/usr/realtime/bin/rtai_load
```

.runinfo file regulates the modules, and other executable files while you run your executable file. It coordinates the files, modules which may loaded. .runinfo file contains the lines below.

```
Latency:ksched+fifos:push sine;./kullanici;popall:control_c
```

After having your all files, you run “make” command to compile the codes. After running this command you will get sine.mod.c, sine.mod.o, sine.ko, and scope executable files.

5.3.12. Troubleshooting

When rebooting your system, you can get kernel panic error. This is caused by enabling SELinux. This problem can be solved by modifying the part for the new kernel in /etc/grub.conf

```
title RTAI_Fedora (2.6.7-adeos)
root (hd0,0)
kernel /vmlinuz-2.6.7-adeos ro root =/dev/VolGroup00/LogVol00 enforcing=0 rhgb quiet
initrd /initrd-2.6.7-adeos.img
```

All RTAI programs need a device to provide communication between hardware and software. While you test your RTAI, you may face a permission error such as “Permission Denied” due to nonexisting device /dev/rtf3. This problem is solved by running a script as below.

```
!/bin/bash
mknod –m 666 /dev/rtai_shm c 10 254
for n in ‘seq 0 9’
do
f=/dev/rtf$n
mknod – m 666 $f c 150 $n
done
```

Instead of the script, you can manually solve such a problem by running lines containing these lines

```
mknod –m 666 /dev/rtai_sch c 10 254
mknod –m 666 /dev/rtf0 c 150 0
mknod –m 666 /dev/rtf1 c 150 1
```

```
mknod –m 666 /dev/rtf2 c 150 2
mknod –m 666 /dev/rtf3 c 150 3
```

Every RTAI code needs a makefile to compose object files. If you try to form an object or executable file without makefile you will have an error such as “No target specified and no makefile”. This problem can be solved by writing a makefile.

5.4. LXRT

LXRT that allows hard real-time programs to run in user space is extension on RTAI. RTAI programs are kernel space programs. LXRT brings the kernel space program API to user land. It allows user to observe any application's outputs on user space. The real time or non real time threads such as normal Linux processes and your rtai program are ran at the same program. To run any LXRT application, rtai_lxrt and rtai_sem modules are loaded.

```
insmod /usr/src/realtime/modules/rtai_lxrt
insmod /usr/src/realtime/modules/rtai_sem
```

5.4.1. Writing a LXRT Application

Linux Scheduler must be initialized at the beginning of the program.

```
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 struct sched_param mysched;  //definition of the scheduler

 mysched.sched_priority = sched_get_priority_max(SCHED_FIFO) - 1;
 if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -1 ) { //starting
puts("ERROR IN SETTING THE SCHEDULER");
 perror("errno");
exit(1);
}
```

```
        return 0;
}
```

The threads is transformed an rtai task. This allows to use RTAI synchronization and communication primitives.

```
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

#include <rtai_lxrt.h>

int main(void)
{
 RT_TASK* task;    // Stores a handle.
 int priority=0;   // Highest
 int stack_size=0; // Use default (512)
 int msg_size=0;   // Use default (256)

 /*----------------------*/
 struct sched_param mysched;

 mysched.sched_priority = sched_get_priority_max(SCHED_FIFO) - 1;
 if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -1 ) {
  puts("ERROR IN SETTING THE SCHEDULER");
  perror("errno");
  exit(1);
 }

 task = rt_task_init( nam2num("Name"), priority, stack_size, msg_size);
 /*----------------------*/
```

```
  // your program ... anything you can think of is allowed here.
  rt_task_delete(task);
  return 0;
 }
```

Every RT_TASK in LXRT need a “Name” to which it can be referenced from kernel and userspace. Having names is an easy way to identify tasks and communication when program is running. The name must be unique. To provide automatic selection of the name *rt_get_name(0)* function is used for valid, thread safe, name.

```
unsigned int processor_mask = 0xFF; // all processors

if (!(task = rt_task_init_schmod( nam2num("Name"), priority, stack_size,
msg_size, SCHED_FIFO, processor_mask)))
{
printf("CANNOT INIT TASK %u\n", taskname(task));
exit(1);
 }
```

This function initializes the scheduler and the main() task. To make a thread function an RTAI task, add the *rt_task_init()* and *rt_task_delete()* function calls at the start and at the end of that function thread. The scheduler only needs to be initialized once from main. To avoid being swapped out, the process is locked at RAM. For this locking action, the following code before *rt_task_init()* is inserted.

