# DESIGN AND IMPLEMENTATION OF A PC-BASED HEART RATE VARIABILITY AND RESPIRATION RECORDING SYSTEM

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Okay Tunca Korkmaz

October, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Y. Ziya İder (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Hayrettin Köymen

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. B. Murat Eyüboğlu

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute Engineering and Science

# ABSTRACT

## DESIGN AND IMPLEMENTATION OF A PC-BASED HEART RATE VARIABILITY AND RESPIRATION RECORDING SYSTEM

Okay Tunca Korkmaz
M.S. in Electrical and Electronics Engineering
Supervisor: Prof. Dr. Y. Ziya İder
October, 2005

Determination of the exact effects of respiration rate on the heart rate variability requires comprehensive experimental studies. In this context, these two quantities should be analyzed simultaneously. In this thesis, design and implementation of a new PC-based heart rate variability recording system with respiration is described. The respiration rate is detected by a thermocouple placed in a nasal tube, whereas a single-channel electrocardiogram (ECG) signal is recorded for heart rate variability analysis in the system. The PC-based data acquisition part, which was designed in compliance to patient safety standards, has a Universal Serial Bus (USB) interface. The speed of data acquisition and patient comfort during recording make the system advantageous. The HRVs of different patients in both spontaneous and controlled breathing conditions were analyzed after short-term recordings with the system. Differences were observed in both its time-domain parameters and power spectral components.

*Keywords:* Heart Rate Variability, Respiration, Data Acquisition, USB.

# ÖZET

# PC TABANLI KALP ATIŞ HIZI DEĞİŞKENLİĞİ VE SOLUNUM KAYIT SİSTEMİ TASARIMI VE GERÇEKLEŞTİRİLMESİ

Okay Tunca Korkmaz
Elektrik- Elektronik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Y. Ziya İder
Ekim, 2005

Solunum hızının kalp atış hızı değişkenliği üzerindeki kesin etkilerinin belirlenmesi, kapsamlı deneysel çalışmalar gerektirmektedir. Bu bağlamda, bu iki niceliğin eşzamanlı olarak kaydedilmesi şarttır. Bu tezde, yeni bir PC tabanlı solunum ve kalp atış hızı değişkenliği kayıt sisteminin tasarımı ve gerçekleştirilmesi anlatılmaktadır. Sistemde, kalp atış hızı değişkenliğinin analizi için tek kanallı elektrokardiogram (EKG) sinyali kaydedilirken, solunum hızı burun ucuna tüple yerleştirilen bir ısılçift ile algılanmaktadır. Hasta güvenliği standartlarına uygun olarak tasarlanan PC tabanlı veri toplama kısmı, bir Evrensel Seri Yol arayüzüne sahiptir. Veri toplama hızı ve kayıt esnasındaki hasta konforu, sistemi avantajlı hale getirmektedir. Sistemi kullanarak kısa süreli kayıtların ardından, kendiliğinden ve kontrollü nefes alma durumlarında, farklı hastaların kalp atış hızı değişkenlikleri analiz edildi. Hem zaman bölgesi parametrelerinde hem de güç spektrumu bileşenlerinde farklılıklar gözlendi.

*Anahtar sözcükler*: Kalp Atış Hızı Değişkenliği, Solunum, Veri Toplama, USB.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Objective and Scope

Heart Rate Variability (HRV) attracts significant interest from researchers due to its use as a risk factor for predicting mortality after acute myocardial infarction.

Variation of the heart rate in respiration frequency range, called respiratory sinus arrhythmia [8], was first observed by Ludwig as early as 1840s. Following this observation, several theories were developed regarding respiration-induced heart rate control mechanisms [13]. Mathematical models of the respiratory role in cardiovascular regulation were also developed [14]. However, the exact mechanisms of respiration induced HRV are still being investigated.

A need for respiration to be recorded simultaneously with the electrocardiogram (ECG) signal has emerged from recent studies directed toward obtaining a clear understanding of respiratory effects on HRV.

Schipke et al [12] analyzed HRV at six different respiration rates to demonstrate the effect of respiration rate on HRV. In this study, the respiration rate was observed using a patient-monitored stopwatch, which is not an entirely accurate means of measurement.

Bernardi et al [4] investigated the effects of controlled breathing, mental stress and speech on HRV. They showed alterations in the variability and spectral content of the RR intervals, influenced by changes in the respiratory pattern. In that study, since a facial mask would itself have affected respiration, they could not use a mask for respiratory recordings. Instead they used an inductive belt plethysmograph to measure the respiration rate, which also causes stress to the patient.

The objective of this study is the design and implementation of a new real-time system which records ECG and a respiratory signal in order to find the instantaneous HRV and respiration rate without causing discomfort to the patient. The block diagram of the system is shown in figure 1.1. ECG and respiratory signals are entered into the front-end circuits and then sent to a PC via the USB line by an isolated data acquisition system.

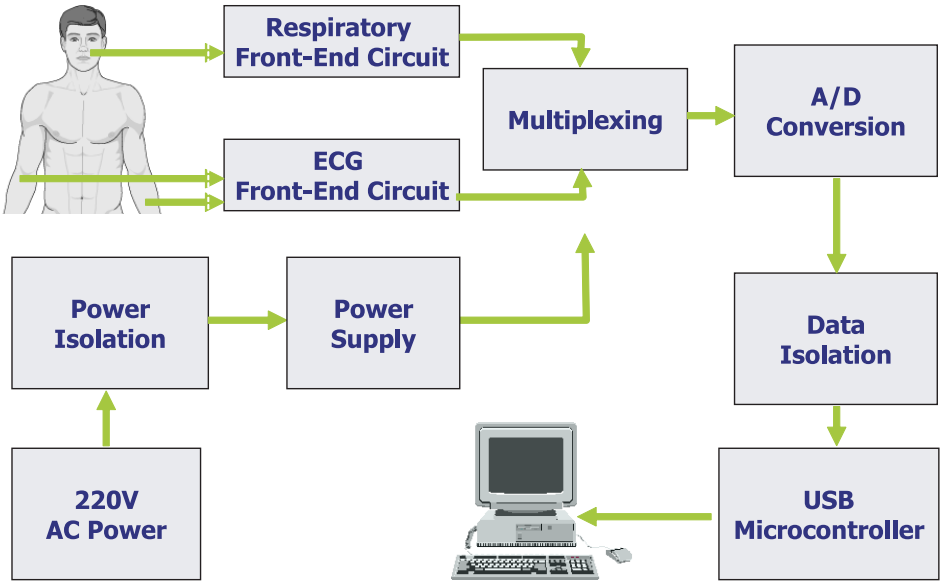



Figure 1.1: Block Diagram of the Recording System

## 1.2 Organization of the Thesis

In the thesis, firstly the entire recording system will be presented in order to give a clear understanding of the design methodology and implementation. Chapter 2 introduces ECG and respiratory signals. The design and implementation of front-end circuits for these bioelectric sources are shown. Design and implementation of the isolated data acquisition system is explained in chapter 3. Design simulations are shown in chapter 4. Performance and safety tests, which were performed on the device for compliance with the standards are described in chapter 5.

Secondly, chapter 6 explains the digital signal processing techniques used for recorded data analysis.

Next, the recording experiments on patients and results are shown in chapter 7.

Finally, chapter 8 includes discussion and chapter 9 includes conclusion of this thesis.

# Chapter 2

# ECG and Respiratory Signal Front-End Circuit

## 2.1 ECG Signal Front-End Circuit Design

During the beating of the heart, heart tissue acts as a volume current source to generate a time-varying electrical field inside the conducting body volume. Due to this electrical field, a potential difference exists between any two points on the body surface, called the Electrocardiogram (ECG) signal.

The ECG signal is low in magnitude ($1 - 2mV_{pp}$) and needs to be amplified. Furthermore, there is a need for signal conditioning circuits in order to minimize the effects of interfering signals and noise.

In most cases, operational amplifiers (opamps) are used as the basic circuit element in amplifiers of signal conditioning circuits. The electrical equivalent model of an opamp is given in figure 2.1, where $V1$ and $V2$ are input terminal voltages, $A_{dm}$ differential mode gain and $A_{cm}$ common mode gain parameters of the opamp.

Ideally opamps should have infinite input impedances (i.e $Z_{in1} = Z_{in2} = \infty$),

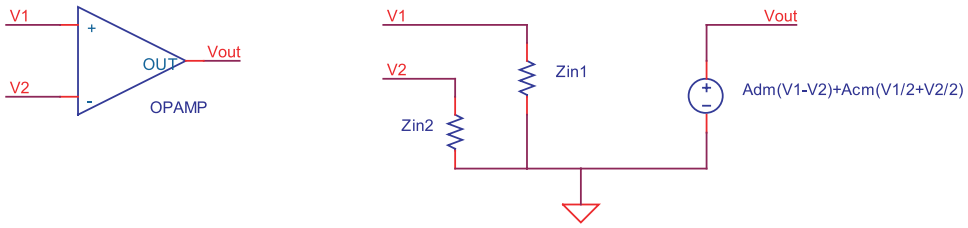



Figure 2.1: Electrical Equivalent Circuit of an OpAmp

infinite differential gain ($A_{dm} = \infty$), zero common mode gain ($A_{cm} = 0$), zero input bias currents and zero input offset voltages. However, in practice there are unbalanced circuitries in opamps, preventing the achievement of these ideal conditions. In the following, we assumed that the opamps used in an ECG circuit have finite input impedances and differential gain and non-zero common mode gain.

### 2.1.1 Basic Differential Amplifiers

In ECG instrumentation devices, the amplification circuitry is based on using a differential amplifier with a reasonable gain [15]. The schematic of such an amplifier is given in figure 2.2. The input-output relationship of a differential amplifier can be easily found using Kirchoff's laws and the electrical equivalent circuit of an opamp given in figure 2.1. When $Z_{in} \gg max\{R1, R2, R3, R4\}$, input terminal voltages of the opamp are

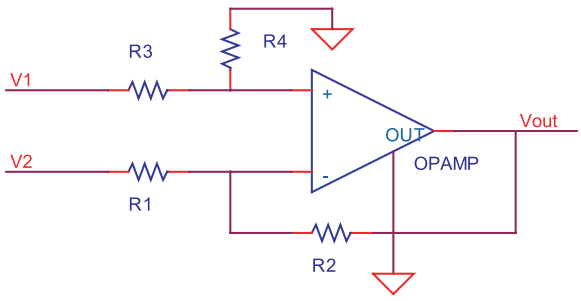



Figure 2.2: A Basic Differential Amplifier

$$V^+ = \frac{R4}{R3 + R4} V1 \tag{2.1}$$

$$V^- = \frac{R2}{R1+R2}V2 + \frac{R1}{R1+R2}V_{out} \tag{2.2}$$

$$V_{out} = A_{dm}(V^+ - V^-) + A_{cm}(\frac{V^+ + V^-}{2}) \tag{2.3}$$

From equations 2.1, 2.2 and 2.3

$$V_{out} = A_{dm}(\frac{R4}{R3+R4}V1 - \frac{R2}{R1+R2}V2 - \frac{R1}{R1+R2}V_{out}) + A_{cm}(\frac{\frac{R4}{R3+R4}V1 + \frac{R2}{R1+R2}V2 + \frac{R1}{R1+R2}V_{out}}{2})$$

$$V_{out}\frac{A_{dm}R1}{R1+R2} \simeq A_{dm}(\frac{R4}{R3+R4}V1 - \frac{R2}{R1+R2}V2) + A_{cm}(\frac{R4}{R3+R4}\frac{V1}{2} + \frac{R2}{R1+R2}\frac{V2}{2})$$

Selecting $R1 = R3$ and $R2 = R4$,

$$V_{out} = \frac{R2}{R1}(V1 - V2) + \frac{R2}{R1}\frac{A_{cm}}{A_{dm}}(\frac{V1+V2}{2}) \tag{2.4}$$

The Common Mode Rejection Ratio (CMRR) parameter of an opamp is defined as

$$CMRR = \frac{A_{dm}}{A_{cm}} \tag{2.5}$$

Thus, equation 2.4 becomes

$$V_{out} = \frac{R2}{R1}(V1 - V2) + \frac{R2}{R1}\frac{1}{CMRR}(\frac{V1+V2}{2}) \tag{2.6}$$

### 2.1.2 Properties of an ECG Amplifier

#### 2.1.2.1 Common-Mode Voltage Rejection

There are very small stray capacitances between a human body and power lines and between a human body and the earth ground. These capacitances produce a common mode voltage on the body, detected on both electrodes. Since the ECG signal is a differential signal that is a few millivolts in amplitude, this common mode voltage may interfere with the ECG signal due to non-ideality of opamps used, i.e $Acm \neq 0$.

Imbalance in differential amplifier resistors enhances common mode disturbances on the output signal $V_{out}$ [10]. Also, the CMRR of the opamp used in the differential amplifier is an important parameter when selecting the opamps. If the

CMRR is not high enough, despite matching the resistors perfectly, the designer cannot achieve good performance in rejection of common mode voltages.

In order to minimize this interference, *the three electrode configuration* is used in ECG amplifier designs. Figure 2.3 illustrates this configuration. Common mode voltage on the body ($V_B$) is caused by stray capacitances $C_{S1}$ and $C_{S2}$. Two electrodes are connected to the instrumentation amplifier inputs, whereas the third electrode is connected between the right leg of the body and the amplifier common via resistance $Z_{RL}$. In this circuit the common mode input of the amplifier is

$$V_B = 220V \times \frac{\frac{1}{jwC_{S2}} \parallel Z_{RL}}{\frac{1}{jwC_{S2}} \parallel Z_{RL} + \frac{1}{jwC_{S1}}} \tag{2.7}$$

Since $Z_{RL} \ll \frac{1}{jwC_{S1}}$ and $Z_{RL} \ll \frac{1}{jwC_{S2}}$,

$$V_B = 220V \times \frac{Z_{RL}}{\frac{1}{jwC_{S1}}} \tag{2.8}$$

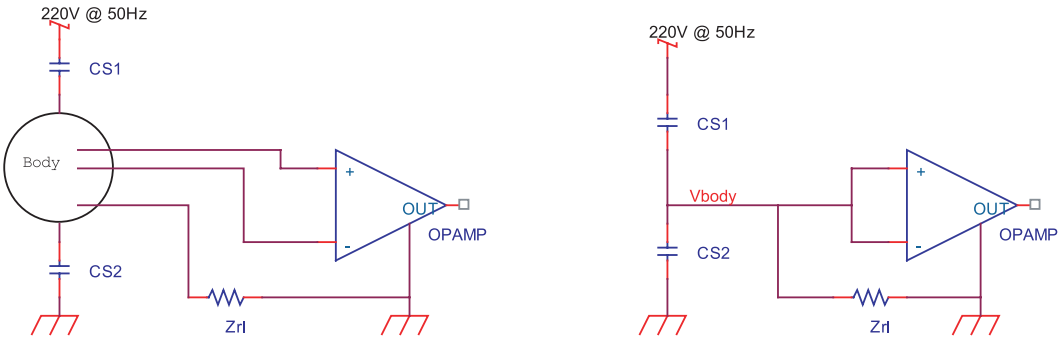



Figure 2.3: The Three Electrode Configuration and its Common Mode Equivalent Circuit

The common mode output of this configuration is negligible in comparison with the ECG signal. However, patient safety needs are not met. According to the "American National Standard: Safe Current Limits for Electromedical Apparatus", the patient sink risk current is limited to $50\mu A$ for a single fault condition [6]. If the power lines are touched, a current $I = \frac{220V}{Z_{RL}}$ flows through the patient, exceeding these safe current limits for properly selected $Z_{RL}$ values. Therefore, all circuitry of the device should be isolated from the earth ground in order to protect the patient.

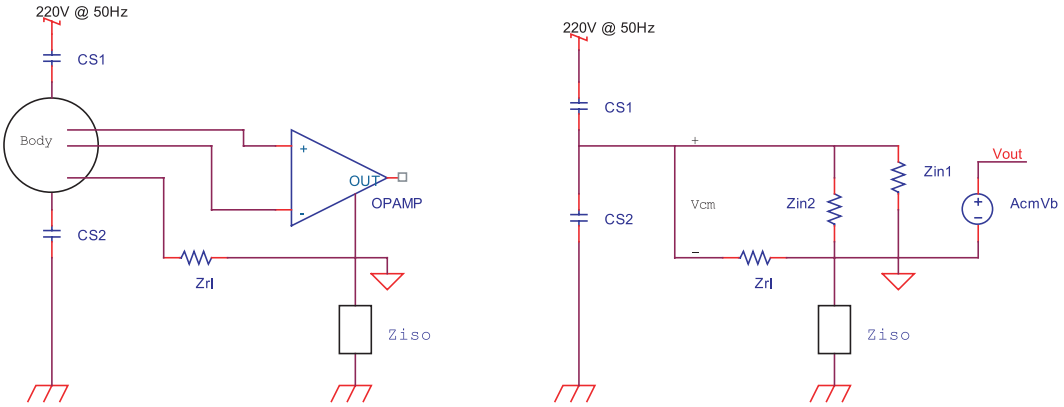



Figure 2.4: An Isolated Amplifier Circuit

Figure 2.4 shows the isolated amplifier circuitry. Isolation impedance is represented by $Z_{iso}$. The common mode voltage on the amplifier terminals when $Z_{in1} = Z_{in2} = Z_{in}$ is,

$$V_{CM} = 220V \times \frac{Z_{RL} \parallel \frac{Z_{in}}{2}}{Z_{RL} \parallel \frac{Z_{in}}{2} + Z_{iso}} \times \frac{\frac{1}{jwC_{S2}} \parallel (Z_{RL} \parallel \frac{Z_{in}}{2} + Z_{iso})}{[\frac{1}{jwC_{S2}} \parallel (Z_{RL} \parallel \frac{Z_{in}}{2} + Z_{iso})] + \frac{1}{jwC_{S1}}} \tag{2.9}$$

Since $Z_{iso} \gg Z_{in} \gg Z_{RL}$,

$$V_{CM} \cong 220V \times \frac{Z_{RL}}{Z_{iso}} \times \frac{\frac{1}{jwC_{S2}} \parallel Z_{iso}}{(\frac{1}{jwC_{S2}} \parallel Z_{iso}) + \frac{1}{jwC_{S1}}} \tag{2.10}$$

which is negligible with respect to differential ECG signal,when $Z_{RL} \ll Z_{iso}$.

When the patient touches the power line, the current flowing through his body can be calculated as

$$I = \frac{220V}{(Z_{RL} \parallel \frac{Z_{in}}{2}) + Z_{iso}} \cong \frac{220V}{Z_{iso}} \tag{2.11}$$

which does not exceed safety limits for high isolation impedances. Therefore, the patient is protected from device failure or other accidents. Isolation circuits must be designed for both the power supply and the signals connected to data acquisition system. This will be discussed in next chapter.

#### 2.1.2.2 Other Noise Sources

In addition to common-mode voltage, there are other noise sources that interfere with the ECG signal. These interferences could be reduced by extra design

considerations regarding the hardware. Some examples are listed below:

- The magnetic field generated by current carrying power lines induces voltage in the loop between the body, electrode leads and the front-end circuit. That voltage could be reduced by *twisting the lead cables.*
- Due to stray capacitances between the body and the power lines, displacement currents flow through the electrode lead cables, generating differential voltages between the cables. By *Shielding the cables* and *connecting the shield* to the amplifier common, this noise is reduced.
- Currents induced by stray capacitances flow through the body, generating differential voltages due to the resistivity of the tissues. *Careful placement of electrodes* prevents this type of noise.
- Higher frequency Electromyogram (EMG) signals from the thorax and arms are added to differential signal between electrodes. *Low-pass filtering* with the cut-off frequency $f_0 = 35$Hz lowers this interference.
- Motion artifacts due to breathing change electrode positions and electrode contact potentials. This effect is called baseline wander. Firmly attached electrodes and *high-pass filtering* above $f_0 = 0.05$Hz overcomes this low frequency interference.

#### 2.1.2.3 Gain and Bandwidth

According to the "American National Standard for Diagnostic Electrocardiographic Devices", an ECG device should be capable of responding and displaying differential voltages up to $\pm 5mV$ from a dc offset voltage in the range of $\pm 300mV$, when applied to any lead [5].

The frequency response of the device to satisfy the performance requirements of the standard is $-30\%$ between $40Hz$ and $100Hz$ relative to its response at $10Hz$. Therefore, the upper cut-off frequency (3dB) of the device should be at

minimum $100Hz$. The lower cut-off should be at $0.05Hz$ to overcome baseline wander.

### 2.1.3 Design of the ECG Front-End Circuit

The ECG front-end design was developed taking the interference and safety issues discussed above into consideration. Figure 2.5 illustrates this design.

The amplifier has three stages. In multiple-stage designs, common mode rejection is determined by using differential and common mode gains of all these stages [9]. That makes it necessary to design in each stage very carefully.

In the first stage, two opamps are used as buffers, in order to have high input impedance at lead connections to the patient's arms. $OP07$ opamps are used since their offset voltages are very low and it is easier to match them. $10M\Omega$ resistances ($R9$ and $R10$) are used to minimize input impedance for smaller common mode voltages. The third electrode lead input from the right leg of the patient is connected to the amplifier common through $Z_{RL} = R5 = 10K\Omega$. This stage has a unity gain ($A_1 = 1$).

The second stage is the differential amplifier, where the differential gain is $A_2 = \frac{R4}{R1} = \frac{R3}{R2} = 25.5$. These resistors are matched by the Agilent $34401A$ Multimeter down to %1 tolerance. $OP07$ is used for its high CMRR (the parameters are given in the Appendices). The gain of this stage is small, to prevent the opamp from entering saturation due to electrode contact potentials.

After the differential amplifier, there is a $1\mu F$ coupling capacitor ($C3$) to block DC offset from electrode contact potential, which also forms a high-pass filter with a $3.3M\Omega$ resistor($R8$) to discard motion artifact interference on the signal.

The last stage is a non-inverting amplifier circuit. The gain of this stage is $A_3 = \frac{R7}{R6} = 32$. A $3.3M\Omega$ resistor($R12$) is used to equalize input impedances at the opamp terminals, reducing input offset bias current error. $R7$ and $C1$ form a

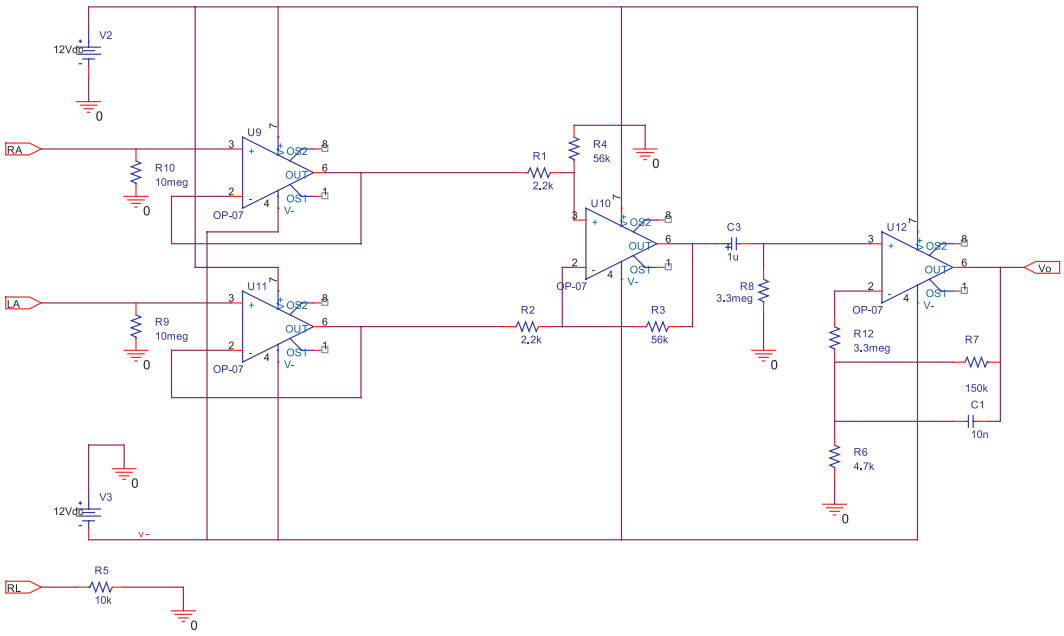



Figure 2.5: The ECG Amplifier Circuit

low-pass filter with the cut-off frequency $f_c = 107Hz$ to filter frequencies above ECG signal.

## 2.2 Respiratory Signal Front-End Circuit Design

The respiration process does not involve an independent electrical signal so it can be measured by various other physical quantities. These are the flow rate of expired or inspired air, instantaneous volume changes during respiration and the temperature difference between body and its environment.

Spirometry is the classical respiratory function test to measure the amount of air leaving and entering the body during breathing. Although it satisfies quantitative needs for tidal and forced expiratory volume (FEV) measurements, it cannot collect data for calculation of the absolute lung volume and similar parameters. Since its probe causes discomfort to the patient, spirometry cannot be used in spontaneous breathing conditions.

Hot-wire anemometry is another way of detecting respiratory action. Heat transfer from a heated surface depends on the flow passing through it. This method is based on this dependence. In hot-wire anemometry, gas flow is sensed on a biased platinum wire. The wire is heated by current passing on it with a feedback circuit. Due to cooling effect of the gas, resistance of the wire changes, creating an imbalance in a Wheatstone bridge circuit to change output voltage. This voltage is proportional to the flow rate of the gas.

Other methods to detect respiration rely on the principle of the temperature difference between the human body and the ambient environment. When the ambient temperature on the sensor is $24^oC$ for example, during exhalation, $36^oC$ gas is blown out from the body, creating an instantaneous increase in sensor temperature. During inhalation, cold air flows into the body, causing a decrease in the temperature of the sensor as it returns to its resting value. This variation can be sensed by thermal sensors such as Resistance Temperature Detectors (RTD), thermocouples and thermistors.

Although they are not methods to quantitatively measure volume and flow of respiration, thermal sensors provide accuracy in measuring respiration rates. Initiation of inspiratory and expiratory periods can be easily extracted from the output of these systems, using edge detection algorithms.

| | RTDs | Thermocouples | Thermistors |
|---|---|---|---|
| Temperature range | $-260$ to $850^oC$ | $-270$ to $1800^oC$ | $-80$ to $150^oC(typical)$ |
| Sensor Cost | Moderate | Low | Low |
| Linearity | Best | Moderate | Poor |
| Sensitivity | Moderate | Low | Best |
| Size | Large | Small | Moderate |
| Response | Slow | Fast | Moderate |

Table 2.1: Comparison of Temperature Sensors

### 2.2.1 Thermocouples

In this project, the aim was to record respiration and heart rate variability simultaneously. Thus, accurate detection of the timing between respiratory phases has

higher priority for us than precise volume or flow measurements. Thermal sensors provide this accuracy. Biomedical safety is another advantage of these devices since they measure inspired and expired air through the mouth or nose, instead of using an electrical connection to the body surface. Therefore, we decided to use thermal sensors to detect respiration.

The basic properties of thermal sensors are given in table 2.1. Although they show lower sensitivity characteristics with respect to other sensors, thermocouples are more suitable for this project due to their fast response. Furthermore, their small size and light weight, and the ease of mounting the probes on the patient make thermocouples more convenient.

As a result, thermocouples were selected for the project as thermal sensors in order to detect respiratory activity.

#### 2.2.1.1 The Working Principle of Thermocouples

In a closed circuit of two dissimilar metals, if their junctions are kept at different temperatures, a current flows. This current leads to a temperature dependent concentration of valence electrons on each metal. At the junctions, these valence electrons diffuse to create an electromotive force (emf) between metals of opposite polarities. The formation of this proportionally temperature-dependent emf is called Seeback effect [11].

The sensitivity of thermocouples is the property relating the emf of the Seeback effect with temperature difference such that

$$S = \frac{e}{t} = a + bt + ct^2 \tag{2.12}$$

where a,b and c are constants and t is the temperature difference. Its unit is $\mu V/^oC$. If t is small,

$$e \cong at \tag{2.13}$$

Despite their nonlinear sensitivity characteristics, thermocouples could be considered almost linear in low temperature variations, as in equation 2.13. The

*Type-J Thermocouple* was selected for this project as the sensing element, with its parameters shown in table 2.2.

| Parameter | Value |
|---|---|
| Metal Alloys Used | Iron(+)/Constantan(-) |
| Composition | $Fe/57\%Cr - 43\%Ni$ |
| Sensitivity($\mu V/^oC$) | 45 to 57 |
| Accuracy($^oC$) | $\pm 2.2$ |
| Range($^oC$) | $-18$ to $+276$ |

Table 2.2: Parameters of Type J (ISA Standard) Thermocouple

#### 2.2.1.2 Design Applications

In figure 2.6, junction $J1$ is called the *hot junction*, since it is exposed to the medium whose temperature is to be measured. Reference junctions $J2$ and $J3$ are formed between the copper lines of the circuit and the thermocouple leads, when they are connected to an electrical circuit. If these junctions are kept at the same temperature, no unwanted effect occurs on emf and they can be considered together as the *cold junction* (reference junction). A differential voltage is formed between these two copper wires related to the temperature difference $\Delta t$=$T1$-$T2$ where $T1$ and $T2$ are temperatures of junctions $J1$ and $J2$(or $J3$) respectively.

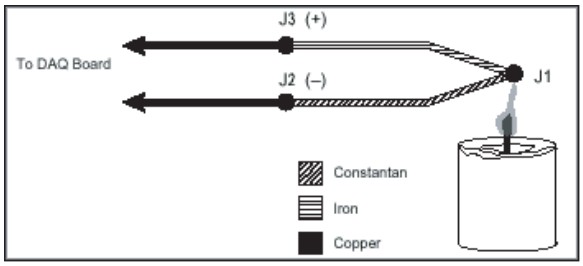



Figure 2.6: Type J Thermocouple Sensor

Thermocouple sensors produce an emf of a few millivolts in small temperature variations. That's why an amplification and noise rejection circuit should be used instead of directly using this voltage.

### 2.2.2 Design of the Respiratory Signal Front-End Circuit

For use in respiratory measurements, the thermocouple should provide significant variation of output between the body temperature ($T1$) and the temperature of the ambient environment(T2).

The amplification circuit can be described as

$$V_{out} = A * V_{in} + V_{err} \tag{2.14}$$

where $V_{out}$ is the analog output, $V_{in}$ is the emf produced by the thermocouple, $V_{err}$ is the error voltage and $A$ is the amplifier gain.

The analog multiplexer of the data acquisition system designed in this study accepts an input range of $\pm 5V$. Therefore

$$V_{out,max} = 5V \tag{2.15}$$

Type J thermocouple produces an emf

$$V_{in} \cong 50\mu V/^oC * \Delta t \tag{2.16}$$

where $\Delta$t=$T1$-$T2$. If minimum temperature $T2$ of the experimental setup is $15^oC$ and body temperature is $36^oC$,

$$\Delta t_{max} = 36 - 15 = 21^oC \tag{2.17}$$

If equation 2.16 is evaluated:

$$V_{in,max} \cong 50\mu V/^oC * \Delta t_{max} = 1,050mV \tag{2.18}$$

Using equations 2.14, 2.15 and 2.18

$$A_{max} = \frac{V_{out,max}}{V_{in,max}} = \frac{5V}{1,05mV} = 4761 \tag{2.19}$$

without considering the effect of $V_{err}$ term in equation 2.14. The gain could not exceed this limitation of the design. $\pm 12VDC$ voltages are used as op-amp supply sources, to safely ensure that the amplifiers are not saturated.

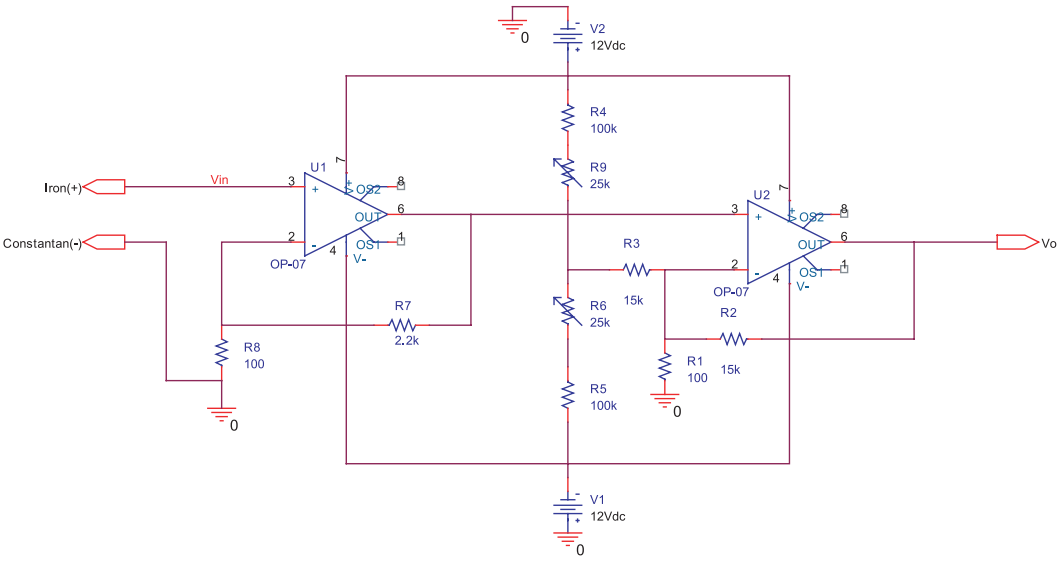



Figure 2.7: The Respiratory Signal Front-End Circuit

Error voltage $V_{err}$ is related to the input offset voltage and the input offset bias currents of the operational amplifier used in the design. Noise error due to 50Hz interference also exists.

Figure 2.7 shows the thermocouple amplifier design used in this project. Since the gain of the amplifier to be designed was high, amplification has been divided into two non-inverting stages. Error voltage of the two stages is

$$V_{err} = A2 * (A1 * V_{err1} + V_{err2}) \tag{2.20}$$

where $A1$ and $A2$ are gains, and $V_{err1}$ and $V_{err2}$ are input offset voltages of the first and second stages, respectively. By selecting $A1$ small and adding an offset voltage cancellation sub-circuit in the second stage, $V_{err}$ was minimized. Gains of stages in the design are

$$\begin{aligned} A1 &= \frac{R7}{R8} + 1 = 23 \\ A2 &= \frac{R2}{R1 \parallel R3} + 1 \cong 151 \end{aligned}$$

Therefore, overall gain of amplifier is

$$A = A1 * A2 = 3473 \tag{2.21}$$

$R3 \gg R1$ is selected to prevent non-inverting gain from decreasing. Since the magnitude and polarity of the input offset voltage cannot be predicted, a $50K$ trimpot is used to adjust cancellation voltage.

$R1$ and $R8$ resistances were selected small enough to minimize errors arising from input offset bias currents of the amplifiers used.

OP07 opamps were used in amplification since they have very low input offset voltage and high common-mode rejection. OP07 blocks noise in terminals with very high common mode input resistance. Both error voltages due to 50Hz power interference and input offset were decreased by selecting these appropriate amplifiers. Also, the shield of the thermocouple lead wires was connected to the circuit ground to lower 50Hz interference from stray capacitances.

## 2.3 Implementation of the Design

The design was first implemented on breadboards, in order to make corrections and improvements. The final implementation of both the ECG and respiratory signal front-end circuits was done on the same printed circuit board (PCB), after a schematic and layout design procedure using *Cadsoft Eagle* 4.09 PCB design software. PCB is made from FR4 Copper on its faces and insulator inside. Electrical connections between its faces were established by conducting holes. IR mask process was also executed on the PCB.

A 9-pin D-SUB (or RS232) male connector was selected as the input connector for the ECG leads. The leads are transmitted by a shielded cable, the shield of which is grounded by the connector. Therefore, 50Hz noise on the leads due to stray capacitances between the leads and nearby power lines is reduced. A 2-pin connector, chosen for its small size, ease of plugging and connectivity, is used in the thermocouple probe connection. Figure 2.8 shows the printed board implemented as the ECG and respiratory front-end circuit. The thermocouple sensor was placed inside a nasal tube as shown in figure 2.9, so recordings of spontaneous breathing could be done in comfort.

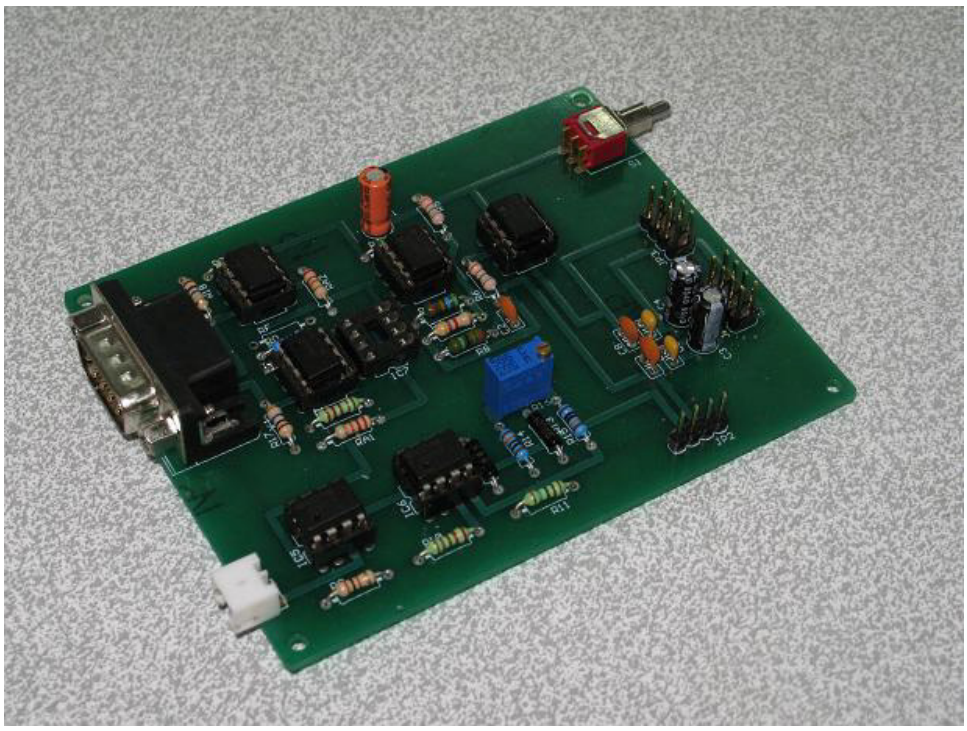


Figure 2.8: Implementation of the Front-End Circuit

Figure 2.9: Implementation of the Thermocouple Probe

# Chapter 3

# Isolated Data Acquisition System

Amplified and conditioned ECG and respiratory signals should be transferred into a PC for monitoring and signal processing. For this reason, a data acquisition card was designed meeting biomedical safety standards.

## 3.1 Hardware Design Materials and Methods

Data acquisition comprises Multiplexing, Analog to Digital Conversion(A/D), a microcontroller(M/C) for communication with PC and a bus interface. However, since our system is concerned with biomedical signals, there also exists an isolation circuit.

Figure 3.1 illustrates this system as blocks. First block, called the "Isolated Circuit", is electrically connected to the patient via the signal amplification and conditioning card. Therefore, it requires isolation from the earth ground to meet patient safety conditions. Second block is the "Non-Isolated Circuit" which is connected to a PC by the Universal Serial Bus (USB) line. Isolation is done for both data and control signals between these two blocks.

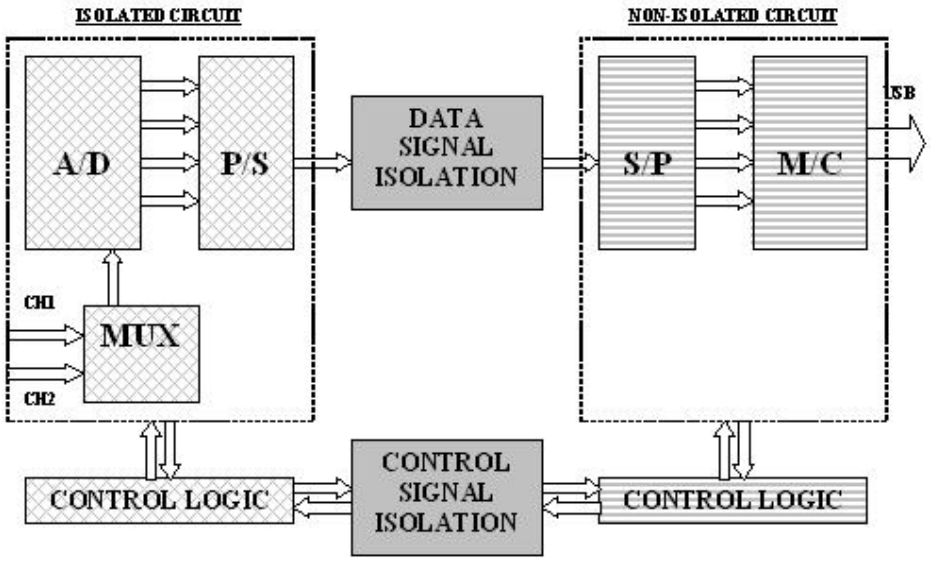



Figure 3.1: Block Diagram of the Data Acquisition Hardware

### 3.1.1 Isolated Circuit

Isolated part of data acquisition system receives its inputs from analog amplifiers. Thus, equipment used in this part have the same common with these isolated amplifiers. Multiplexing, A/D conversion and Parallel/Serial conversion (P/S) operations are done in this circuit. Figure 3.2 illustrates the design made for isolated circuit part.

$CD4052$ Dual 4-Channel Analog Multiplexer was selected for purpose. One of two input selector bits was hard-wired to signal ground. Therefore, we could control the multiplexer via just one control bit (MUX bit). This control bit selects between two analog input signals coming from ECG amplifier and respiratory signal amplifier. Output of the multiplexer is directly connected to the Analog Voltage Input (Pin 14) of the A/D converter.

$AD574AKN$ is the 12-bit A/D converter used. Its datasheet is given in Appendices. This converter provides parallel outputs in 8-bit or 12-bit according to 12/8 bit. For better accuracy 12-bit output was selected. Pin 1 and Pin 2 were hard-wired to $V_{LOGIC}$ whereas Pin 3 and Pin 4 to ground. As a result, 12-bit conversion operation is initiated when Read/Convert ($R/\bar{C}$) input bit (Pin

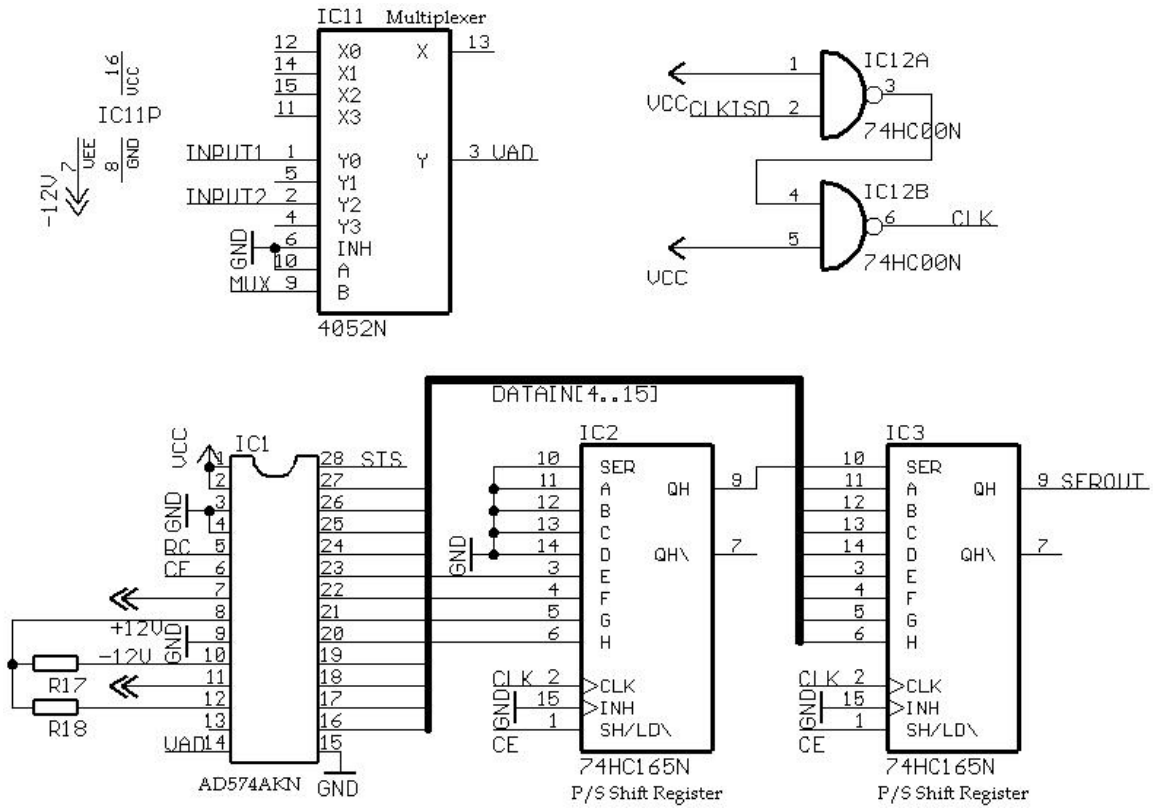



Figure 3.2: Isolated Circuit of the Data Acquisition System

5) is logic $'0'$ and Chip Enable ($CE$) bit (Pin 6) is logic $'1'$. When conversion is complete, Status(STS) output (Pin 28) of A/D converter goes low. To read converted data, output buffers (Pins $16-27$) should be enabled by $R/\bar{C} =' 1'$ and $CE =' 1'$. A/D converter was set to bipolar operation mode which quantizes $\pm 10V$ input range with 12-bit resolution.

A pair of $74HC165$ 8-bit Parallel-To-Serial (P/S) Shift Registers were chosen for transmitting the 12-bit parallel output of A/D converter by a serial line. So, the isolation of signal could be done on just one data transmission line instead of using 12 parallel data isolation circuits. 4 bits of trailing $'0's$ were added to inputs of the registers, expanding data signal to 2 bytes (16 bits). Parallel Load (Pin 1) inputs of the registers have also been connected to $CE$ control signal. When $CE =' 0'$ parallel input data are loaded to registers, and they are shifted on each rising clock edge of $CE =' 1'$ condition.

Isolated power supply is designed to give DC voltage outputs of $\pm 12V$ and $+5V$. Power supply is shown in figure 3.3. For $\pm 12V$, half-wave rectifiers are used to clip AC signal. However, for $+5V$ regulation, full-wave bridge rectifier is used since power consumption on this output is larger. Charging capacitors are loaded with a regulator to form regulated DC outputs. 7805, 7812 and 7912 were selected for voltage regulation.

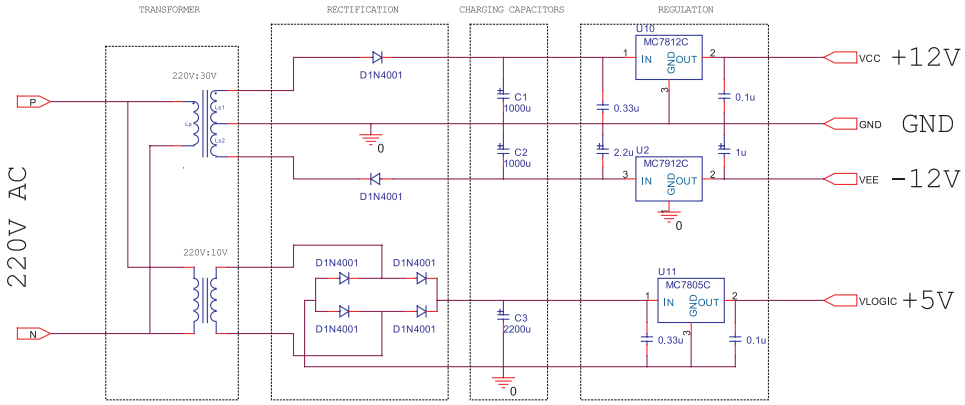



Figure 3.3: Isolated Power Supply

## 3.1.2 Isolation

Isolation of the patient connection from the earth ground was accomplished by two different techniques. First one is isolating the power supply common from earth ground and the second one is isolating both data and control signals of the system. Power supply uses magnetic isolation, whereas data and control signals are optically isolated. Therefore, there exists no electrical connection between isolated and non-isolated sides. These isolation circuits are described below:

### 3.1.2.1 Power Supply Isolation

Power supply common has been isolated from the earth ground by using a transformer which has primary and secondary windings separated from each other by a distance of $1.5 - 2mm$.

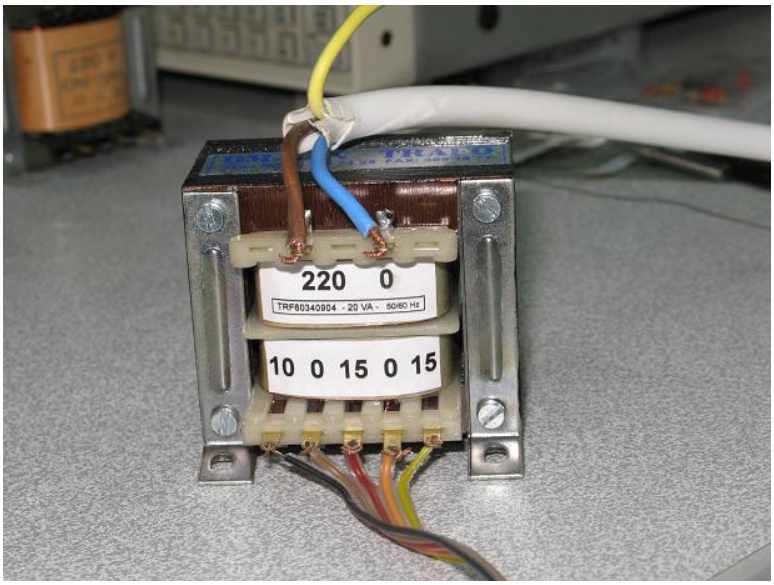



Figure 3.4: Power Supply Isolation by the Transformer

As can be seen from figure 3.4, transformer used for isolation transforms $220VAC$ down to $\pm 15VAC$ and $+10VAC$ at the secondary winding.