```
 #include <sys/mman.h>
//...
 int main(void)
 {
 // ...
```

```
mlockall(MCL_CURRENT | MCL_FUTURE);
// ...
 }
```

The application is soft realtime and was scheduled with the native Linux FIFO scheduler. To get the deterministic realtime scheduling of RTAI, the timer must be started and the hard real time capabilities must be add to application.

Start the realtime timer (do this ONCE) rt_start_timer() : The rt_timer (or scheduler) programs the interrupt controller of the processor to generate periodic interrupts (periodic_mode) or reprograms the timer each time to fire when the next task must be started (oneshot_mode). Oneshot mode is less efficient but more flexible than periodic mode. This is further explained in the RTAI documentation. You can check if the timer is started with rt_is_hard_timer_running(). Stopping the timer is normally not necessary, unless you want to reconfigure it, which requires stopping all your hard realtime tasks.

Make your RT_TASK periodic and hard realtime rt_make_hard_realtime() and rt_make_soft_realtime() : These calls make the calling thread switch from the Linux scheduler to the RTAI scheduler and back.

```
int main(void)
{
 // ...
// after rt_init_task
if (oneshot)
rt_set_oneshot_mode();
else
rt_set_periodic_mode();
// <period_in_nanosecs> is the clock rate of the realtime scheduler
// (rt_timer) in nano seconds.
period = (int) nano2count((RTIME)period_in_nanosecs);
start_rt_timer(period);
// Periodic HardRT tasks are able to run now.
```

```
// ...
}
{
// ...
// In a RT_TASK thread function or in main, after the timer is started
if (hard_realtime) {
rt_make_hard_real_time();
}
// this determines when rt_wait_period() will wake the first time.
rt_task_make_periodic(hrt_task, rt_get_time() + period, period);

while (continue)
{
// put periodic functionality here
// ...
rt_wait_period();
}

if (hard_realtime) {
rt_make_soft_real_time();
}
// end of program/thread
// ...
}
```

6. CONCLUSIONS AND FUTURE WORK

The ECG and BP signals are represented. The continuous non invasive blood volume measurement system is introduced. A new method called oximetry which is measured by optical system gives opinion about the heart rate and blood pressure. It shows proportional change in time.

RTAI provides a real-time support for Linux. It adds small kernels in to standard kernel. These small kernels provide scheduling, and priority identification. When the priority of any application gets low value (higher priority), CPU takes the interrupts by hardware abstraction layer and decides to priority for the application. All tasks or processes of the CPU are neglected and the application is executed. The time delay becomes smaller. The completion time has no deviation.

This time concept of any RTAI module provides instantaneous response of any process. Some experiments such as counting the pulses from external pulse generator shows that the count of pulses measured by RTAI application in specific time interval is greater than the count of pulses measured by Linux application.

Furthermore, for future researches, the measurement results will be evaluated by the assistance of a doctor for non invasive optical measurement system. The data acquisition module can be implemented as portable system with RTAI. Some publications about results will be done to show the optical measurement of non invasive method is reliable or not.

BIOGRAPHY

I was born in Ordu, Turkey, in 1980. I completed the high school education in Samsun. I received the B.S. degree in Electrical and Electronics Engineering from Çukurova University, Adana, Turkey in 2003. After completion my B.S. training, I have started MSc degree in the department of Electrical and Electronics Engineering in Çukurova University and have been working there as a research assistant since 2004.

My areas of interest include software developing with Delphi, C++ and Linux operating system's kernel development tools and Real Time modeling and electronic signal implementation devices.

I am a member of Turkish Chamber of Electrical Engineers.

REFERENCES

ADVANTECH Co. Ltd., 1994. PCL 818 DAQ Card Manual.

AYDIN, A. 2004. Simple and Low-Cost Biosignal Amplifier Design for ECG and EEG, McS Thesis.

CARR, J.J., BROWN, J.M., 1981. Introduction To Biomedical Equipment Technology. John Wiley & Sons, 430s

DERIN O., 2005 A Crude Survey on Real Time Application Interface.

EVANS W.F., 1971. Anatomy and Phsicology, The Basic Principles, Englewood Cliffs, N.J., Prentice Hall, Inc.

FLEWELLING R., 1995. Noninvasive Optical Monitoring, Biomedical Engineering Handbook, Prentice Hall, Inc.

FURR S., 2002. What is Real Time and Why do I need It?. QNX Software Systems Ltd.

GEDDES L.A., 1995. The Electrocardiography, Biomedical Engineering Handbook, Prentice Hall, Inc.

INCE, N.F., 2002. A Computer Based Data Acquisition and Signal Processing System for Measuring the Baroreceptor Sensitivity.

NAGEL, J.H., 1995. The Biomedical Engineering HandBook., CRC Press Inc, pp.1185-1189.

ROUCHOUSE B., 2003. Rtai Installation Guide.

SAROLAHTI P., 2001. Real Time Application Interface, Research seminar on Real

STANKOVIC J.A, RAMAMRITHAM K, 1993. Advances in Real Time Systems, Published IEEE Computer Society.

The DIPARTIMENTO DI INGENERIA AEORSPAZIALE POITECNICO DI MILANO (DIAPM) RTAI RESEARCH GROUP, 2000. Rtai Programming Guide 1.0, Lineo Inc.

The DIPARTIMENTO DI INGENERIA AEORSPAZIALE POITECNICO DI MILANO (DIAPM) RTAI RESEARCH GROUP, 2002. Rtai Beginners Guide. (www.aero.polimi.it/rtai/documentation/articles/guide.html)

Time Linux and Java, University of Helsinki.

APPENDIX A

Specifications of the PCI 818

Analog Input (A/D converter):

- **Channels**: 16 single-ended or 8 differential, switch selectable
- **Resolution**: 12 bits
- **Input ranges (bipolar, VDC)**: ±0.625, ± 1.25, ±2.5, ±5 or ±1.25, ± 2.5, ±5, ±10
- **Overvoltage**: Continuous ±30 V max.
- **Conversion type**: Successive approximation
- **Conversion rate**: 40 KHz max.
- **Accuracy**: ±(0.01% of reading), ±1 bit
- **Linearity**: ±1 bit
- **Trigger mode**: Software trigger, on-board programmable pacer trigger or external trigger
- **Ext. trigger**: TTL compatible.

Load is 0.4 mA max. at 0.5 V and -0.05 mA max. at 2.7 V

- **Data transfer**: Program, interrupt or DMA

Analog output (D/A converter):

- **Channels**: 1 channel
- **Resolution**: 12 bits
- **Output range**: 0 to +5 (+10) V with on-board -5 (-10) V reference. x. ±10 V with external DC or AC reference
- **Reference**:

  Internal: -5 V or -10 V

  External DC or AC: ±10 V max.
- **Conversion type**: 12 bit monolithic multiplying
- **Linearity**: ±0.5 bit
- **Output drive**: ±5 mA max.

- **Settling time**: 5 microseconds

Digital input

- **Channel**: 16 bits
- **Level**: TTL compatible
- **Input voltage**:

  Low: 0.8 V max.

  High: 2.0 V min.
- **Input load**:

  Low: 0.4 mA max. at 0.5 V

  High: 0.05 mA max. at 2.7 V

Digital Output

- **Channel**: 16 bits
- **Level**: TTL compatible

- **Output voltage**:

  Low: Sink 8 mA at 0.5 V max.

  High: Source -0.4 mA at 2.4 V min.

Programmable timer/counter

- **Device**: Intel 8254 or equivalent
- **Counters**: 3 channels, 16 bit.

  2 channels are permanently configured as programmable pacers

  1 channel is free for your applications
- **Input, gate**: TTL/CMOS compatible
- **Time base**:

  Pacer channel 1: 10 MHz or 1 MHz, switch selectable