Transformer was tested under $220VAC$ voltage input, by measurement of leakage current between windings. Test results are covered in the Device Compliance Tests part of this thesis. Results are compliant to AAMI standarts.

#### 3.1.2.2 Signal Isolation

Optocoupler is the combination of a LED and a phototransistor. There is only optical connection between its input and output. Signal isolation circuit was designed using $HCPL-2531$ Dual Hi-Speed Optocoupler chips. These chips can withstand up to $2500V_{RMS}$ and have insulation resistance of $10^{12}\Omega$.

An optical isolator circuit using optocoupler is shown in figure 3.5. The input diode current is transferred to form a collector current with a specified current transfer ratio ($CTR$). Load resistance ($R_L$) is important for switching propagation delay times, when high frequency performance is expected. In case of

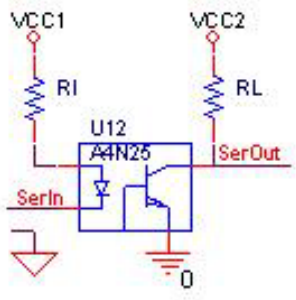



Figure 3.5: Optical Signal Isolation by an Optocoupler

digital signal transmission, linear characteristics are not necessary for an isolator. Therefore, these nonlinear optocoupler chips do not cause drawbacks to the design. Grounds of the input and the output are insulated from each other.

When the input is logic $'1'$, diode is off and no collector current flows through the load resistor. Therefore, output voltage on collector pin is high. However, a logic $'0'$ input opens the diode and an input current flows. Magnitude of this current is adjusted by the resistor ($RI$) on the input. Current on the collector flows through load resistor, pulling the output down to logic $'0'$.

Schematic design of the whole isolation circuit is given in figure 3.6. Totally 6 optocouplers are used in 3 chips. $2.2K\Omega$ load resistors were selected in the design whereas input resistors were $150\Omega$ at the non-isolated side and $220\Omega$ at the isolated side.

In design abbreviations, signal names ending with $'2'$ refer to non-isolated part, whereas others are the same signals after isolation. So, it can bee seen that signal isolation is in both directions. Since we use dual optical isolator chips, emitters of phototransistors on each chip are common. That's why we isolated signals flowing to the same direction in the same chip.

### 3.1.3 Non-Isolated Circuit

This circuit has its common grounded to a PC via the USB connector. The logic supply coming from USB has been regulated to $3.3V$ since USB microcontroller

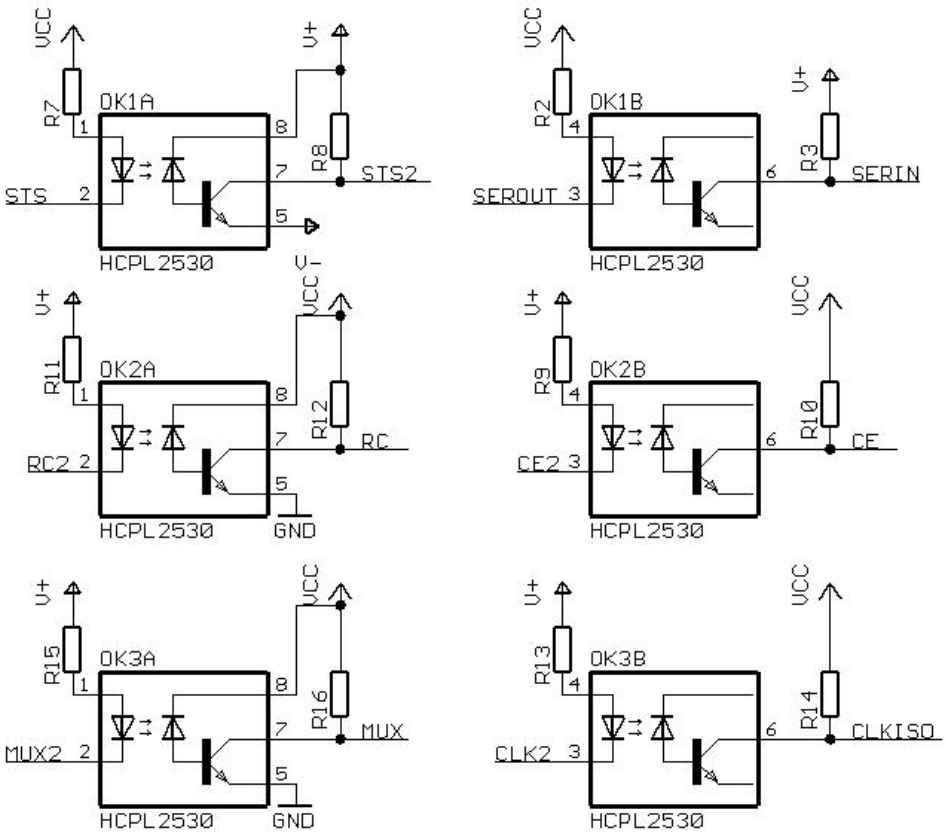



Figure 3.6: The Signal Isolation Circuit

(M/C) works with that value. All other digital chips in this part had to work properly with the $3.3V$ supply. Therefore, 74HC series chips were selected for the design. Data connections of the design can be easily seen in figure 3.7.

$74HC164$ 8-bit Serial-in/Parallel-out (S/P) Shift Registers are used to convert isolated serial data signal to 16-bit parallel data in order to read from ports of USB microcontroller. Resetting and clocking of these registers are synchronized with P/S registers in isolated part of the circuit.

PC communication protocol of the system was selected as Universal Serial Bus (USB). In this choice, high speed, easy connection, automatic settings, simplicity and flexibility features of USB were influential on us [2]. Therefore, core of the system was selected to be a USB microcontroller.

*Cypress* $AN2131QC$ 8-bit USB Microcontroller (EZ-USB) chip is the main

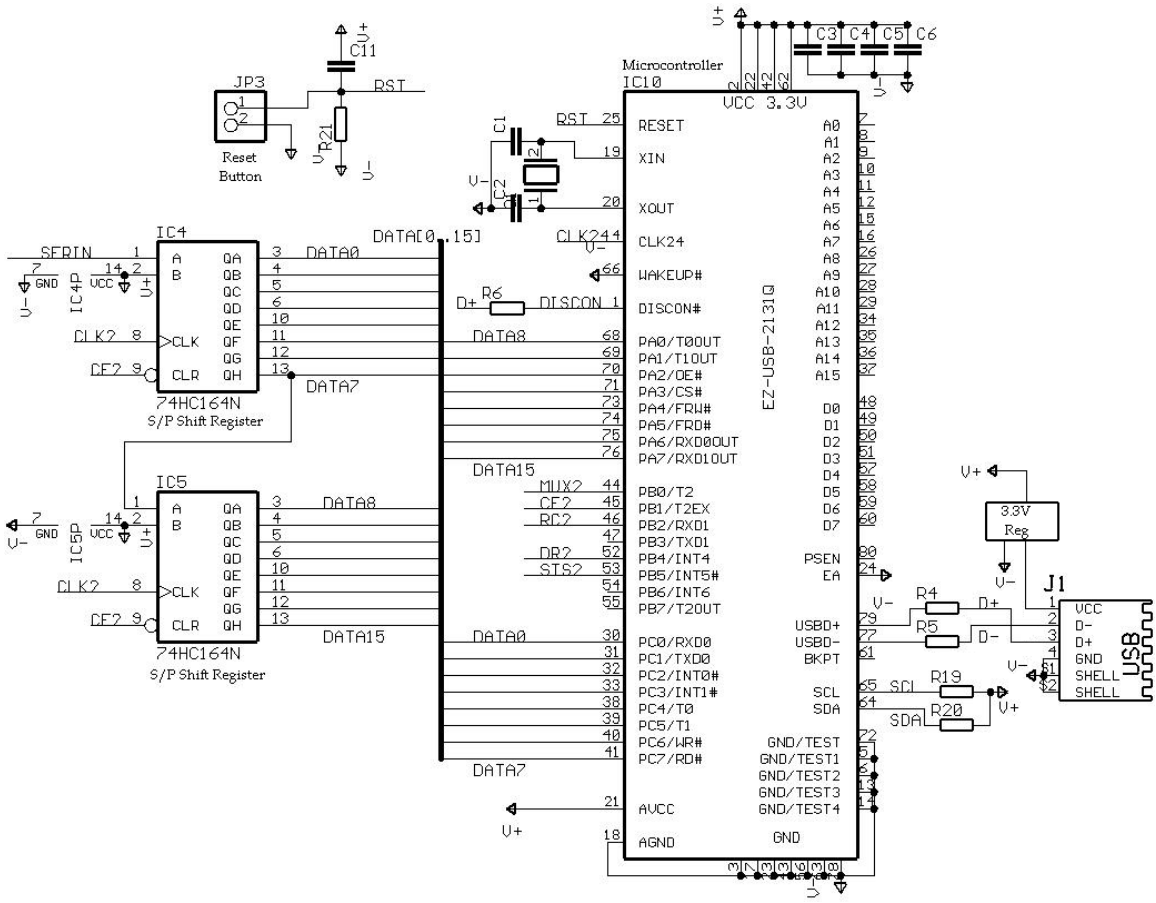



Figure 3.7: Non-Isolated Circuit of the Data Acquisition System

control unit of the data acquisition system, which organizes the timing of A/D conversion, multiplexing, serial data transmission from isolated circuit and transfer of data to computer via USB. This chip uses a standard 8051 CPU with embedded USB core.

PortA and PortC of Ez-USB were set as input data ports. PortB has been used for interrupt inputs, and control signal outputs. In table 3.1, instead of $D[15-0]$ data inputs, $STS$ and $DR$ are interrupt inputs, whereas $R/\bar{C}$, $CE$ and $MUX$ are output control signals.

All $V_{CC}$ inputs of the chip were connected to $3.3V$ DC voltage with $100nF$ decoupling capacitors. For $AVCC$ pin, there exists a $2.2\mu F$ tantalum decoupling capacitor in parallel with $100nF$. USB data pins $D+$(Pin 79) and $D-$(Pin 77) are

| | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|
| PortA | D[15] | D[14] | D[13] | D[12] | D[11] | D[10] | D[9] | D[8] |
| PortB | not used | not used | $STS$ | DR | not used | R/$\bar{C}$ | CE | MUX |
| PortC | D[7] | D[6] | D[5] | D[4] | D[3] | D[2] | D[1] | D[0] |

Table 3.1: Port Assignment of Ez-USB Chip

connected to a Type-B USB connector via 27Ω resistors ($R4$ and $R5$). DISCON (Pin 1) is connected to $D+$ line through a $1.5K\Omega$ resistor ($R6$) in convenience to Ez-USB Technical Reference.

Active high reset of 8051 and USB SIE is done through a Reset button connected to RST (Pin 25) input. When the button is pressed this pin is shorted to the ground. Normally, the pin is driven logic '1' through a $100nF$ capacitor($C11$) and a $10K\Omega$ resistor ($R21$).

Serial EEPROM is not used in this design. Therefore, SCL (Pin 65) and SDA (Pin 64) inputs have been connected to VCC via $2.2K\Omega$ resistors ($R19$ and $R20$). External memory access is also disabled by grounding the EA (Pin 24) input.

Ez-USB is driven by a $12MHz$ crystal connected between XIN (Pin19) and XOUT (Pin 20), with two parallel $30pF$ capacitors ($C1$ and $C2$). The chip provides $24MHz$ clock output from CLK24 (Pin 4) which is connected to the input of the clock control logic of the system shown in figure 3.8. This control logic consists of a $74HC590$ 8-bit Binary Counter, a $74HC163$ 4-bit Binary Counter, a $74HC74$ Dual D-Type Flip-flop and a $74HC00$ Quad-NAND chip. $74HC590$ is in free-running mode. Its QG (Pin 6) output is used as clock. Thus, it forms a square-wave clock signal of $187.5KHz$ by dividing the $24Mhz$ input frequency.

$187.5KHz$ clock signal is distributed to all sequential chips on the card when a Clock Enable ($CLKEN$) signal is in its logic high state. $CLKEN$ is low only when $CE =' 1'$ and $DR =' 1'$. $DR$ is the output of D flip-flop connected to carry bit(Pin 15) of $74HC163$ 4-bit counter. This counter is reset to $'0000'$ when $CE =' 1'$ to give output carry after 15 clock cycles. Therefore, after A/D converter buffers are enabled, this clock signal changes its state for only 16 cycles to transmit 2 bytes data serially from isolated side to non-isolated side.

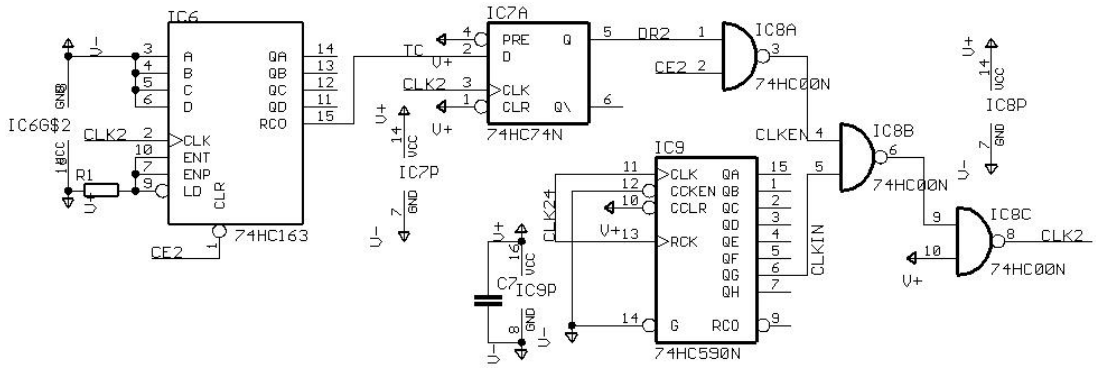



Figure 3.8: Clocking Logic

### 3.1.4 USB Interface

In a USB cable, there are only 4 lines $VCC$, $GND$, $D+$ and $D-$. $D+$ and $D-$ carry data in differential form, increasing the transfer speed. Ez-USB chip works with USB 2.0 which is the High-Speed Specification of USB allowing up to $480 Mbits/sec$. A standard USB cable is used between the data acquisition system and the PC.

### 3.1.5 Implementation of Data Acquisition Hardware with Isolation

Data acquisition system had been implemented on breadboards using Cypress Ez-USB $2131QC$ Development Kit, before PCB implementation. This kit is a useful development tool for firmware design of custom hardware. It includes a Developer Board, a USB generic driver, a control panel application software and example codes for firmware development. Detailed information can be found in the official web site of Cypress Semiconductors. After development of the design, Data Acquisition Card with Isolation was implemented on PCB. Layout was designed by *Cadsoft Eagle* 4.09 PCB design software. Figure 3.9 shows this implementation. Isolated and non-isolated circuit elements were placed distant from each other to meet biomedical safety standards as shown in figure 3.10.

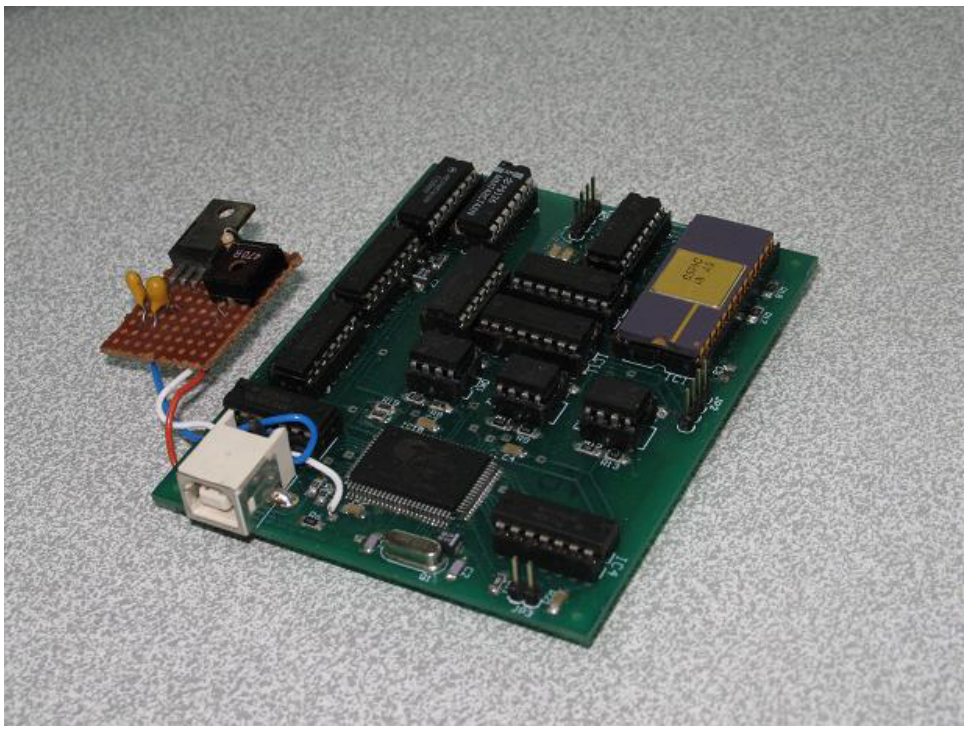


Figure 3.9: Implementation of the Data Acquisition Card with Isolation

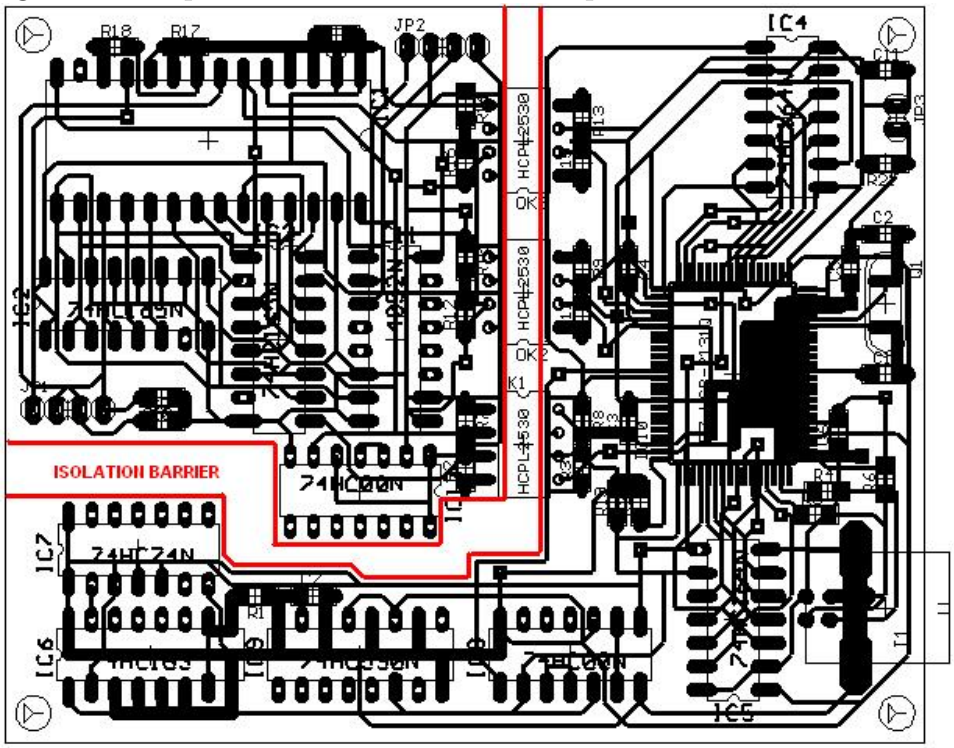



Figure 3.10: Isolation Barrier in the Data Acquisition Card Layout

## 3.2 Software Design Materials and Methods

### 3.2.1 Microcontroller Software

The operation of Ez-USB is controlled by an A51 assembly code loaded to RAM of chip. Code was written in *Keil µVision2 Software* platform. The code is fully given in Appendices. Registers and their special function bits are defined in "ezregs.inc" include file.

#### 3.2.1.1 Introduction to USB

USB Topology and Transfer Types

In USB topology, there is a root hub which has the control software of all the transfers between computer and its peripherals. This is the *host controller*. Apart from it, there are hubs or peripheral devices connected. Hubs can be connected to multiple devices or other hubs since devices share time on the same USB line controlled by host controller. Transfers are done between a *host* and its *device*.

If a transfer is done from host to device, this transfer is abbreviated as OUT transfer. Otherwise it is an IN transfer.

There is a device address and an *endpoint* address in all USB transactions. Host communicates with the device at given address. It accomplishes transaction by reading from or writing to the specified endpoint location on the device.

Before transfer, a *pipe* should be associated between host controller software and device. USB pipes are not physical objects.

In USB, there are 4 different transfer types: Control Transfers, Bulk Transfers, Isochronous Transfers and Interrupt Transfers.

Control Transfers send requests and data related to abilities of device and its

configuration [2]. These transfers are done in device enumeration and renumeration purposes, interface settings, pipe configuration etc. This transfer type uses endpoint0.

Bulk Transfers are guaranteed data accuracy transfers, due to its handshaking and re-try mechanism for erroneous data. When there is available bus time, bulk transfer tokens are sent from host to device. If USB bus is idle or no other devices are connected, bulk transfer gets faster. Packet of size 8, 16, 32 or 64 bytes travel in this transfer. Its typical use is in printers, scanners etc.

Interrupt transfers are like bulk transfers, but it is only defined for IN transactions. Host controller regularly pings interrupt endpoints of the device after a specified polling interval. Mouse and keyboard use interrupt transfers.

Isochronous transfers are time-critical transfers. In every USB frame, there is an allocated bandwidth for isochronous data. But there can be erroneous data. Isochronous transfers are used in time-critical video streaming and similar purposes.

Ez-USB Bulk Transfers

Our data acquisition system requires both data accuracy and minimum delivery time. Bulk transfer method is selected for purpose due to its guaranteed data accuracy with no other devices connected to USB root hub in order to minimize time-sharing of the serial USB line.

Bulk transfer is used for both transfer of digitized and isolated data to PC and user application requests from PC to device such as sampling frequency selection, A/D conversion initiation etc.

Ez-USB Bulk IN transfers are proceeded by first locating data to be transferred into corresponding IN endpoint buffers (i.e. IN2BUF). When the number of data bytes is loaded into the Byte Count Register of the endpoint (i.e. IN2BC), bulk transfer is armed. In case of an incoming IN token from the host, device sends data in endpoint buffers by handshaking.

Ez-USB Bulk OUT transfers are started after an incoming OUT token from the host controller. Device accepts and writes data transferred into its corresponding OUT endpoint buffers (i.e. OUT6BUF).

Status of bulk transfers can be followed from Control and Status Registers (i.e. OUT4CS).

The microcontroller software design procedure can be divided into three main parts: Initialization, Main Loop and Interrupt Services.

#### 3.2.1.2 Ez-USB Initialization

Initialization code configures ports and enable USB interrupts in order to make microcontroller ready for communication with user application.

EZ-USB chip has 3 ports, bits of which can be set as input or output (I/O) data bits or special function bits. If bits of port configuration registers (PORTACFG, PORTBCFG or PORTCCFG) are set as $'1'$, corresponding pins are used with their special functions described in Ez-USB Technical Reference, otherwise they are set as data I/O pins.

Direction of the pins are assigned by Output Enable registers (OEA, OEB or OEC). If its $OE$ bit is set logic '1', a pin is configured as an output. Data written to output registers (OUTA, OUTB or OUTC) are seen on output pins. If $OE =' 0'$, corresponding pin is a high resistance input. The logic state seen on an input pin is written to pin registers (PINSA, PINSB or PINSC).

According to table 3.1, all Ez-USB port pins were set as standard I/O or special function pins with their I/O directions.

PortB[0] ($MUX$) is initialized as logic $'0'$, in order to start A/D conversion always from first analog data channel. PortB[1] ($CE$) and PortB[2] ($R/\bar{C}$) are initalized as logic $'0'$ to be sure that A/D converter is ready for conversion.

Registers $R1 - R6$ are used for different purposes. $R1$ holds sampling period

information to be loaded to timer counter. $R2$ is the data pointer showing next buffer location where data would be written into. $R3$, $R4$ and $R5$ are used as temporary buffers while reading data from ports. $R6$ is the pointer showing the endpoint to be filled before data IN transfer. Register $R1$ is initialized for $1KHz$ sampling of each channel. Other registers are initialized to decimal zero.

Different USB transactions create $INT2$ interrupt. Therefore, it has an auto-vectoring structure, starting from a base address. Each USB interrupt source is auto-vectored and jumped to a distinctive interrupt service routine (ISR). Auto-vectoring is enabled by setting USBBAV[0] (AVEN bit). EIE[0] is the interrupt enable mask for $INT2$. These bits are set at initialization. OUT6 and OUT4 endpoint interrupts are enabled to be used in PC communications.