  Pacer channel 2: Takes input from channel 1

  Pacer channel 0: Internal 100 KHz or external clock (10 MHz max).

  Source selected with Timer/Counter Enable Register (BASE+10)
- **Pacer output**: 0.00023 Hz (71 minutes/pulse) to 2.5 MHz

Interrupt channel

- **Level**: IRQ 2 to 7, software selectable
- **Enable**: Via INTE bit of Control Register (BASE+9)

DMA channel

- **Level**: 1 or 3, jumper selectable
- **Enable**: Via DMAE bit of Control Register (BASE+9)

General

- **Power consumption**:

  +5 V: 210 mA typical, 500 mA max.

  +12 V: 20 mA typical, 100 mA max.

  -12 V: 20 mA typical, 40 mA max.
- **I/O connector**: 20 pin post headers for I/O connection. Adapter available to convert to DB-37 connector
- **Analog input/output/counter connector**: DB-37
- I/**O base**: Requires 16 consecutive address locations. Base address definable by the DIP switch SW1 for address line A9-A4. (Factory setting is Hex 300).

APPENDIX B

RTAI Hard Real-Time LXRT Parallel Port Example

The parallel port interrupt handler LXRT code is illustrated below.

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <sched.h>
#include <signal.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <rtai_lxrt.h>
#include <rtai_sem.h>
#include <rtai_usi.h>
#include <sys/io.h>
#define PARPORT_IRQ  7
#define BASEPORT 0x378

static SEM *intsem, *dspsem;
static volatile int end = 1;
static volatile int endsquare = 1;
static volatile int ovr, intcnt, retval, maxcnt;

#define PERIOD 1000000000

static void *timer_handler(void *args)
{
RT_TASK *handler;
        if (!(handler = rt_task_init_schmod(nam2num("HANDLR"), 0, 0, 0,
SCHED_FIFO, 0xF))) {
```

```
            printf("CANNOT INIT HANDLER TASK > HANDLR <\n");
            exit(1);
}
rt_allow_nonroot_hrt();
mlockall(MCL_CURRENT | MCL_FUTURE);
rt_make_hard_real_time();
end = 0;
while (!end) {
            do {
                        ovr = rt_wait_intr(intsem, (void *)&retval);
                        hard_sti();
                        if (end) break;
                        // overrun processing, if any, goes here
                        if (ovr > 0) {
                                    rt_sem_signal(dspsem);
                        }
                        /* normal processing goes here */
                        intcnt++;

                        rt_sem_signal(dspsem);                  // notify main()

                        rt_ack_irq(PARPORT_IRQ);
            } while (ovr > 0);
            rt_pend_linux_irq(PARPORT_IRQ);
}
rt_make_soft_real_time();
rt_task_delete(handler);
intcnt = maxcnt;
return 0;
}

static void *square_handler(void *args)
{
RT_TASK *handler;
```

```
RTIME period;

        if (!(handler = rt_task_init_schmod(nam2num("SQHDLR"), 0, 0, 0,
SCHED_FIFO, 0xF))) {
        printf("CANNOT INIT HANDLER TASK > SQHDLR <\n");
        exit(1);
}
rt_allow_nonroot_hrt();
mlockall(MCL_CURRENT | MCL_FUTURE);

rt_set_oneshot_mode();
start_rt_timer(0);
        period = nano2count(PERIOD);
        rt_make_hard_real_time();
endsquare = 0;
rt_task_make_periodic(handler, rt_get_time() + period, period);

while ( !endsquare ) {
        outb_p(0, BASEPORT);
        rt_task_wait_period();
        outb_p(255, BASEPORT);
        rt_task_wait_period();
}

stop_rt_timer();
rt_make_soft_real_time();
rt_task_delete(handler);
intcnt = maxcnt;
return 0;
}

static pthread_t thread, squarethread;

int main(void)
```

```
{
  RT_TASK *maint;

printf("GIVE THE NUMBER OF INTERRUPTS YOU WANT TO COUNT: ");
scanf("%d", &maxcnt);

if (!(maint = rt_task_init(nam2num("MAIN"), 1, 0, 0))) {
        printf("CANNOT INIT MAIN TASK > MAIN <\n");
        exit(1);
}
// Create semaphore for notification when interrupt occurs
// In the thread we wait with rt_wait_intr until an interrupt occurs
if (!(intsem = rt_sem_init(nam2num("IRQSEM"), 0))) {
        printf("CANNOT INIT SEMAPHORE > IRQSEM <\n");
        exit(1);
}
// create semaphore to notify main() when interrupt occurs
if (!(dspsem = rt_sem_init(nam2num("DSPSEM"), 0))) {
        printf("CANNOT INIT SEMAPHORE > DSPSEM <\n");
        exit(1);
}
if (iopl(3)) {  // ask for permission to access the parallel port from user-space
        printf("iopl err\n");
        rt_task_delete(maint);
        rt_sem_delete(intsem);
        rt_sem_delete(dspsem);
        exit(1);
}

outb_p(0x10, BASEPORT + 2); //set port to interrupt mode; pins are output

// create thread
pthread_create(&thread, NULL, timer_handler, NULL);
// wait until thread went to hard real time
```

```
        while (end) {
                usleep(100000);
        }

        // create squarethread
        pthread_create(&squarethread, NULL, square_handler, NULL);
        // wait until thread went to hard real time
        while (endsquare) {
                usleep(100000);
        }
        // request the interrupt and bind it to semaphore
        rt_request_global_irq(PARPORT_IRQ, intsem, USI_SEM);
        rt_startup_irq(PARPORT_IRQ);
        rt_enable_irq(PARPORT_IRQ);

        while (intcnt < maxcnt) {
                rt_sem_wait(dspsem);
                printf("RETVAL %d, OVERRUNS %d, INTERRUPT COUNT %d\n",
retval, ovr, intcnt);
        }
        end = 1;
        endsquare = 1;
        printf("TEST ENDS\n");
        outb_p(0, BASEPORT);
        outb_p(255, BASEPORT); // generate final interrupt to unblock rt_wait_intr
        outb_p(0, BASEPORT);
        pthread_join(thread, NULL);
        pthread_join(squarethread, NULL);
        rt_free_global_irq(PARPORT_IRQ);
        rt_task_delete(maint);
        rt_sem_delete(intsem);
        rt_sem_delete(dspsem);
        return 0;
        }
```