Global interrupt enable bit is IE[7]. When this bit is not set, none of the sources on the chip create an interrupt.

#### 3.2.1.3 Ez-USB Main Loop

After the initialization, program enters into an infinite loop. This loop checks register $R2$ whether 64 bytes of data has been written to IN buffer or not. If true, it changes the endpoint buffer to be filled and arms IN transfer of 64-bytes data from full endpoint buffer. Otherwise, it waits buffer to be filled.

#### 3.2.1.4 Ez-USB Interrupts

INT2, INT4, $\overline{INT5}$, Timer0 and Timer1 interrupts were used in design procedure.

$Timer1$ creates an interrupt when its interrupt service is enabled and timer flag, TF1, is set. Interrupt service enters into *first_data* ISR. TF1 is automatically set when all bits of timer counter reach to $'1'$. IE[3] should be set before to have interrupt service when timer flag is set.

Starting from an initial value, timer counts up every 12 clock cycle of Ez-USB

chip. Since Ez-USB clock frequency is 24MHz, the counting frequency of timer is 2Mhz. The initial value of timer1 is determined by TH1 (first 8 bits of timer) and least significant five bits of TL1 registers. TL1 register is set as zero. Thus,

$$T_s = \frac{1}{f_t}[2^{13} - 2^5(TH - 1)] \tag{3.1}$$

where $T_s$ is sampling period, $f_t$ is timer frequency and TH is initial value of TH1 register before counting.

From equation 3.1,

$$TH = 256 - \frac{T_s f_t}{32} + 1 \tag{3.2}$$

Calculating initial TH1 values from equation 3.2 with $f_t = 2$MHz, TH values are found as decimal 7 for 250Hz, 132 for 500Hz, 195 for 1KHz and 226 for 2KHz sampling frequencies.

When TR1 bit is set high, counting starts. Timer flag, TF1, should also be cleared before the end of counting.

$Timer1$ determines the sampling frequency of A/D conversion. This timer counts for one sampling period. When interrupt service is handled, a short pulse is given to CE pin for initiating the A/D conversion of first analog data channel.

$Timer0$ interrupt enters into *second_data* ISR. It was set for converting the second analog channel, immediately after reading converted first channel data. Therefore, a short and fixed delay occurs between sampling of different channels. Timer0 operates at the same procedure of timer1. Counter initialization is done through TH0 and TL0 registers. TR0 is the start of counting bit, whereas TF0 is the flag which rises when counting ends. IE[1] is interrupt enable bit of this timer.

$Timer0$ counter was fixed to wait for **200** $\mu$sec, whereas sampling period of channels can be loaded from computer to TH1 and TL1 registers of Ez-USB timer1.

$\overline{INT5}$ interrupt is controlled by PortB[5] pin, which jumps the program to *Int5Isr* interrupt service at the falling edge of $STS$ signal. This signal falls when

A/D conversion is completed. EIE[3] bit is the interrupt enable mask of $\overline{INT5}$. $R/\bar{C}$ is set high and $CE$ is given a short pulse to activate output latches of A/D converter and load shift registers. Rapidly $CE$ is set high again to start shifting of 16-bits converted data to non-isolated side of circuit.

$INT4$ interrupt jumps program into *Int4Isr*. Its control is from pin PortB[4], which starts the interrupt service when $DR$ is at its rising edge. $DR =' 1'$ after all 16 data bits have been serially transmitted to non-isolated side and ready to be read from I/O ports. This interrupt is enabled by setting EIE[2].

$INT4$ Interrupt service firstly complements multiplexer control bit, to change analog data channel to be converted next. Then it reads PortA and PortC into registers $R4$ and $R5$. $CE$ and $R/\bar{C}$ are cleared in order to disable A/D converter and make it ready for next conversion. According to data pointer $R2$, 2-bytes data are written into their corresponding location at the IN endpoint buffers and $R2$ is increased by two.

$INT2$ interrupts are used for two sources: Endpoint4 OUT and Endpoint6 OUT tokens. *Isr_Ep4Out* writes data sent from computer into register $R1$ to save new sampling rate information. *Isr_Ep6Out* initiates A/D conversion after host request, by setting interrupt enable masks for $Timer1$, $Timer0$ and $\overline{INT5}$ interrupts.

### 3.2.2 The User Application

The data acquisition system is controlled from the host computer by a user application. The application program was written in Microsoft Visual C++ 6.0, using Microsoft Windows Win32 API functions and Ez-USB I/O control functions. Its code is fully given in Appendices. Each function in the software will be described in this part.

#### 3.2.2.1 Functions For Initialization and Main Loop

**int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, INT iCmdShow)**

WinMain function is the main function running in the application. This function calls initialization functions to create a window and append a menu bar on it. Then it continuously waits for user messages sent to the window.

**int WindowCreate (void)**

Win32 Application software is initiated by creating a window in Microsoft Windows platform. Firstly, a window class is defined and its properties are entered. This window class should be registered by Microsoft Windows. Therefore, *RegisterClass* function is used to register defined window class *wndclass*. After registration of its class, the window is created in an overlapped structure using *CreateWindow* command. The window functions are accessed through its handle *hWnd*. It is then maximized to screen size by *ShowWindow(hWnd, SW_SHOWMAXIMIZED)* and *UpdateWindow(hWnd)* commands.

**void menubar(HWND hWnd)**

This function creates a menu tree for the application and appends this tree to the window. Main menu and its sub menus are created by CreateManu command. These objects are attached to their global HMENU type handles.

Menu items are declared for all sub menus, and then these sub menus are attached to their parent menu using *InsertMenuItem* command. *Main Menu* has two branches, *File* and *Options*.

File Menu has *Download Firmware* and *Exit* commands. On the other hand, Options menu contains controls for data acquisition and display settings. Options Menu maintain *Start/Stop Reading*, *Start/Stop Recording*, *Change Sampling Rate* and *Change Full-Scale Voltage Range* alternatives. Each menu item has a unique *item wID*, to distinguish user message sent. Main menu is appended to window by *SetMenu(hWnd, hMenu)* command.

Lastly, Options Menu is selected to be the Popup Menu. Therefore, it can be accessed by right-click of mouse on the screen.

**int WaitMessages(void)**

This function gets messages are sent to window by *GetMessage(&msg, NULL, 0, 0))* command. Then, it calls *TranslateMessage(&msg)* function to convert virtual-key messages into character messages. *DispatchMessage(&msg)* function is called last to send character message into window process function *WndProc*.

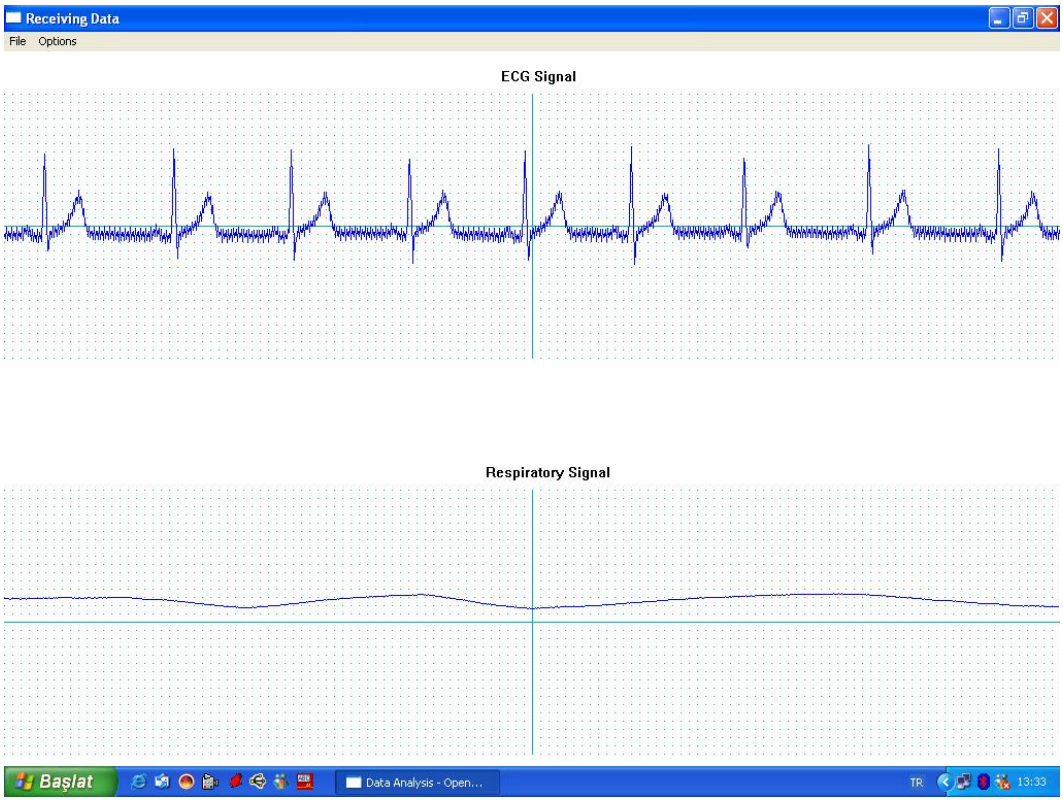



Figure 3.11: User Application Software

#### 3.2.2.2 Functions Called after User Commands

**LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)**

This is the window process function which has user defined sub-functions to be executed for different window messages. These messages can be VM_CREATE, VM_PAINT, VM_DESTROY, VM_COMMAND etc.

A VM_PAINT message is received when *UpdateWindow(hWnd)* function is called, in order to redraw the invalidated part of the client screen. A device context is created by *BeginPaint* function to prepare window for painting and fill the paint structure with information on painting.

Firstly, axes and grids for both ECG and Respiratory Signal channels are drawn. Then, for each x-pixel (1024x768 screen) on the screen, corresponding y-pixel is found by calling functions *f(x)* and *g(x)*. By drawing lines from consecutive y values, continuous signals are drawn on the client area. Figure 3.11 shows the application window screen while receiving data from the device.

VM_DESTROY message sends *PostQuitMessage(0)* command to close window.

VM_COMMAND message is sent when one of the menu items is selected. By checking the wParam parameter of the message, selected item is distinguished. This parameter should be same as wID of menu item selected. Application responses for different wID values are given in table 3.2.

VM_RBUTTONDOWN message is sent after right-click of mouse. x and y pixels of the mouse pointer are found. Using TrackPopupMenu command, Options Menu is brought to screen as popup menu at that screen coordinate.

**void BeginReadProc (void)**

This function initializes Ez-USB connection. Firstly a handle to USB device is opened by calling bOpenDriver function. This handle is used in all USB control

| wID | Command | Value | Response |
|---|---|---|---|
| 'd' | Download Firmware | - | anchordownload() called |
| '1' | Sampling Rate | 250Hz | ChangeSamplingRate(7) called,<br>8 samples/packet-ch displayed |
| '2' | Sampling Rate | 500Hz | ChangeSamplingRate(132) called,<br>4 samples/packet-ch displayed |
| '3' | Sampling Rate | 1KHz | ChangeSamplingRate(195) called,<br>2 samples/packet-ch displayed |
| '4' | Sampling Rate | 2KHz | ChangeSamplingRate(226) called,<br>1 samples/packet-ch displayed |
| '5' | CH1 Voltage Scale | ±5V | Set voltagescale1=1 |
| '6' | CH1 Voltage Scale | ±2V | Set voltagescale1=2 |
| '7' | CH1 Voltage Scale | ±1V | Set voltagescale1=5 |
| '8' | CH1 Voltage Scale | ±500mV | Set voltagescale1=10 |
| '9' | CH1 Voltage Scale | ±100mV | Set voltagescale1=50 |
| '10' | CH2 Voltage Scale | ±5V | Set voltagescale2=1 |
| '11' | CH2 Voltage Scale | ±2V | Set voltagescale2=2 |
| '12' | CH2 Voltage Scale | ±1V | Set voltagescale2=5 |
| '13' | CH2 Voltage Scale | ±500mV | Set voltagescale2=10 |
| '14' | CH2 Voltage Scale | ±100mV | Set voltagescale2=50 |
| 'r' | Start/Stop Reading | Start | StopRead flag is FALSE<br>BeginReadProc() initializes connection,<br>WriteOutProc(16) enables Ez-USB,<br>A suspended reading thread created,<br>Thread priority set 'high' and resumed |
| | | Stop | StopRead flag is TRUE,<br>Call EndReadProc() closes connection |
| 's' | Start/Stop Recording | Start | Rec flag is TRUE,<br>Recording buffers are allocated,<br>fopen creates output files,<br>Recorded data counter is '0' |
| | | Stop | Rec flag is FALSE,<br>fwrite writes buffers to files,<br>fclose close output files,<br>Buffers are freed |
| 'x' | Exit | - | Window is quitted |

Table 3.2: Application Response to Messages Sent by User Commands

functions.

For communication with Ez-USB, DeviceIoControl command is used. DeviceIoControl input parameters are handle, corresponding IO control code (IOCTL), and buffers used in the communication process with their sizes. Ez-USB Interface '0' with Alternate Setting '1' is loaded to device. Control code for interface setting is IOCTL_Ezusb_SETINTERFACE.

Lastly, 64-byte buffers are allocated for Bulk Data Transfer in reading process and the function returns.

**void WriteOutProc (BYTE command)**

To enable Ez-USB timer interrupts, command= $0x16$ is sent from the host computer. This OUT transfer is done on pipe 6. Firstly, pipe is reset using DeviceIoControl with IOCTL_Ezusb_RESETPIPE IO control code. A 1-byte size buffer is allocated to hold command input. A Bulk Transfer Control Structure is created and pipe information is entered to it. DeviceIoControl is called for Ez-USB Bulk OUT Transfer. IOCTL_EZUSB_BULK_WRITE is the control code for this transfer. Before returning, function calls DeviceIoControl once more to abort active request tokens on the pipe. IOCTL_Ezusb_ABORTPIPE control code is entered as function input. There can be more commands sent to Ez-USB through this function. Ez-USB can compare incoming byte and respond.

**ULONG _stdcall READ_ISO_BUFFER (LPVOID lpParameter)**

The reading thread is this _stdcall structure. Two pipes are used in reading data process, pipe 1 and pipe3. While StopRead flag is FALSE, a reading operation is looped. After every packet, pipe used is toggled and reset.

DeviceIoControl with IOCTL_EZUSB_BULK_READ control code reads from IN endpoints of Ez-USB into the allocated 64-byte buffer isobuf. This buffer is decomposed into two temporary data buffers of 2-byte union elements with size 1024. These buffers are for CH1 and CH2 data displayed on the screen. After reception of every packet, these buffers are shifted by the number of displayed samples/packet at selected sampling frequency. New data are written to emptied

locations. If record flag is set, without discarding any data, isobuf is directly decomposed and 2-bytes data samples are located into channel record buffers. The screen is updated by invalidating the specified rectangle on the client window, and calling UpdateWindow(hWnd) function. This call is done every 64msec for each channel.

**void EndReadProc(void)**

Both pipes used in reading thread are aborted for waiting IN tokens and reading buffer is freed.

**void ChangeSamplingRate (BYTE rate)**

Input byte is transferred directly by Ez-USB Bulk OUT Transfer. It is sent over pipe 4. Therefore, sampling rate can be changed online while receiving data. Pipe is reset and aborted as in other transfers.

**int f(int x)**

f(x) and g(x) functions calculate y pixels for each x pixel given. While reading the display buffers of the channels, data value of '0' corresponds to $-10$V, whereas data value of 65535 corresponds to 10V analog voltage. The full-scale range is 256 vertical pixels, but full-scale voltage range can be reduced by magnifying the output by a scale factor *voltagescale1 (or 2)*.

**BOOLEAN bOpenDriver (HANDLE * phDeviceHandle, PCHAR devname)**

This function is the access of the application software to the Ez-USB driver kernel. The device handle is created like a file using CreateFile command for the device at address "\\\\.\\Ezusb-0"

# Chapter 4

# Design Simulations

Before implementation of the hardware design, some simulations were performed in Orcad PSpice simulation environment.

## 4.1 ECG Front-End Circuit Simulations

Designed front-end circuit is simulated for its frequency response characteristics. Differential gain of the ECG front-end circuit is illustrated in figure 4.1 for a $1mV$ voltage difference between $RA$ and $LA$ inputs. $58,5dB$ gain is observed in operating frequencies. The differential gain of the differential amplifier stage is $ADM_{dB} = 58,50dB - 30.10dB = 28.40dB$. Cut-off frequencies are at 0.05Hz and 105Hz in simulation.

Figure 4.2 shows the common mode gain of the circuit with $20V_{pp}$ common mode voltage at input terminals $RA$ and $LA$. It can be easily seen that, $ACM_{dB} = -67,50dB - 30.10dB = -97.40dB$ common mode rejection occurs at the output of differential amplifier stage at operating frequencies. Therefore CMRR of the whole circuit simulated is $126dB$. However, in practice these values cannot be achieved due to unbalanced elements in circuit.

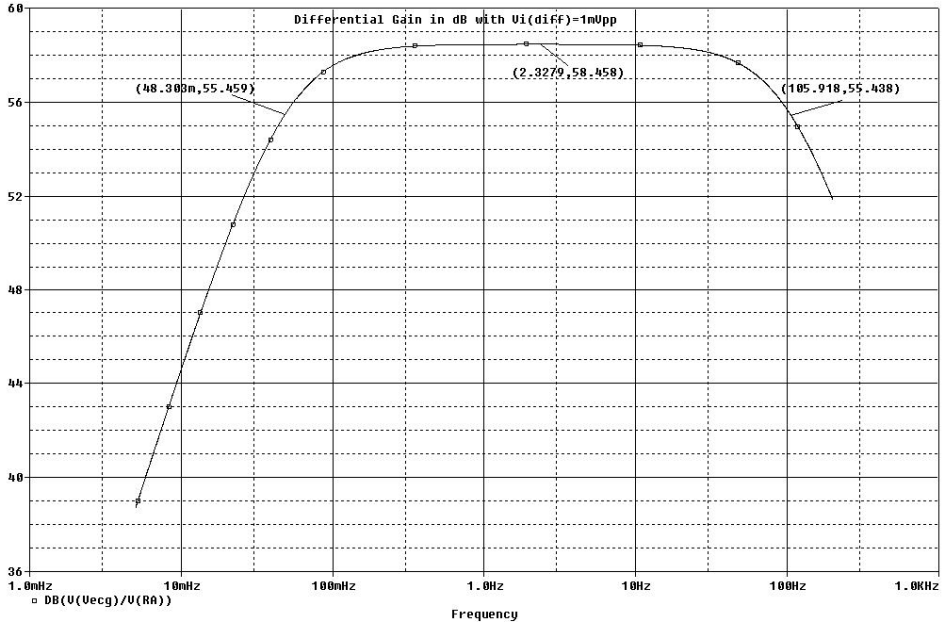



Figure 4.1: Differential Gain of the ECG Front-End Circuit

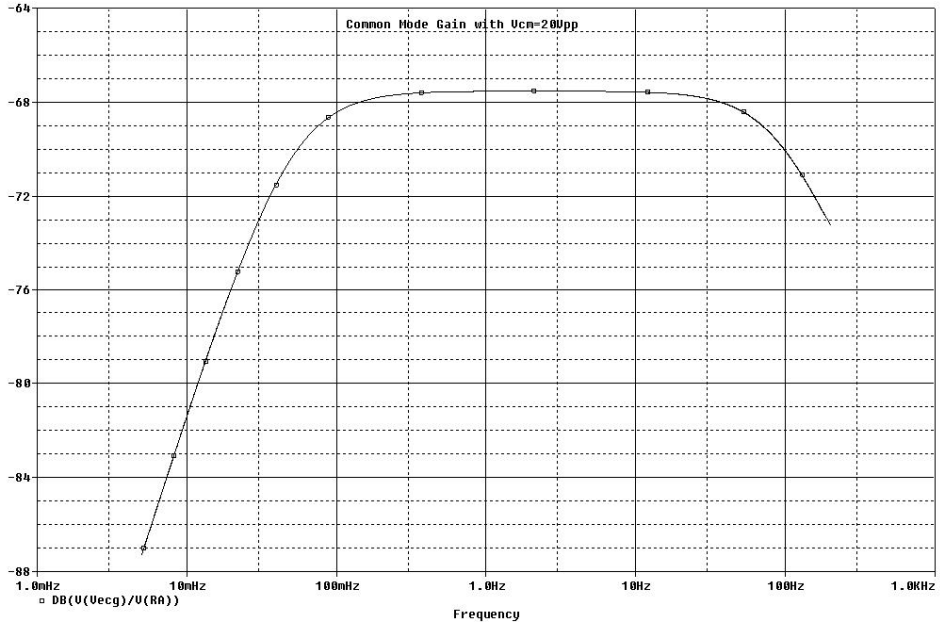



Figure 4.2: Common Mode Gain of the ECG Front-End Circuit

## 4.2 Respiratory Signal Front-End Circuit Simulation

Same circuit in respiratory signal front-end design is simulated for its differential gain as it can be seen from figure 4.3.

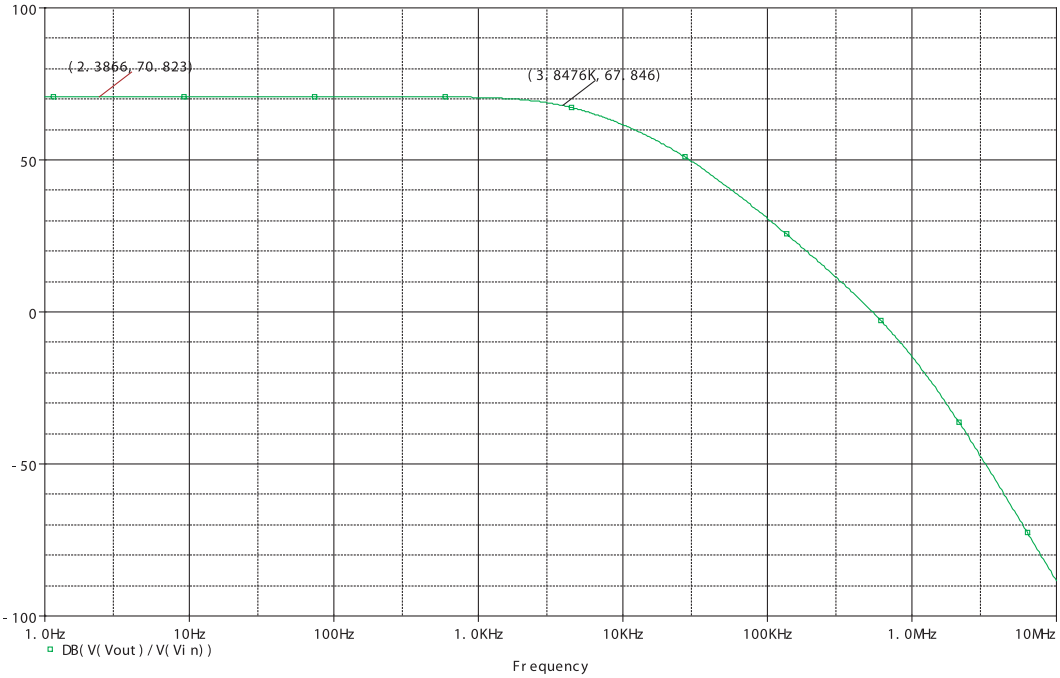



Figure 4.3: Differential Gain of the Respiratory Front-End Circuit

## 4.3 Data Acquisition System Registers Timing Simulation

The operation of the P/S and S/P registers are simulated using control signals and clocking. Test circuit is shown in figure 4.4. Since, PSpice models of AD574 and Cpress AN2131QC chips do not exist, they are not involved in the simulation. Also optocoupler operation of serial data signal is tested in this simulation. CE and CLKOUT signals are also isolated through optocoupler in design, although simulation circuit does not involve. VCC and GND signals are common in PSpice so simulation circuit is not isolated as in design.

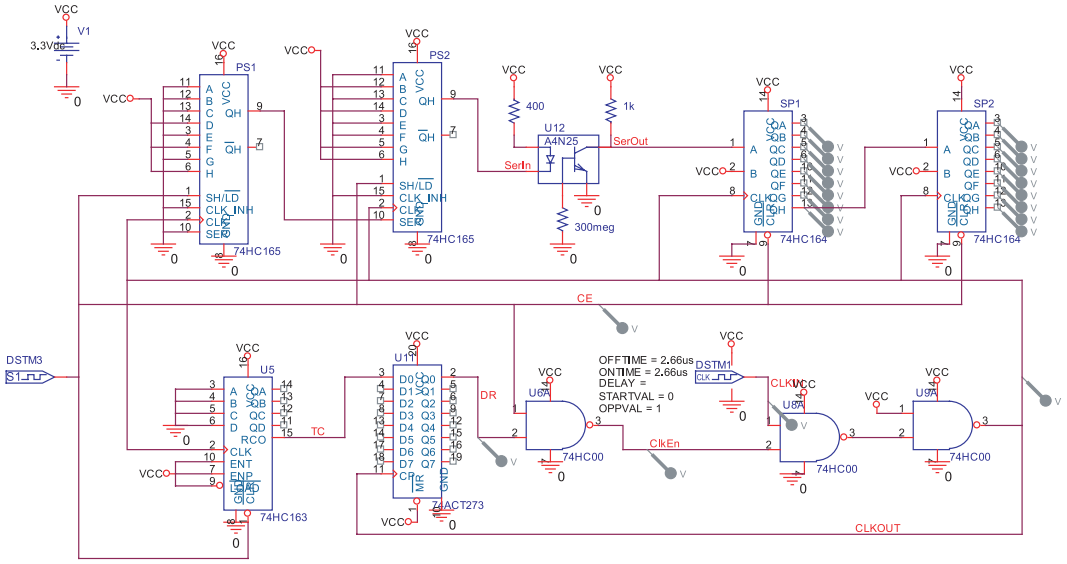



Figure 4.4: Shift Registers Timing Simulation Circuit

Simulation is displayed in figure 4.5. The inputs of P/S registers were held constant at binary "1100101010101000". During simulation, CE signal rises to initiate serial transmission. After 16 clock cycles, CLKEN signal goes low to hold output latches of the S/P registers constant. When $CE = 0$, clock is enabled again. Read operation is done at that interval in design before $CE$ goes low.

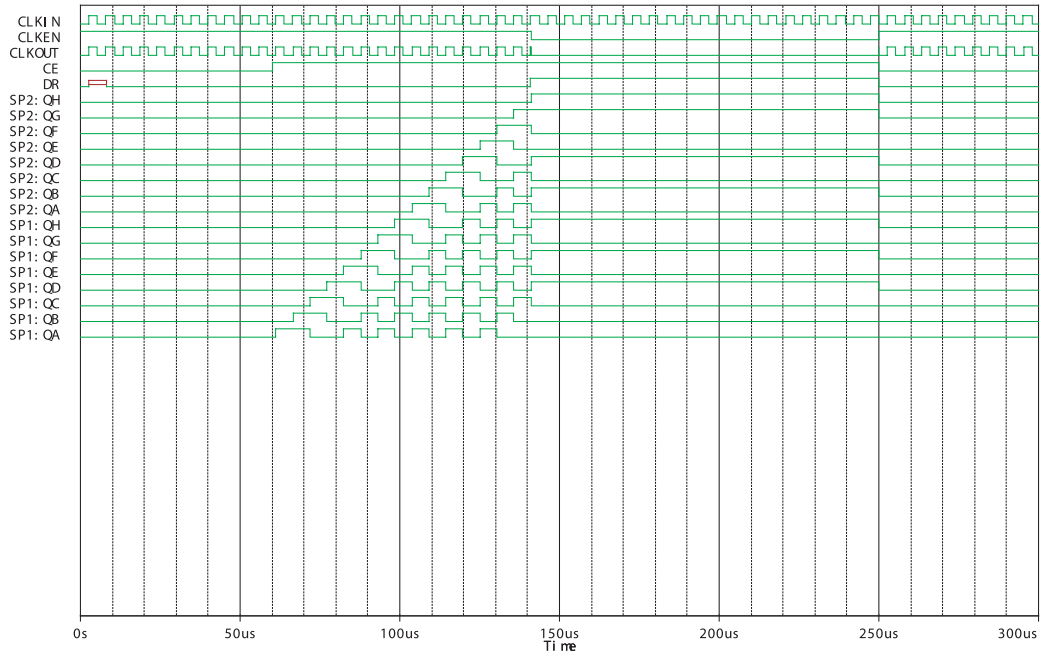



Figure 4.5: Shift Registers Timing Simulation Results

## 4.4 Power Supply Simulation

The operation of power supply was simulated before implementation with 50Ω load resistances (see figure 4.6). Instead of transformer, three sinusoidal voltage sources with magnitudes $15V_{RMS}$,$15V_{RMS}$ and $10V_{RMS}$ were used in simulation. Loading currents of the device are less than the ones in simulation. Therefore, this power supply works properly with the device.

Figure 4.7 shows simulation results for power supply circuit designed. $\pm 12VDC$ and $5VDC$ regulated outputs are supplied by the circuit.

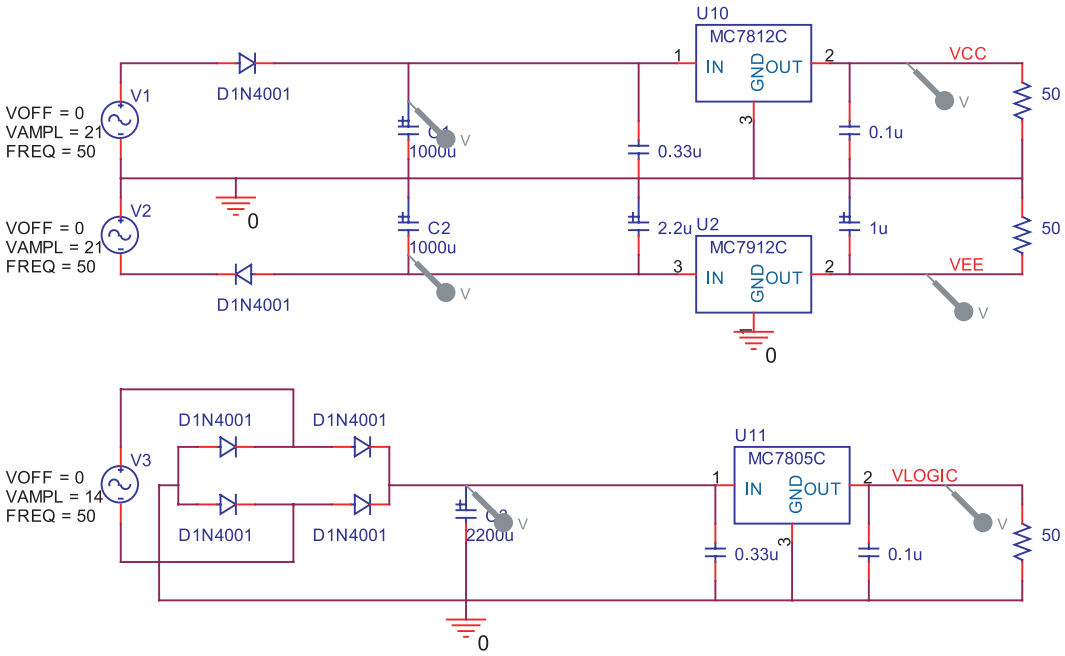



Figure 4.6: Power Supply Simulation Circuit

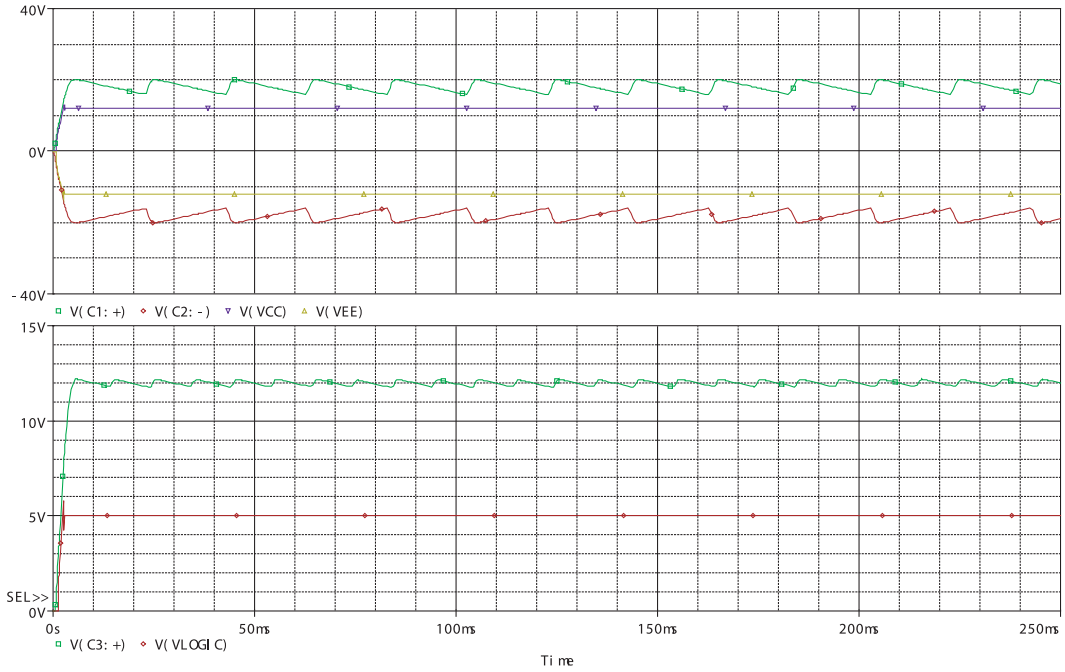



Figure 4.7: Power Supply Simulation Results

# Chapter 5

# Device Compliance Tests

The system has been tested for compliance with ANSI AAMI standards. These tests are explained in this chapter.

## 5.1 Power Supply Isolation Test

Since power supply isolation is done through the transformer, it was tested for its isolation resistance at $220V_{RMS}$ line voltage. Test circuit is given in figure 5.1.

A $1K\Omega$ resistor was connected in series with secondary winding of transformer.

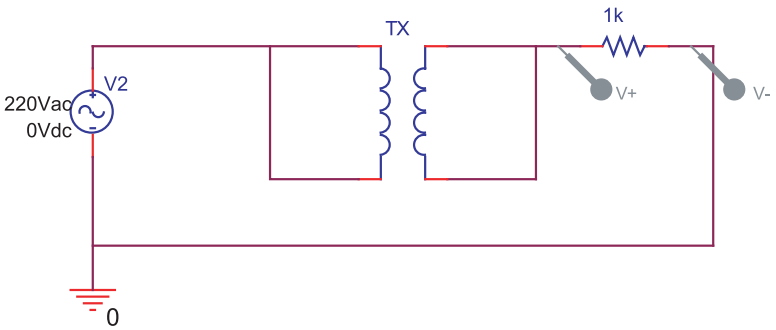



Figure 5.1: Power Supply Isolation Test Circuit

Primary winding was at $220V_{RMS}$ line voltage. The voltage read on series resistor is

$$V_{RMS} = I_{RMS}R \tag{5.1}$$

So leakage current I is

$$I_{RMS} = V_{RMS}/1000 \tag{5.2}$$

Voltage on $1K\Omega$ resistor was measured as $19mV_{RMS}$. Therefore, from equation 5.2, leakage current of transformer at line voltage was calculated as

$$I_{RMS} \quad = \quad 19\mu A$$

## 5.2 ECG Front-End Circuit Performance Test

### 5.2.1 CMRR of the Front-End Circuit

The common mode gain of the circuit was tested with $20V_{pp}$ connected to both $RA$ and $LA$ input terminals. Voltage on the output of front-end circuit was observed on oscilloscope screen. Measured output voltage is

$$V_{OUT} = 80mV_{pp} \tag{5.3}$$

Thus common mode gain measured at the output of the ECG front-end circuit is

$$ACM_{dB,overall} = 20log(\frac{V_{OUT}}{V_{IN}}) = -47.96dB \tag{5.4}$$

Since the last stage of the circuit has a differential gain of $30.10dB$, common mode rejection at the output of the differential amplifier stage is

$$ACM_{dB} = -47.96dB - 30.10dB = -78.06dB \tag{5.5}$$

Differential gain of the circuit was tested with $RA = 2.6mV_{pp}$ and $LA = 0V$. Front-end output voltage measured on oscilloscope is

$$V_{OUT} = 2.20V_{pp} \tag{5.6}$$

Therefore, differential gain of the circuit is

$$ADM_{dB,overall} = 20log(\frac{V_{OUT}}{V_{IN}}) = 58.55dB \tag{5.7}$$

When referred to the output of the differential amplifier stage

$$ADM_{dB} = 58.55dB - 30.10dB = 28.45dB \tag{5.8}$$

The CMRR of the differential amplifier stage is,

$$CMRR_{dB} = ADM_{dB} - ACM_{dB} = 28.45dB - (-78.06dB) = 106.51dB \tag{5.9}$$

### 5.2.2 Cut-off Frequencies of the Front-End Circuit

Cut-off frequencies of the circuit were found by using an input of $RA = 2.6mV_{pp}$ and $LA = 0V$ at different frequencies. Maximum front-end circuit output voltage measured on the oscilloscope is

$$V_{OUT,max} = 2.20V_{pp}$$

Output voltage decreased to

$$V_{OUT} = \frac{1}{\sqrt{2}}V_{OUT,max} = 1.56V_{pp}$$

when

$$f_{c,1} = 0.05Hz \tag{5.10}$$

$$f_{c,2} = 110Hz \tag{5.11}$$

which are the $3dB$ cut-off frequencies. These measured cut-off frequencies are very close to the simulation results.

## 5.3 AAMI Compliance Test of the CMR of the Device

A $20V_{pp}$ common mode signal was connected between the patient connections and the earth ground. The recorded output voltage of the system is

$$V = 80mV_{pp} \tag{5.12}$$

When we refer this voltage to the patient connection inputs

$$V = \frac{80mV_{pp}}{820} = 97.5\mu V_{pp} \tag{5.13}$$

AAMI standard for this voltage is $1mV_{pp}$ when referred to the patient connections. So, the system is compliant to performance standards of AAMI [5].

## 5.4 AAMI Compliance Test of the Safety Risk Currents

The device was tested for its compliance with AAMI safe current limits, by following the test instructions in the standard.

### 5.4.1 The Patient Source Risk Current Tests

The test circuit is given in figure 5.2. The current is measured from the patient connections to the earth ground, from device enclosure to the earth ground, or between the patient connections when

- the reversing switch is reversed or not
- the device power is on or off
- the ground is open or intact

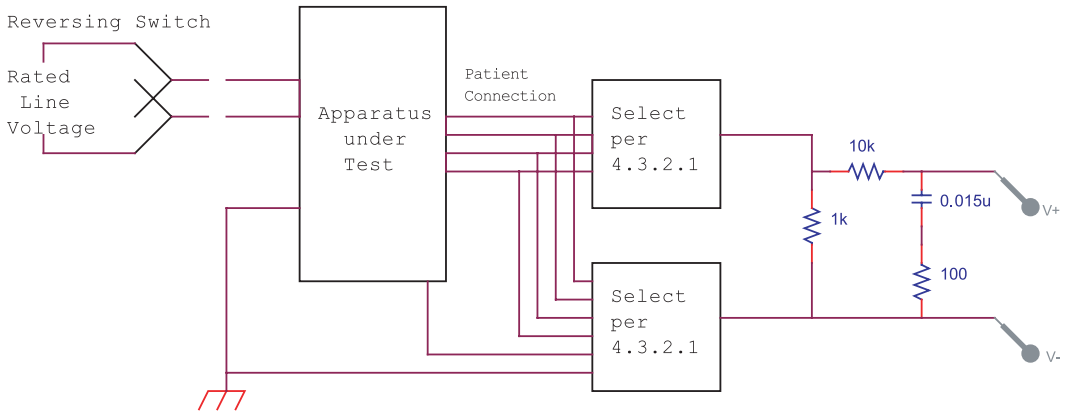



Figure 5.2: Patient Source Risk Currents Test Circuit

| | ON | OFF | S Rev (ON) | S Rev (OFF) |
|---|---|---|---|---|
| LA/Earth | 1.67 | 1.09 | 1.54 | 3.30 |
| RA/Earth | 1.56 | 0.99 | 1.61 | 3.27 |
| RL/Earth | 2.00 | 1.26 | 2.42 | 3.43 |
| All Leads/Earth | 1.83 | 0.99 | 2.13 | 3.43 |
| RA/RL | 1.00 | 0.60 | 1.08 | 2.56 |
| RA/LA | 0.78 | 0.60 | 0.72 | 1.02 |
| LA/RL | 1.10 | 0.64 | 1.09 | 2.05 |

Table 5.1: Measured Patient Source Currents ($\mu A$)

Ground is intact with the device enclosure, in order to reduce $50Hz$ interference. Therefore, ground is not switched to open position. Patient source currents are found by the values read on the millivoltmeter from equation 5.1 where $|R| = 1000$ at $50Hz$. Measurement results are shown in table 5.1. The risk current limit is $10\mu A$ for patient source currents. These results are compliant with the standard.

### 5.4.2 The Patient Sink Risk Current Tests

For an isolated patient-connected apparatus, the patient sink risk current limit is $50\mu A$ for single fault situations with $220VAC$ input from the patient ends of the connection cables. The test circuit is given in figure 5.3.

Measured patient sink currents are given in table 5.2 when device power ON.

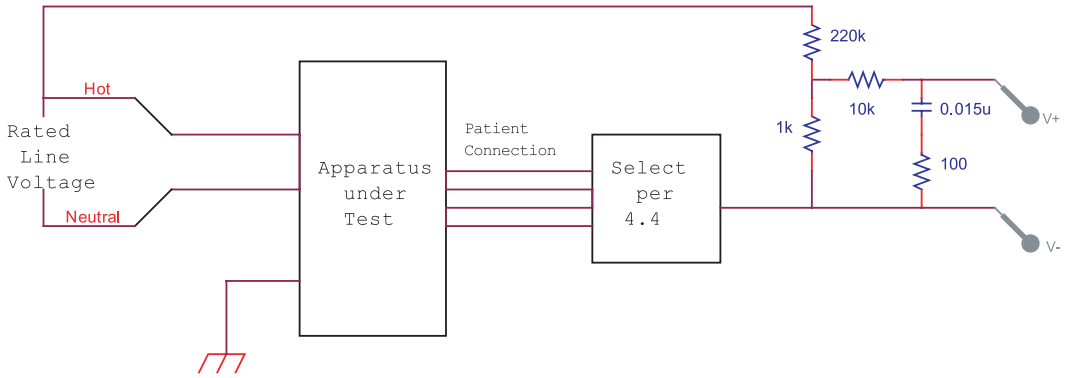



Figure 5.3: Patient Sink Risk Currents Test Circuit

| | RA | LA | RL |
|---|---|---|---|
| Power ON | 17 | 17 | 19 |

Table 5.2: Measured Patient Sink Currents ($\mu A$)

### 5.4.3 The Chassis Source Risk Current Test

This test was applied between device enclosure and earth ground. The limit for chassis source current is $100\mu A$ for isolated patient-connected apparatus.

The measured current value is $2.03\mu A$ for both ON and OFF states of the device power.

# Chapter 6

# Digital Signal Processing Software

Data saved by acquisition system is processed by digital signal processing methods using MATLAB 6.5 Release 13 Software tool. This signal processing includes filtering of the recorded data and analysis techniques for research purposes.

## 6.1 ECG Signal Processing and Heart Rate Variability Analysis

### 6.1.1 ECG Filtering

In spite of using various noise rejection methods in hardware design process, ECG signal cannot be recorded without interference. Thus, it should be filtered to suppress noise frequencies.

Filtering process is done in time domain using *Filter* command of MATLAB. Filter is a Finite Impulse Response (FIR) Filter. Filters used in design are described below:

**Lowpass Filter** Lowpass filtering is needed to suppress EMG and other high frequency signal interferences. A Moving Average (MA) lowpass filter is designed for its linear phase response. The equation for a noncausal moving average low pass filter is

$$x[n] = \frac{1}{2N} \sum_{i=-N+1}^{N} \delta[n-i] \tag{6.1}$$

Z-Transform of signal x[n] is

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x[n] z^{-n} \\ &= \frac{1}{2N} z^{N-1} \sum_{n=-\infty}^{\infty} \sum_{k=0}^{2N-1} \delta[n-k] z^{-n} \\ &= \frac{1}{2N} z^{N-1} \sum_{k=0}^{2N-1} \sum_{n=-\infty}^{\infty} \delta[n-k] z^{-n} \\ &= \frac{1}{2N} z^{N-1} \sum_{k=0}^{2N-1} z^{-k} \\ &= \frac{1}{2N} z^{N-1} \frac{1-z^{-2N}}{1-z^{-1}} \\ &= \frac{1}{2N} z^{N-1} \frac{z^{-N}}{z^{-1/2}} \frac{z^{N}-z^{-N}}{z^{1/2}-z^{-1/2}} \end{aligned}$$

$$X(z) = \frac{1}{2N} z^{-1/2} \frac{z^{N}-z^{-N}}{z^{1/2}-z^{-1/2}} \tag{6.2}$$

Fourier Transform of a filter is found from its Z-Transform, using

$$X(j\omega) = X(z) \mid_{z=e^{j\omega}} \tag{6.3}$$

where $\omega$ is between $[0, 2\pi]$. Therefore, frequency response of the filter is

$$X(j\omega) = e^{-j\omega/2} \frac{sinc(N\omega)}{sinc(\omega/2)} \tag{6.4}$$

$$\forall \omega : N\omega = k\pi, k \in Z^{+} \rightarrow X(j\omega) = 0 \tag{6.5}$$

This lowpass filter is also used to eliminate 50Hz interference, by suppressing $50Hz$ frequency. For $f = 50$Hz signal sampled at $f_s = 1$KHz, normalized frequency is

$$w = 2\pi \frac{f}{f_s} = 2\pi \frac{50}{1000} = 0.1\pi$$

From equation 6.5, when

$$N = \frac{\pi}{\omega} = \frac{\pi}{0.1\pi} = 10$$

the frequency response sinks to zero at 50Hz.

Therefore, lowpass filter was designed with $N = 10$. Frequency response of the filter is shown in figure 6.1. The cut-off frequency is at $f_c = 22$Hz which reduces the EMG signal interference on the ECG signal.

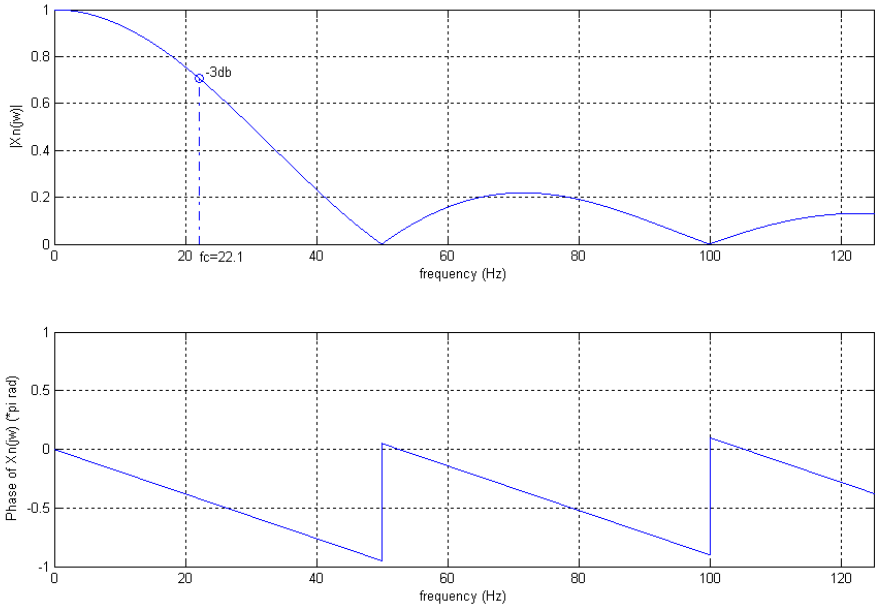



Figure 6.1: Frequency Response of the Lowpass Filter with 50Hz Notch

Time and frequency domain representations of the lowpass filter are

$$x_{lp}[n] = \frac{1}{20}\sum_{i=-9}^{10}\delta[n-i]$$

$$X_{lp}(j\omega) = e^{-j\omega/2}\frac{sinc(10\omega)}{sinc(\omega/2)}$$

**Highpass Filter** Electrode contact potential changes and other motion artifacts create very low frequency components. These frequencies should be filtered in order to have nearly same baseline in determination of QRS complexes.

Linear-phase filter design is implemented not to make original signal corrupted.

A linear phase moving average highpass filter is,

$$x_{hp}[n] = \delta[n] - \frac{1}{2N} \sum_{i=-N+1}^{N} \delta[n-i] \tag{6.6}$$

Z-Transform of signal $x_{hp}[n]$ is,

$$X_{hp}(z) = 1 - \frac{1}{2N} z^{-1/2} \frac{z^N - z^{-N}}{z^{1/2} - z^{-1/2}} \tag{6.7}$$

From equation 6.3, frequency response of the filter is

$$X_{hp}(j\omega) = 1 - e^{-j\omega/2} \frac{sinc(N\omega)}{sinc(\omega/2)} \tag{6.8}$$

A filter with the 3db cut-off frequency at $f_c = 1.5$Hz was designed. At 1KHz sampling rate $N = 500$ was selected as the filter length. Frequency response of the highpass filter is shown in figure 6.2.

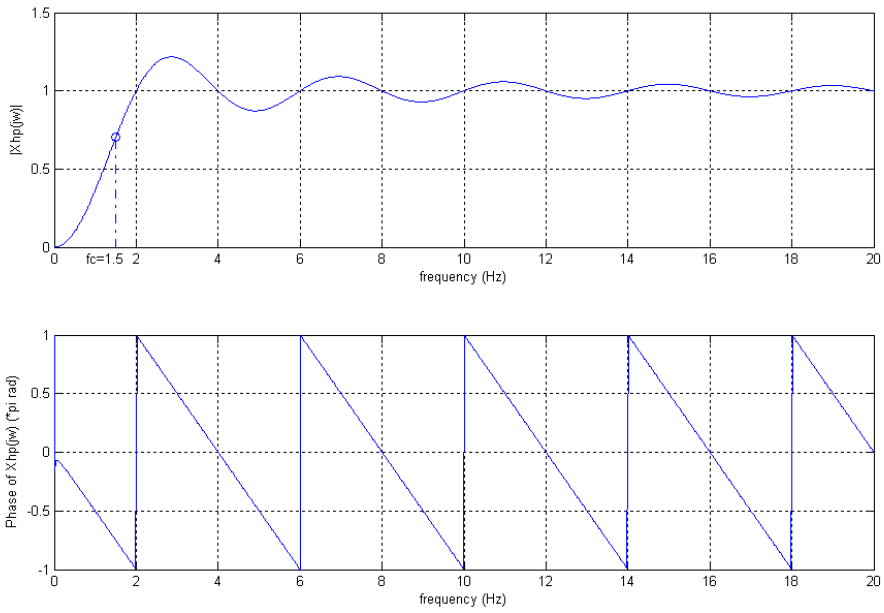



Figure 6.2: Frequency Response of the ECG Highpass Filter

These filters are applied successively on the recorded ECG data in order to have a smooth input signal for heart rate variability signal processing. In figure 6.3, application of the filters on a recorded data is shown.

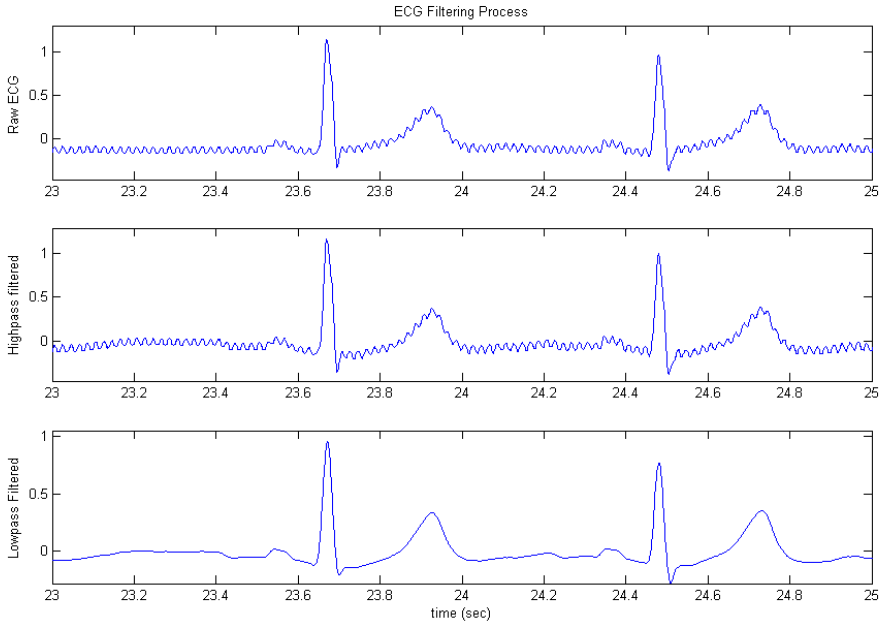



Figure 6.3: Successive Application of the ECG Filters

### 6.1.2 Heart Rate Variability Analysis

Heart rate variability is evaluated by determination of time intervals between successive R-R complexes [1]. Therefore, R-waves on each heart period should be found.

In determination of R-waves, firstly QRS complexes are found. For this purpose, $DF1$ method is used which is based on first derivative of the signal. This method was explained by Friesen et al [7].

In R-wave detection method,

- Firstly, the first derivative is found by filtering with

$$yfilt[n] = [2/10, 1/10, 0, -1/10, -2/10]$$

  where $n \geq -2$.

- Then, the maximum values of the first derivative in 3sec intervals are found.
- Median value of these maximums is found and the slope threshold is determined as 0.7 of this value.
- When 3 successive points of the first derivative exceeds slope threshold, the first point is determined as an R-wave.
- There cannot be another R-wave in the first 0.3 seconds after the determination of one.

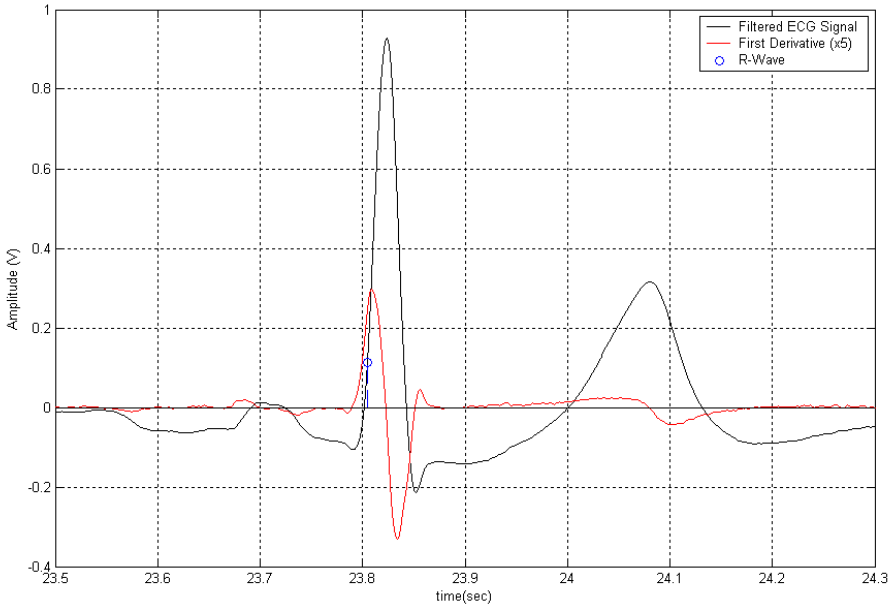



Figure 6.4: R-Wave Detection

R-wave detection using this method is illustrated in figure 6.4.

After determination of R-waves, heart rate variability signal is formed [3]. As shown in figure 6.5, time intervals between successive R-waves are found as event

series. Then, interval function which gives R-R intervals as a function of time is formed. In order to convert this discrete-time signal into uniformly sampled form, it is interpolated at $4Hz$ sampling rate.

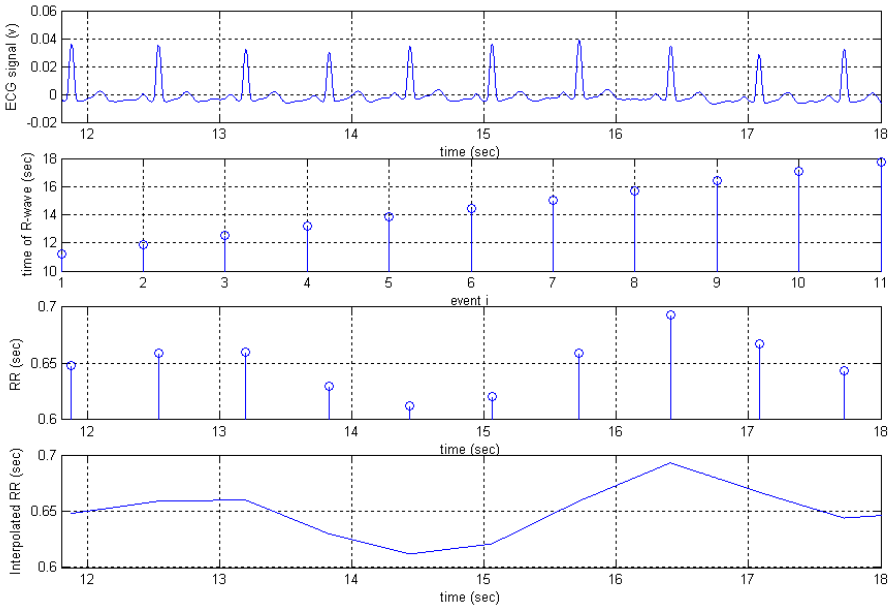



Figure 6.5: HRV Signal Formation

Time domain analysis of HRV is done by calculating the mean and the standard deviation of the signal for the whole record. Lastly, power spectral density estimate (PSD) of the HRV signal is found using the covariance method.

## 6.2 Respiratory Signal Processing

Respiration is recorded simultaneously with ECG data. So in signal processing, time axis used in processing should be the same with ECG. That's why filtering and processing is done through the same time variable "t".

### 6.2.1 Filtering

**Highpass Filtering** To reduce error in respiratory rate calculations, baseline differences on the signal was removed by highpass filtering. A moving average filter with 3db cut-off frequency at $f_c = 0.075$Hz is designed. At 1KHz sampling rate $N = 5000$ is selected as filter length. Frequency response of designed highpass filter is shown in figure 6.6.

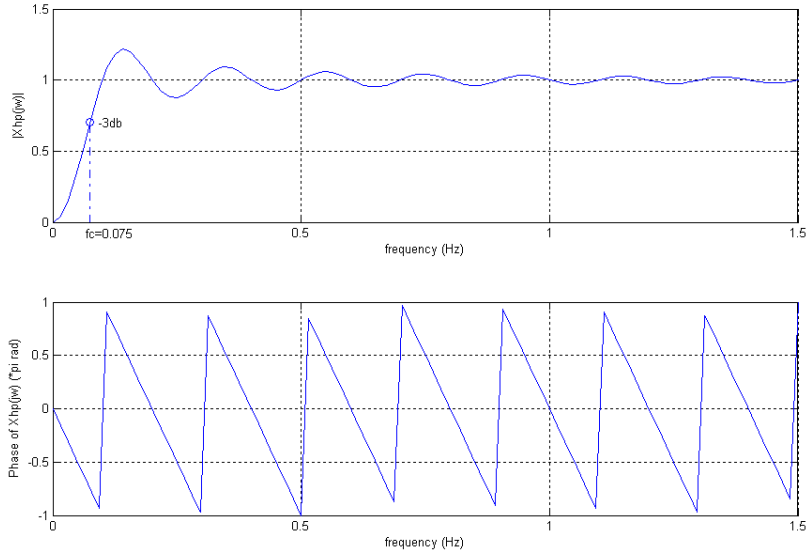



Figure 6.6: Frequency Response of the Respiratory Highpass Filter

The filter response is

$$X_{hp}(j\omega) = 1 - e^{-j\omega/2}\frac{sinc(5000\omega)}{sinc(\omega/2)} \tag{6.9}$$

**Lowpass Filtering** The lowpass filter designed for ECG is used also in respiration signal processing. Thus, $50Hz$ interference is reduced. There is also another MA lowpass filter with $N = 320$ (at $1KHz$ sampling rate) used for noise reduction in respiratory signal. The cut-off frequency of this filter is at $f_c = 1.4Hz$. Since respiration rate is below 1Hz, the signal is not corrupted. Its frequency response is given in the figure 6.7.

The filters are successively applied to respiratory signal as shown in figure 6.8.

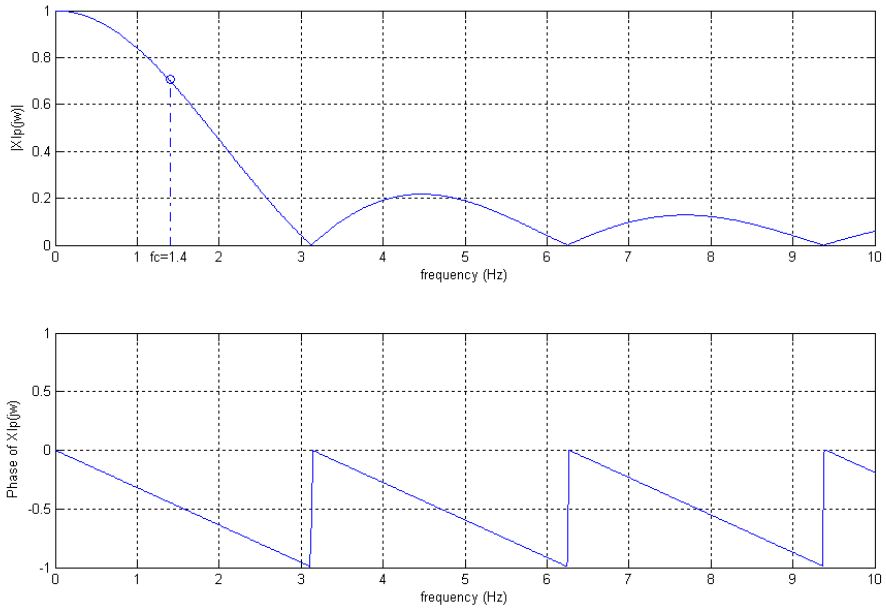



Figure 6.7: Frequency Response of the Respiratory Lowpass Filter

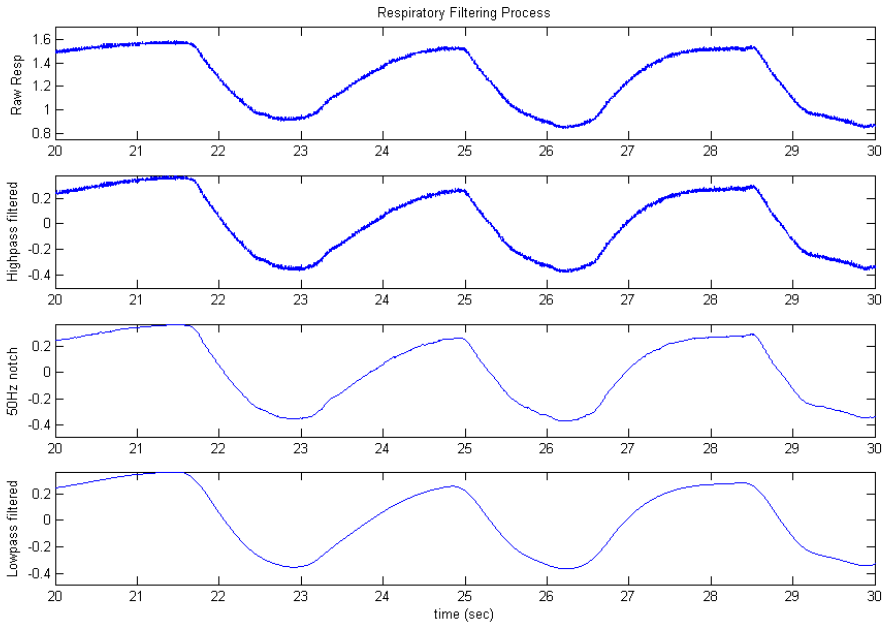



Figure 6.8: Successive Application of the Respiratory Signal Filters

### 6.2.2 Respiration Variability Analysis

Due to thermocouple circuitry, respiration signal rises during expiration. On the other hand, signal falls after inspiration starts. Since the inspiration is not uniform during breathing, respiratory fiducial points are determined according to initiation of expiration. The same differentiation and threshold method is used with R-wave detection for purpose. Firstly the signal is differentiated by filtering with

$$respfilt[n] = [2/10, 1/10, 0, -1/10, -2/10]$$

If the derivative is above a specified threshold, the corresponding point is a fiducial point of respiration. Application of method on a recorded respiration signal is shown in figure 6.9.

Respiration variability signal is found, by interpolating discrete time-series function of successive fiducial points. Respiration variability signal is then analyzed with its mean,standard deviation and histogram.

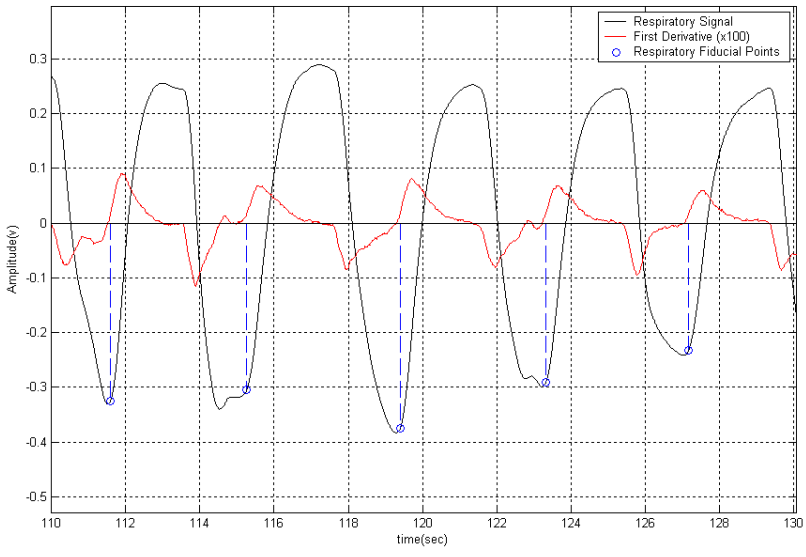



Figure 6.9: Respiratory Fiducial Point Determination

# Chapter 7

# Recording Results

The recording experiments were done on different patients. In these experiments, ECG and respiration of the patient were recorded by the following method:

- Firstly, during spontaneous breathing of the patient, ECG and respiration were recorded for $4-5$ minutes.
- Secondly, a metronome was adjusted to 2 seconds and the patient controlled his/her respiration to have a constant period at 4-sec. During this breathing, ECG and respiration were recorded for $4-5$ minutes.

Results from two patients are given below.

## 7.1 Results of Patient 1

### 7.1.1 Spontaneous Breathing

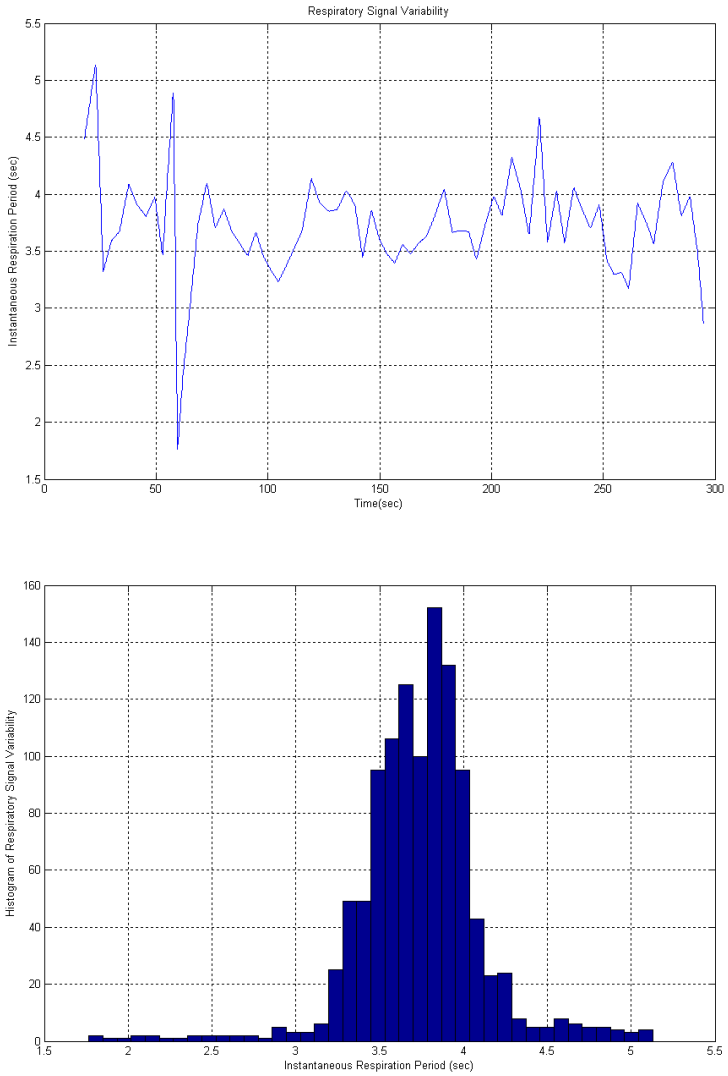



Figure 7.1: Spontaneous Breathing of Patient 1: Respiration Variability

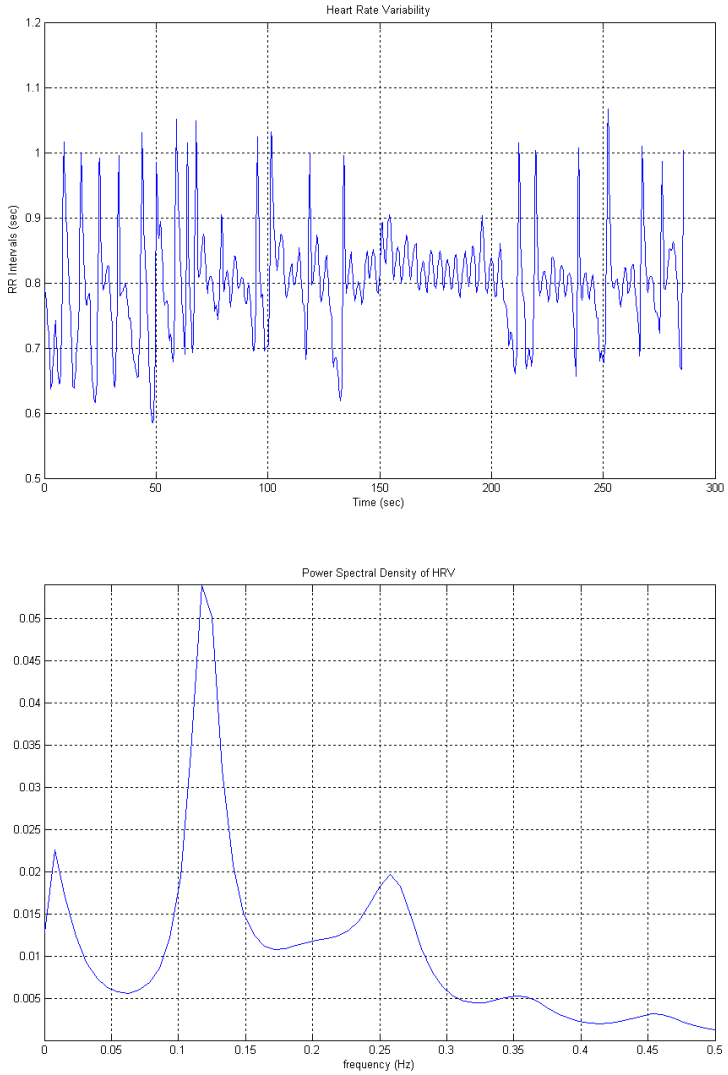



Figure 7.2: Spontaneous Breathing of Patient 1: HRV

| Breathing Type | Spontaneous | Controlled |
|---|---|---|
| Respiratory Mean | 3.7431 | 3.9509 |
| Respiratory Standard Deviation | 0.3754 | 0.2197 |
| HRV Mean | 0.7986 | 0.7214 |
| HRV Standard Deviation | 0.0758 | 0.0977 |

Table 7.1: Time Domain Parameters of Patient 1 Records

### 7.1.2 Controlled Breathing

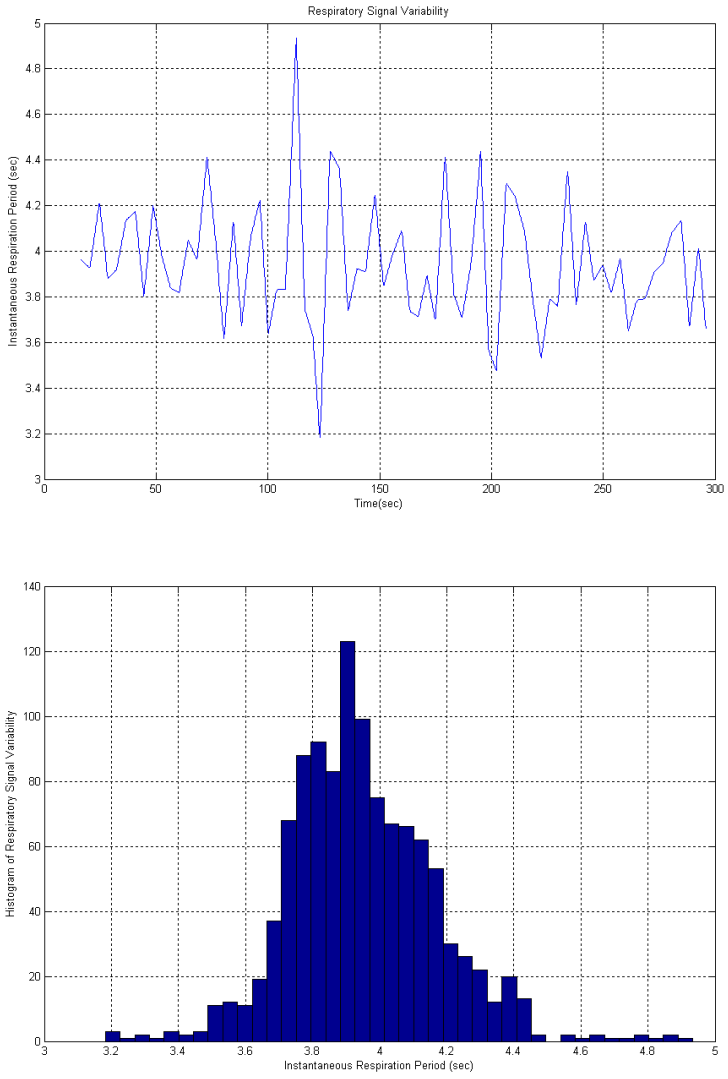



Figure 7.3: Controlled Breathing of Patient 1: Respiration Variability

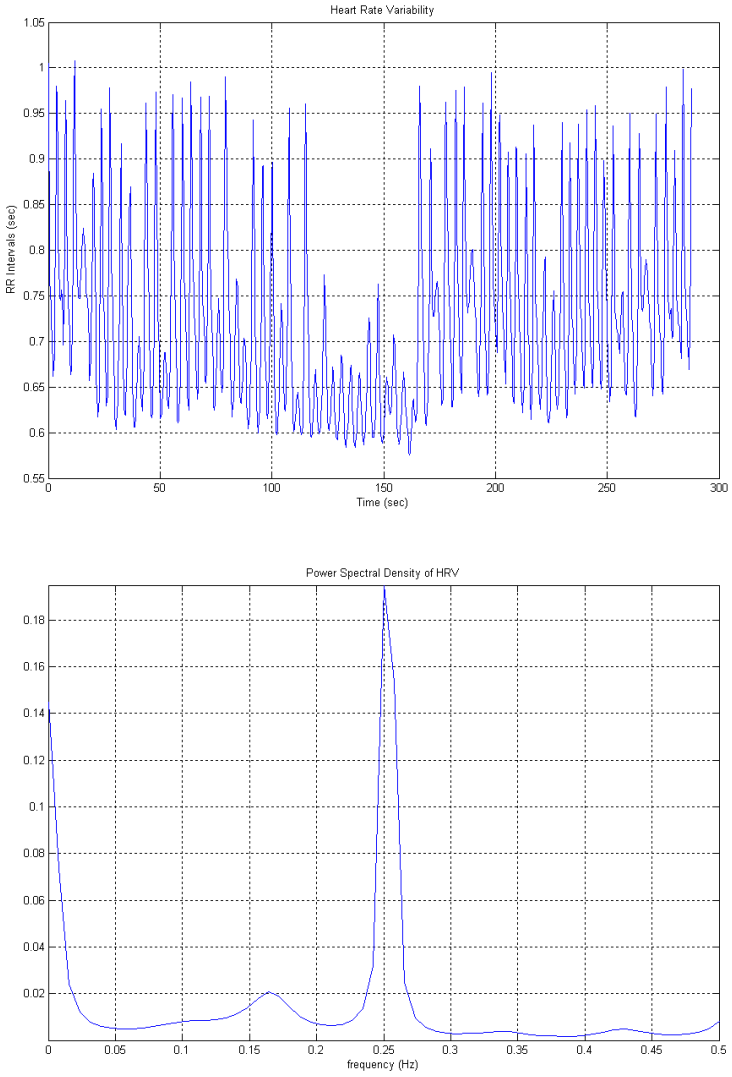



Figure 7.4: Controlled Breathing of Patient 1: HRV

## 7.2 Results of Patient 2

### 7.2.1 Spontaneous Breathing

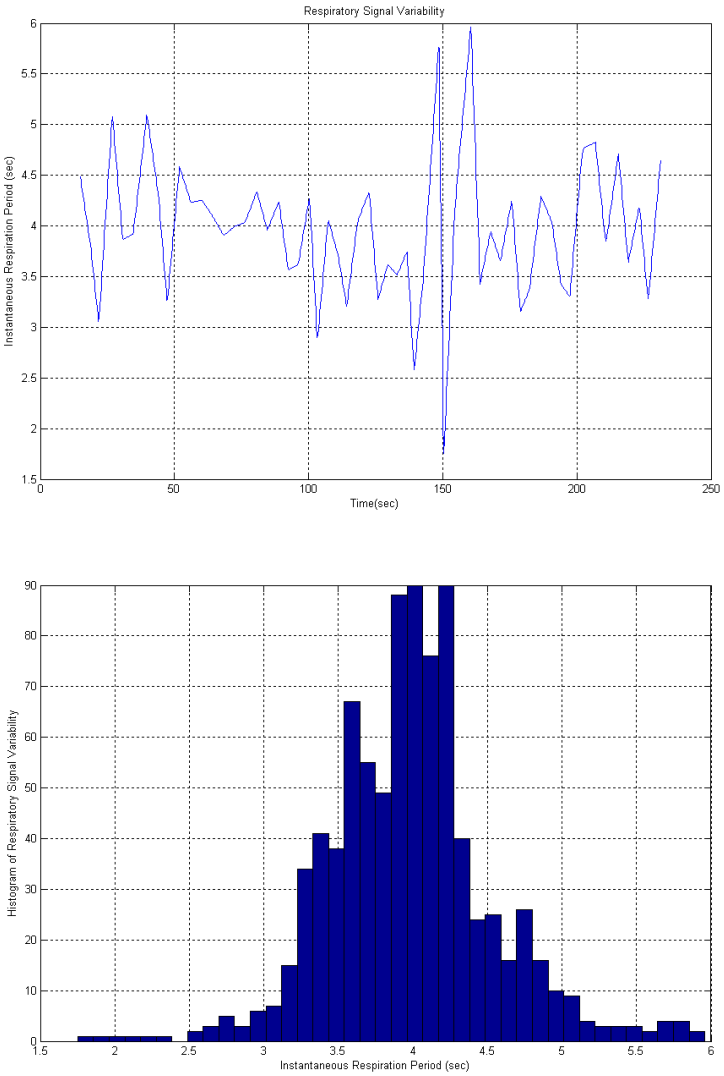



Figure 7.5: Spontaneous Breathing of Patient 2: Respiration Variability

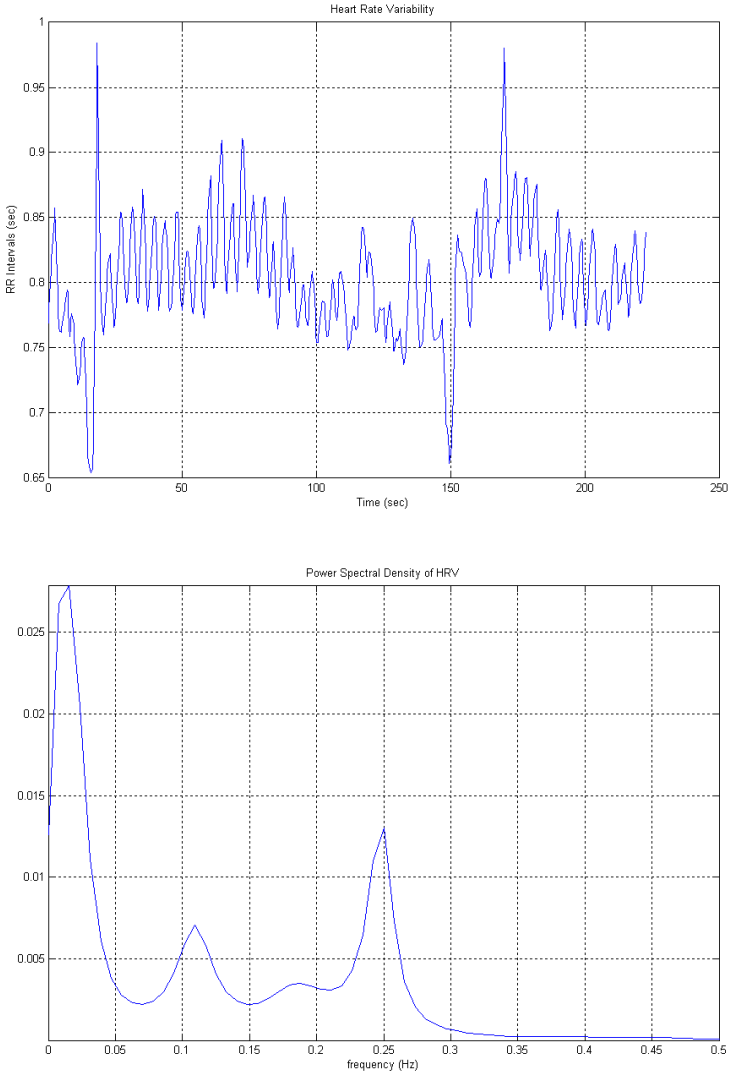



Figure 7.6: Spontaneous Breathing of Patient 2: HRV

| Breathing Type | Spontaneous | Controlled |
|---|---|---|
| Respiratory Mean | 3.9809 | 4.1793 |
| Respiratory Standard Deviation | 0.5501 | 0.2681 |
| HRV Mean | 0.8039 | 0.7898 |
| HRV Standard Deviation | 0.0438 | 0.0453 |

Table 7.2: Time Domain Parameters of Patient 2 Records

### 7.2.2 Controlled Breathing

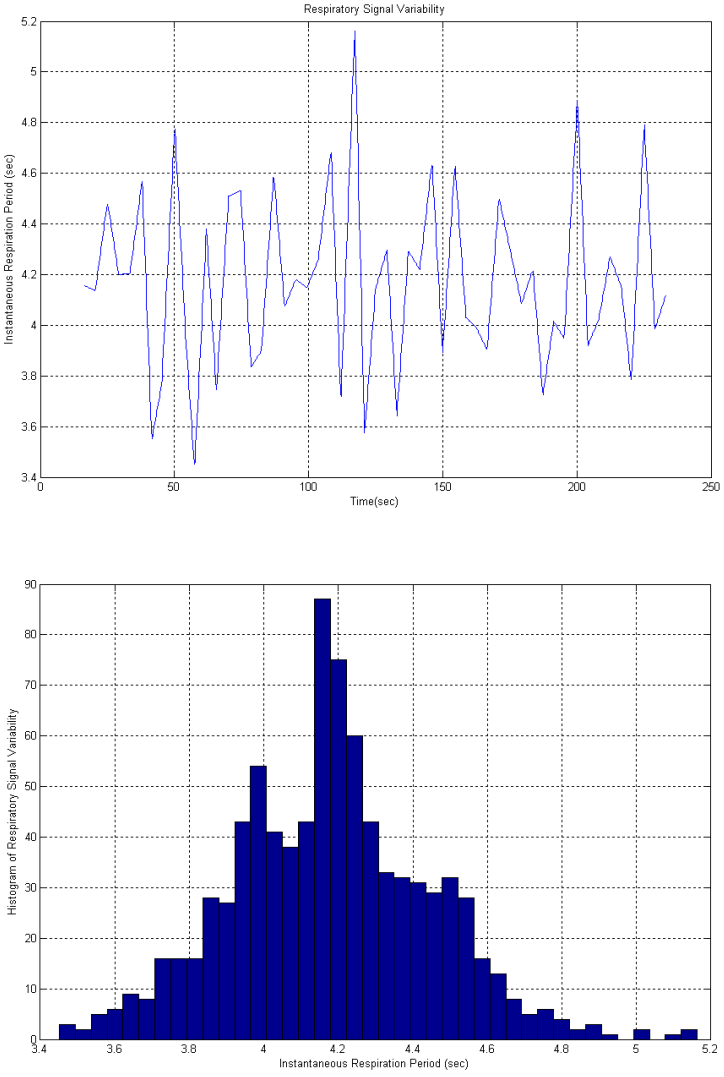



Figure 7.7: Controlled Breathing of Patient 2: Respiration Variability

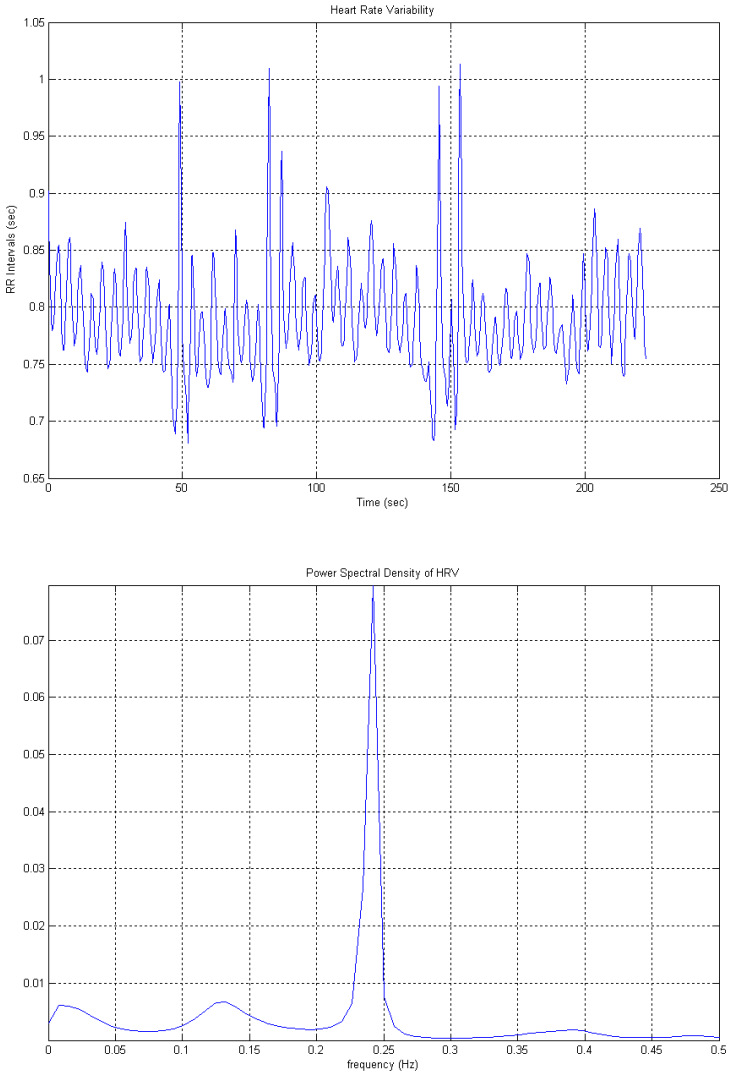



Figure 7.8: Controlled Breathing of Patient 2: HRV

# Chapter 8

# Discussion

The recording results show that, in spontaneous breathing, standard deviation of the respiratory signal is large. Power spectral estimate of HRV has its maximum power in Very Low Frequency (VLF, $f < 0.04$Hz) and Low Frequency (LF, $f \tilde{=} 0.1$Hz) components.

However, in controlled breathing experiments, decrease in the standard deviation of respiration causes the power spectral estimate of HRV to change. The maximum power is observed in the High Frequency (HF) component, which shows the respiratory rate.

The experiments were done for short-term recordings of $4-5$ minutes. In case of long-term recordings, more reliable results can be observed in power spectral estimation of the HRV.

In the instrumentation of the device, power consumption was not an important design limitation for us. Therefore, selection of the circuit elements such as A/D converter, resistors and capacitors was not based on low power consumption. With careful selection of these parts, a lower power device can be implemented.

The data acquisition sytem was implemented with USB Bulk Transfer method, which satisfies data accuracy rather than delivery time accuracy. However, in the experiments, if the USB 2.0 hub of the computer is connected to only this device,

the system works like a real-time implementation.

Device compliance tests for AAMI standards have satisfactory results for the patient source, the patient sink and the chassis source currents.

# Chapter 9

# Conclusion

A PC-based HRV recording system with respiration was designed and implemented in this study. Respiration was recorded from the nose by a thermocouple which had been placed in a nasal tube. HRV recording was done by a single-channel ECG amplifier. USB interface was used for the PC communications.

Firstly, its safety tests were performed for compliance with AAMI standards.

Secondly, successful short-term recording experiments were done with different patients for HRV changes between spontaneous breathing and controlled breathing. After these experiments, analysis of recorded data revealed differences in both time and frequency domain analysis of the HRV.

With its data acquisition speed and patient comfort, the device gives accurate results of HRV and respiration analysis. We believe that in the future, the exact effects of respiration on the HRV will be clearly determined by use of such a simultaneous recording device.

# Bibliography

[1] Heart rate variability: standards of measurement, physiological interpretation and clinical use. task force of the european society of cardiology and the north american society of pacing and electrophysiology. *AHA Circulation*, 93(5):1043–1065, March 1996.

[2] J. Axelson. *USB Complete: Everything You Need to Develop Custom USB Peripherals*. Lakeview Research, Madison, 1999.

[3] G. Baselli, S. Cerutti, S. Civardi, F. Lombardi, A. Malliani, M. Merri, M. Pagani, and G. Rizzo. Heart rate variability signal processing: A quantitative approach as an aid to diagnosis in cardiovascular pathologies. *Int J Biomed Comput*, 20(1-2):51–70, January 1987.

[4] L. Bernardi, J. Wdowczyk-Szulc, C. Valenti, S. Castoldi, C. Passino, G. Spadacini, and P. Sleight. Effects of controlled breathing, mental activity and mental stress with or without verbalization on heart rate variability. *J Am Coll Cardiol*, 35(6):1462–1469, May 2000.

[5] Association for the Advancement of Medical Instrumentation. American national standard for diagnostic electrocardiographic devices. Standard EC11, American National Standards Institute, September 1982.

[6] Association for the Advancement of Medical Instrumentation. American national standard, safe current limits for electromedical apparatus. Standard ES1, American National Standards Institute, December 1993.

[7] G. M. Friesen, T. C. Jannett, M. A. Jadallah, S. L. Yates, S. R. Quint, and H. T. Nagle. A comparison of the noise sensitivity of nine qrs detection algorithms. *IEEE Transactions on Biomedical Engineering*, 37(1):85–98, January 1990.

[8] J. A. Hirsh and B. Bishop. Respiratory sinus arrhythmia in humans: how breathing pattern modulates heart rate. *Am J Physiol Heart Circ Physiol*, 241(4):H620–H629, 1981.

[9] R. Pallas-Areny and J. G. Webster. Common mode rejection ratio for cascaded differential amplifier stages. *IEEE Transactions on Instrumentation and Measurement*, 40(4):677–681, August 1991.

[10] R. Pallas-Areny and J. G. Webster. Common mode rejection ratio in differential amplifiers. *IEEE Transactions on Instrumentation and Measurement*, 40(4):669–676, August 1991.

[11] C. S. Rangan, G. R. Sarma, and V. S. V. Mani. *Instrumentation Devices and Systems*. Tata McGraw-Hill, New Delhi, 1983.

[12] J. D. Schipke, M. Pelzer, and G. Arnold. Effect of respiration rate on short-term heart rate variability. *J Clin Basic Cardiol*, 2:92–95, 1999.

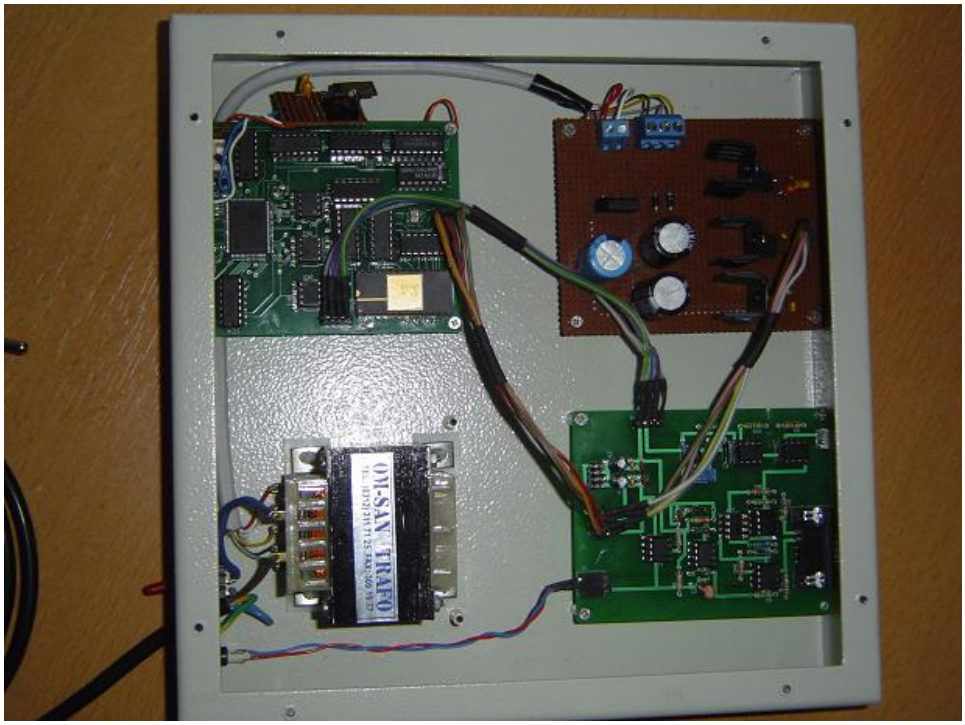
[13] Lt. Col. K. K. Tripathi. Respiration and heart rate variability: a review with special reference to its application in aerospace medicine. *Int J Aerospace Med*, 48(1):64–75, 2004.

[14] M. Ursino and E. Magosso. Role of short-term cardiovascular regulation in heart period variability: a modeling study. *Am J Physiol Heart Circ Physiol*, 284:1479–1493, April 2003.

[15] J. G. Webster. *Medical Instrumentation Application and Design*. John Wiley and Sons, New York, 1998.

# Appendix A

# Final Implementation of the Device


OL-SAN TRAFO

# Appendix B

# The Source Codes

## B.1 Instruction Set for the Microcontroller

CYPRESS

*Table B-1 Legend for Instruction Set Table*

| Symbol | Function |
|---|---|
| A | Accumulator |
| Rn | Register (R0–R7, in the bank selected by RS1:RS0) |
| direct | Internal RAM location (0x00 - 0x7F in the "Lower 128", or 0x80 - 0xFF in "SFR" space) |
| @Ri | Internal RAM location (0x00 - 0x7F in the "Lower 128", or 0x80 - 0xFF in the "Upper 128") pointed to by R0 or R1 |
| rel | Program-memory offset (-128 to +127 bytes relative to the first byte of the following instruction). Used by conditional jumps and SJMP. |
| bit | Bit address (0x20 - x2F in the "Lower 128," and SFRs 0x80, 0x88, ...., 0xF0, 0xF8) |
| #data | 8-bit constant (0 - 255) |
| #data16 | 16-bit constant (0 - 65535) |
| addr16 | 16-bit destination address; used by LCALL and LJMP, which branch anywhere in program memory |
| addr11 | 11-bit destination address; used by ACALL and AJMP, which branch only within the current 2K page of program memory (i.e., the upper 5 address bits are copied from the PC) |
| PC | Program Counter; holds the address of the currently-executing instruction. For the purposes of "ACALL", "AJMP", and "MOVC A,@A+PC" instructions, the PC holds the address of the first byte of the instruction *following* the currently-executing instruction. |

*Table B-2 EZ-USB Instruction Set*

| Mnemonic | Description | Bytes | Cycles | PSW Flags Affected | Opcode (Hex) |
|---|---|---|---|---|---|
| **Arithmetic** | | | | | |
| ADD A, Rn | Add register to A | 1 | 1 | CY OV AC | 28-2F |
| ADD A, direct | Add direct byte to A | 2 | 2 | CY OV AC | 25 |
| ADD A, @Ri | Add data memory to A | 1 | 1 | CY OV AC | 26-27 |
| ADD A, #data | Add immediate to A | 2 | 2 | CY OV AC | 24 |
| ADDC A, Rn | Add register to A with carry | 1 | 1 | CY OV AC | 38-3F |
| ADDC A, direct | Add direct byte to A with carry | 2 | 2 | CY OV AC | 35 |
| ADDC A, @Ri | Add data memory to A with carry | 1 | 1 | CY OV AC | 36-37 |
| ADDC A, #data | Add immediate to A with carry | 2 | 2 | CY OV AC | 34 |
| SUBB A, Rn | Subtract register from A with borrow | 1 | 1 | CY OV AC | 98-9F |
| SUBB A, direct | Subtract direct byte from A with borrow | 2 | 2 | CY OV AC | 95 |
| SUBB A, @Ri | Subtract data memory from A with borrow | 1 | 1 | CY OV AC | 96-97 |
| SUBB A, #data | Subtract immediate from A with borrow | 2 | 2 | CY OV AC | 94 |
| INC A | Increment A | 1 | 1 | | 04 |
| INC Rn | Increment register | 1 | 1 | | 08-0F |
| INC direct | Increment direct byte | 2 | 2 | | 05 |

Table B-2 EZ-USB Instruction Set (Continued)

| Mnemonic | Description | Bytes | Cycles | PSW Flags Affected | Opcode (Hex) |
|---|---|---|---|---|---|
| INC @ Ri | Increment data memory | 1 | 1 | | 06-07 |
| DEC A | Decrement A | 1 | 1 | | 14 |
| DEC Rn | Decrement Register | 1 | 1 | | 18-1F |
| DEC direct | Decrement direct byte | 2 | 2 | | 15 |
| DEC @Ri | Decrement data memory | 1 | 1 | | 16-17 |
| INC DPTR | Increment data pointer | 1 | 3 | | A3 |
| MUL AB | Multiply A and B (unsigned; product in B:A) | 1 | 5 | CY=0 OV | A4 |
| DIV AB | Divide A by B (unsigned; quotient in A, remainder in B) | 1 | 5 | CY=0 OV | 84 |
| DA A | Decimal adjust A | 1 | 1 | CY | D4 |
| **Logical** | | | | | |
| ANL, Rn | AND register to A | 1 | 1 | | 58-5F |
| ANL A, direct | AND direct byte to A | 2 | 2 | | 55 |
| ANL A, @Ri | AND data memory to A | 1 | 1 | | 56-57 |
| ANL A, #data | AND immediate to A | 2 | 2 | | 54 |
| ANL direct, A | AND A to direct byte | 2 | 2 | | 52 |
| ANL direct, #data | AND immediate data to direct byte | 3 | 3 | | 53 |
| ORL A, Rn | OR register to A | 1 | 1 | | 48-4F |
| ORL A, direct | OR direct byte to A | 2 | 2 | | 45 |
| ORL A, @Ri | OR data memory to A | 1 | 1 | | 46-47 |
| ORL A, #data | OR immediate to A | 2 | 2 | | 44 |
| ORL direct, A | OR A to direct byte | 2 | 2 | | 42 |
| ORL direct, #data | OR immediate data to direct byte | 3 | 3 | | 43 |
| XRL A, Rn | Exclusive-OR register to A | 1 | 1 | | 68-6F |
| XRL A, direct | Exclusive-OR direct byte to A | 2 | 2 | | 65 |
| XRL A, @Ri | Exclusive-OR data memory to A | 1 | 1 | | 66-67 |
| XRL A, #data | Exclusive-OR immediate to A | 2 | 2 | | 64 |
| XRL direct, A | Exclusive-OR A to direct byte | 2 | 2 | | 62 |
| XRL direct, #data | Exclusive-OR immediate to direct byte | 3 | 3 | | 63 |
| CLR A | Clear A | 1 | 1 | | E4 |
| CPL A | Complement A | 1 | 1 | | F4 |
| SWAP A | Swap nibbles of a | 1 | 1 | | C4 |
| RL A | Rotate A left | 1 | 1 | | 23 |
| RLC A | Rotate A left through carry | 1 | 1 | CY | 33 |
| RR A | Rotate A right | 1 | 1 | | 03 |
| RRC A | Rotate A right through carry | 1 | 1 | CY | 13 |
| **Data Transfer** | | | | | |
| MOV A, Rn | Move register to A | 1 | 1 | | E8-EF |

CYPRESS

*Table B-2 EZ-USB Instruction Set (Continued)*

| Mnemonic | Description | Bytes | Cycles | PSW Flags Affected | Opcode (Hex) |
|---|---|---|---|---|---|
| MOV A, direct | Move direct byte to A | 2 | 2 | | E5 |
| MOV A, @Ri | Move data byte at Ri to A | 1 | 1 | | E6-E7 |
| MOV A, #data | Move immediate to A | 2 | 2 | | 74 |
| MOV Rn, A | Move A to register | 1 | 1 | | F8-FF |
| MOV Rn, direct | Move direct byte to register | 2 | 2 | | A8-AF |
| MOV Rn, #data | Move immediate to register | 2 | 2 | | 78-7F |
| MOV direct, A | Move A to direct byte | 2 | 2 | | F5 |
| MOV direct, Rn | Move register to direct byte | 2 | 2 | | 88-8F |
| MOV direct, direct | Move direct byte to direct byte | 3 | 3 | | 85 |
| MOV direct, @Ri | Move data byte at Ri to direct byte | 2 | 2 | | 86-87 |
| MOV direct, #data | Move immediate to direct byte | 3 | 3 | | 75 |
| MOV @Ri, A | MOV A to data memory at address Ri | 1 | 1 | | F6-F7 |
| MOV @Ri, direct | Move direct byte to data memory at address Ri | 2 | 2 | | A6-A7 |
| MOV @Ri, #data | Move immediate to data memory at address Ri | 2 | 2 | | 76-77 |
| MOV DPTR, #data16 | Move 16-bit immediate to data pointer | 3 | 3 | | 90 |
| MOVC A, @A+DPTR | Move code byte at address DPTR+A to A | 1 | 3 | | 93 |
| MOVC A, @A+PC | Move code byte at address PC+A to A | 1 | 3 | | 83 |
| MOVX A, @Ri | Move external data at address Ri to A | 1 | 2-9* | | E2-E3 |
| MOVX A, @DPTR | Move external data at address DPTR to A | 1 | 2-9* | | E0 |
| MOVX @Ri, A | Move A to external data at address Ri | 1 | 2-9* | | F2-F3 |
| MOVX @DPTR, A | Move A to external data at address DPTR | 1 | 2-9* | | F0 |
| PUSH direct | Push direct byte onto stack | 2 | 2 | | C0 |
| POP direct | Pop direct byte from stack | 2 | 2 | | D0 |
| XCH A, Rn | Exchange A and register | 1 | 1 | | C8-CF |
| XCH A, direct | Exchange A and direct byte | 2 | 2 | | C5 |
| XCH A, @Ri | Exchange A and data memory at address Ri | 1 | 1 | | C6-C7 |
| XCHD A, @Ri | Exchange the low-order nibbles of A and data memory at address Ri | 1 | 1 | | D6-D7 |
| ** Number of cycles is user-selectable. See Section B.2.2. "Stretch Memory Cycles (Wait States)".* | | | | | |
| **Boolean** | | | | | |
| CLR C | Clear carry | 1 | 1 | CY=0 | C3 |
| CLR bit | Clear direct bit | 2 | 2 | | C2 |
| SETB C | Set carry | 1 | 1 | CY=1 | D3 |
| SETB bit | Set direct bit | 2 | 2 | | D2 |
| CPL C | Complement carry | 1 | 1 | CY | B3 |

Table B-2 EZ-USB Instruction Set (Continued)

| Mnemonic | Description | Bytes | Cycles | PSW Flags Affected | Opcode (Hex) |
|---|---|---|---|---|---|
| CPL bit | Complement direct bit | 2 | 2 | | B2 |
| ANL C, bit | AND direct bit to carry | 2 | 2 | CY | 82 |
| ANL C, /bit | AND inverse of direct bit to carry | 2 | 2 | CY | B0 |
| ORL C, bit | OR direct bit to carry | 2 | 2 | CY | 72 |
| ORL C, /bit | OR inverse of direct bit to carry | 2 | 2 | CY | A0 |
| MOV C, bit | Move direct bit to carry | 2 | 2 | CY | A2 |
| MOV bit, C | Move carry to direct bit | 2 | 2 | | 92 |
| **Branching** | | | | | |
| ACALL addr11 | Absolute call to subroutine | 2 | 3 | | 11-F1 |
| LCALL addr16 | Long call to subroutine | 3 | 4 | | 12 |
| RET | Return from subroutine | 1 | 4 | | 22 |
| RETI | Return from interrupt | 1 | 4 | | 32 |
| AJMP addr11 | Absolute jump unconditional | 2 | 3 | | 01-E1 |
| LJMP addr16 | Long jump unconditional | 3 | 4 | | 02 |
| SJMP rel | Short jump (relative address) | 2 | 3 | | 80 |
| JC rel | Jump if carry = 1 | 2 | 3 | | 40 |
| JNC rel | Jump if carry = 0 | 2 | 3 | | 50 |
| JB bit, rel | Jump if direct bit = 1 | 3 | 4 | | 20 |
| JNB bit, rel | Jump if direct bit = 0 | 3 | 4 | | 30 |
| JBC bit, rel | Jump if direct bit = 1, then clear the bit | 3 | 4 | | 10 |
| JMP @ A+DPTR | Jump indirect to address DPTR+A | 1 | 3 | | 73 |
| JZ rel | Jump if accumulator = 0 | 2 | 3 | | 60 |
| JNZ rel | Jump if accumulator is non-zero | 2 | 3 | | 70 |
| CJNE A, direct, rel | Compare A to direct byte; jump if not equal | 3 | 4 | CY | B5 |
| CJNE A, #d, rel | Compare A to immediate; jump if not equal | 3 | 4 | CY | B4 |
| CJNE Rn, #d, rel | Compare register to immediate; jump if not equal | 3 | 4 | CY | B8-BF |
| CJNE @ Ri, #d, rel | Compare data memory to immediate; jump if not equal | 3 | 4 | CY | B6-B7 |
| DJNZ Rn, rel | Decrement register; jump if not zero | 2 | 3 | | D8-DF |
| DJNZ direct, rel | Decrement direct byte; jump if not zero | 3 | 4 | | D5 |
| **Miscellaneous** | | | | | |
| NOP | No operation | 1 | 1 | | 00 |

There is an additional reserved opcode (A5) that performs the same function as NOP.

## B.2 The Microcontroller Software Source Code

Listing B.1: Software in A51 format

```
;-----------------------------------------------------------------------
$NOMOD51                                        ; disable predefined 8051 registers
$nolist
$include (ezregs.inc)   ; EZ-USB register assignments
$list
;
NAME    adc
; ----------------------------------------------

CSEG    AT 0x00

LJMP  Start

CSEG    AT 0x0B
        timer0: LJMP second_data

CSEG    AT 0x1B
        timer1: LJMP first_data

CSEG    AT 0x43

LJMP    USB_Jump_Table

CSEG    AT 0x53
        EXT4INT: LJMP  Int4Isr          ; Data Ready to Read ISR

CSEG    AT 0x5B
        EXT5INT: LJMP  Int5Isr          ; A/D End of Conversion ISR

CSEG AT 500H

USB_Jump_Table:
                ljmp    ISR_Sudav       ; Setup Data Available
                db      0
                ljmp    ISR_Sof ; Start of Frame
                db      0
                ljmp    ISR_Sutok       ; Setup Data Loading
                db      0
                ljmp    ISR_Susp        ; Global Suspend
                db      0
                ljmp    ISR_Ures        ; USB Reset
                db      0
                ljmp    ISR_IBN     ; IN Bulk NAK interrupt
                db      0
                ljmp    ISR_Ep0in       ; End Point 0 In
                db      0
                ljmp    ISR_Ep0out      ; End Point 0 Out
                db      0
                ljmp    ISR_Ep1in       ; End Point 1 In
                db      0
                ljmp    ISR_Ep1out      ; End Point 1 Out
                db      0
                ljmp    ISR_Ep2in
                db      0
                ljmp    ISR_Ep2out
                db      0
                ljmp    ISR_Ep3in
                db      0
                ljmp    ISR_Ep3out
```

```
                db      0
                ljmp    ISR_Ep4in
                db      0
                ljmp    ISR_Ep4out
                db      0
                ljmp    ISR_Ep5in
                db      0
                ljmp    ISR_Ep5out
                db      0
                ljmp    ISR_Ep6in
                db      0
                ljmp    ISR_Ep6out
                db      0
                ljmp    ISR_Ep7in
                db      0
                ljmp    ISR_Ep7out
                db      0

    CSEG    AT 0x200

Start:
        mov     r1,#11000011b   ;holds sampling rate info: Starting f=1kHz
        mov     r2,#0   ;counts bytes read from ADC
        mov     r3,#0
        mov     r4,#0
        mov     r5,#0
        mov     r6,#0   ;switches between buffers to write data

        mov     dptr,#USBBAV    ; ENABLE AUTOVECTOR
        movx    a,@dptr
        setb    acc.0           ; AVEN=1
        movx    @dptr,a

        mov     dptr,#OUT07IEN  ; EP0—7 OUT Interrupt Enable Register
        mov     a,#01010000b
        movx    @dptr,a         ; Enable EP6 & EP4 OUT interrupt
        SETB    EIE.0           ; Enable INT2 interrupt

        SETB    IE.7    ;ENABLE ALL INTERRUPTS
        mov     dptr, #OUT6BC   ; load the EP6OUT byte count to arm OUT transfer
        mov     a, #64          ; 64 byte payload
        movx    @dptr, a
        mov     dptr, #OUT4BC   ; load the EP6OUT byte count to arm OUT transfer
        mov     a, #64          ; 64 byte payload
        movx    @dptr, a


        mov     dptr, #PORTBCFG
        mov     a, #00110000b
        movx    @dptr,a

        mov     dptr, #PORTCCFG
        mov     a, #00000000
        movx    @dptr, a

        mov     dptr, #PORTACFG
        mov     a, #00000000
        movx    @dptr, a

        mov     dptr, #OEB
        mov     a, #11001111b
        movx    @dptr, a

        mov     dptr, #OEA
        mov     a, #00000000b
        movx    @dptr, a

        mov     dptr, #OEC
```

```
            mov     a, #00000000b
            movx    @dptr, a

            mov     dptr, #PINSB
            movx    a, @dptr
            clr     acc.1                   ; initialize CE=0
            mov     dptr, #OUTB
            movx    @dptr, a

            mov     dptr, #PINSB            ; initialize multiplexer ch.
            movx    a, @dptr
            clr     acc.0                   ; by clearing MUX
            mov     dptr, #OUTB
            movx    @dptr, a

            mov     dptr, #PINSB    ;
            movx    a, @dptr
            clr     acc.2           ; initialize R/C=0
            mov     dptr, #OUTB
            movx    @dptr, a

            mov     r2, #0

    here:
            cjne    r2, #64, intret
            mov     a, r6
            cpl     a                       ; change buffer to send
            mov     r6,a
    wait:

            cjne    r6, #0, writebuf2

    writebuf:
            mov     dptr, #IN4CS            ; EP4IN Control & Status register
            movx    a, @dptr
            jb      acc.1, writebuf

            mov     r2, #0

            mov     dptr, #IN4BC    ; load the EP4IN byte count to re-arm IN transfer
            mov     a, #64          ; 64 byte payload
            movx    @dptr, a
            jmp     here

    writebuf2:
            mov     dptr, #IN2CS            ; EP2IN Control & Status register
            movx    a, @dptr
            jb      acc.1, writebuf2

            mov     r2, #0

            mov     dptr, #IN2BC    ; load the EP2IN byte count to re-arm IN transfer
            mov     a, #64          ; 64 byte payload
            movx    @dptr, a

    intret:
            jmp     here

    ;=================ISR for starting first channel conversion =========
    first_data:
            push dps
            push dpl
            push dpl1
            push dph1
            push acc

            mov     dptr, #PINSB
```

```
            movx    a, @dptr
            setb    acc.1           ; CE=1
            mov     dptr, #OUTB
            movx    @dptr, a

            call    delay

            mov     dptr, #PINSB
            movx    a, @dptr
            clr     acc.1           ;CE=0
            mov     dptr, #OUTB
            movx    @dptr, a

            call    delay

            SETB    EIE.2   ;ENABLE INT4 INTERRUPT


            mov     dptr, #PINSB            ; check which mux channel will be converted
            movx    a, @dptr
            jnb     acc.0, timerload        ; by checking PB0

            MOV     TH0,#11110100b  ; reset timer0 to convert second channel
            MOV     TL0,#00000000b
            SETB    TR0
            CLR     TF0

    timerload:
            MOV     TH1,r1          ; reset timer1 to convert first channel
            MOV     TL1,#00000011b

            SETB    TR1
            CLR     TF1

            pop acc
            pop dph1
            pop dpl1
            pop dpl
            pop dps

            RETI
    ;=============================================================
    ;=================ISR for starting second channel conversion =========
    second_data:
            push dps
            push dpl
            push dpl1
            push dph1
            push acc

            mov     dptr, #PINSB
            movx    a, @dptr
            setb    acc.1           ; CE=1
            mov     dptr, #OUTB
            movx    @dptr, a

            call    delay

            mov     dptr, #PINSB
            movx    a, @dptr
            clr     acc.1           ;CE=0
            mov     dptr, #OUTB
            movx    @dptr, a

            CALL    delay
            SETB    EIE.2   ;ENABLE INT4 INTERRUPT

            pop acc
```

```
        pop dph1
        pop dpl1
        pop dpl
        pop dps

        RETI
;=============================================================

;========== ISR for reading a byte ============================
Int5Isr:
        push dps
        push dpl
        push dpl1
        push dph1
        push acc


        mov     dptr, #PINSB            ; enable R/C to read converted data
        movx    a, @dptr
        setb    acc.2                   ; R/C=1
        mov     dptr, #OUTB

        call    delay

        mov     dptr, #PINSB
        movx    @dptr, a                ; enable output buffers of A/D
        setb    acc.1                   ; CE=1
        mov     dptr, #OUTB
        movx    @dptr, a

        call    delay

        mov     dptr, #PINSB
        movx    @dptr, a                ; initialize P/S — S/P registers
        clr     acc.1                   ; CE=0
        mov     dptr, #OUTB
        movx    @dptr, a

        call    delay
        call    delay

        mov     dptr, #PINSB
        movx    @dptr, a                ; initialize P/S — S/P registers
        setb    acc.1                   ; CE=1
        mov     dptr, #OUTB
        movx    @dptr, a

        mov     a, EXIF                 ; clear USB IRQ (INT5)
        clr     acc.7
        mov     EXIF, a

        pop acc
        pop dph1
        pop dpl1
        pop dpl
        pop dps

        RETI

;=============================================================
;========== ISR for reading a byte ============================
Int4Isr:
        push dps
        push dpl
        push dpl1
        push dph1
        push acc
```

```
            mov     dptr, #PINSB            ; multiplex the input
            movx    a, @dptr
            cpl     acc.0                   ; by complementing PB0
336         mov     dptr, #OUTB
            movx    @dptr, a

            mov     dptr,#PINSA             ; read ports and save to r4&r5
            movx    a,@dptr
341         mov     r4,a
            mov     dptr,#PINSC
            movx    a,@dptr
            mov     r5,a

346         mov     dptr, #PINSB            ; disable ADC
            movx    a, @dptr
            clr     acc.1                   ; CE=0
            mov     dptr, #OUTB
            movx    @dptr, a
351
            call    delay
            call    delay

            mov     dptr, #PINSB            ; change R/C to set ADC to convert mode
356         movx    a, @dptr
            clr     acc.2                   ; R/C=0
            mov     dptr, #OUTB
            movx    @dptr, a

361         cjne    r6, #0 , readportb

    readport:                               ; locate bytes read to bulk transfer buffers EP2IN & EP4IN
            mov     dptr, #IN2CS            ; Check EP2IN Control & Status register
            movx    a, @dptr
366         jb      acc.1, readport

            mov     a,r2
            mov     r3,a
            inc     r3
371
            mov     dptr,#IN2BUF-1

    loop:
            inc     dptr
376         djnz    r3, loop
            mov     a,r4
            movx    @dptr,a
            inc     dptr
            mov     a,r5
381         movx    @dptr,a
            inc     r2                      ; since 2 bytes written to IN BUFFER
            inc     r2                      ; pointer is increased by 2
            jmp     cont

386 readportb:                              ; after EP2IN is fully loaded, switch to EP4IN

            mov     dptr, #IN4CS            ; Check EP4IN Control & Status register
            movx    a, @dptr
            jb      acc.1, readportb
391
            mov     a,r2
            mov     r3,a
            inc     r3

396         mov     dptr,#IN4BUF-1

    loopb:
            inc     dptr
            djnz    r3, loopb
```

```
401         mov     a,r4
            movx    @dptr,a
            inc     dptr
            mov     a,r5
            movx    @dptr,a
406         inc     r2                      ; since 2 bytes written to IN BUFFER
            inc     r2                      ; pointer is increased by 2

    cont:
            mov     a, EXIF                 ; clear USB IRQ (INT4)
411         clr     acc.6
            mov     EXIF, a

            CLR     EIE.2           ;DISABLE INT4 INTERRUPT

416         pop acc
            pop dph1
            pop dpl1
            pop dpl
            pop dps
421
            RETI
    ;===================================
    delay:
            nop
426         nop
            nop
            nop
            nop
            nop
431         nop
            nop
            nop
            nop
            nop
436         nop
            nop
            nop
            nop
            nop
441         nop
            nop
            nop
            nop
            nop
446         nop
            nop
            nop
            nop
            nop
451         nop
            nop
            nop
            nop
            nop
456         nop

            ret

    ;============================================================
461 ;========== Interrupt Services for USB transactions===========

    ISR_Sudav:
                    RETI
    ISR_Sof:
466                 RETI
    ISR_Sutok:
                    RETI
```

```
ISR_Susp:
                RETI
ISR_Ures:
                RETI
ISR_IBN:
                RETI
ISR_Ep0in:
                RETI
ISR_Ep0out:
                RETI
ISR_Ep1in:
                RETI
ISR_Ep1out:
                RETI
ISR_Ep2in:
                RETI
ISR_Ep2out:
                RETI
ISR_Ep3in:
                RETI
ISR_Ep3out:
                RETI
ISR_Ep4in:
                RETI
ISR_Ep4out:                     ; ISR for changing sampling rate
        push dps
        push dpl
        push dpl1
        push dph1
        push acc

        mov     dptr,#OUT4BUF   ; write sampling rate info to system timer
        movx    a,@dptr
        mov     r1,a

        mov     a, EXIF                 ; clear USB IRQ (INT2)
        clr     acc.4
        mov     EXIF, a

        mov     dptr, #OUT07IRQ         ; clear IRQ flag
        mov     a,#00010000b
        movx    @dptr, a

        mov     dptr, #OUT4BC   ; load the EP6OUT byte count to re-arm OUT transfer
        mov     a, #64          ; any byte payload
        movx    @dptr, a

        pop acc
        pop dph1
        pop dpl1
        pop dpl
        pop dps
        RETI

ISR_Ep5in:
                RETI
ISR_Ep5out:
                RETI
ISR_Ep6in:
                RETI
ISR_Ep6out:
        push dps
        push dpl
        push dpl1
        push dph1
        push acc

        mov     dptr,#OUT6BUF
```

```
            movx    a,@dptr
            cjne    a, #16, another

            SETB    IE.3    ;ENABLE TIMER1 INTERRUPT
541         SETB    IE.1    ;ENABLE TIMER0 INTERRUPT
            SETB    EIE.3   ;ENABLE INT5 INTERRUPT
            MOV TMOD,#00000000b
            MOV TH1,#00000000b
            MOV TL1,#00000000b
546         SETB TR1        ; timer
            CLR TF1         ; enable

    another:
            mov     a, EXIF                 ; clear USB IRQ (INT2)
551         clr     acc.4
            mov     EXIF, a

            mov     dptr, #OUT07IRQ
            mov     a,#01000000b
556         movx    @dptr, a

            mov     dptr, #OUT6BC   ; load the EP6OUT byte count to re—arm OUT transfer
            mov     a, #64          ; 64 byte payload
            movx    @dptr, a
561
            pop acc
            pop dph1
            pop dpl1
            pop dpl
566         pop dps
            RETI

    ISR_Ep7in:
                    RETI
571 ISR_Ep7out:
                    RETI

    END
```

## B.3 The User Application Software Source Code

Listing B.2: Software in Microsoft Visual C/C++ format

```
#include <windows.h>
#include <math.h>
#include <malloc.h>
#include <assert.h>
#include <process.h>
#include <Mmsystem.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "main.h"
#include <winioctl.h>
#include "ezusbsys.h"

// Global constant for pi:
const double pi = 3.1415926535;
const char szAppName[]="WinApp";
const char szWndName[] = "Data Analysis";

// Global Parameters
HINSTANCE hInstance;
HWND hWnd;
HMENU hPopup = NULL;
HMENU hFileMenu;                /* file menu handle */
HMENU hOptionsMenu;             /* draw menu handle */
HMENU hMenu;                    /* menu bar handle */
HMENU hSamplingRate;            /* sampling menu handle */
HMENU hVoltageScale;            /* voltage scale menu handle */
HMENU hVoltageScale1;           /* voltage scale menu handle */
HMENU hVoltageScale2;           /* voltage scale menu handle */
MENUITEMINFO item;                      /* item info */

MSG msg;
WNDCLASS wndclass;
HANDLE hDevice;
HANDLE hRead;
FILE* output1;
FILE* output2;

// Ez-USB configuration variables
char pcDriverName[]="Ezusb-0";
const int Interface=0;
const int AltSetting=1;
const int PipeNumber1=1;
const int PipeNumber2=3;
const int PipeOut=6;
const int SamplingPipe=4;
const int PackCount=1;
const int PackSize=64;
const int BufCount=1;
const int FramesPerBuf=1;

// axis locations on the screen
int axisx1=512;
int axisy1=168;
int axisx2=512;
int axisy2=552;
```

```
    LARGE_INTEGER llnHPTimerFreq;   // High Performance Timer: Frequency
    LARGE_INTEGER llnHPT1;                  // High Performance Timer: Time 1
    LARGE_INTEGER llnHPT2;                  // High Performance Timer: Time 2
61  LARGE_INTEGER llnT_uSec;                // Time in microseconds

    long double datareceived;

    BOOL StopRead=TRUE;     // flag shows reading operation
66  BOOL Rec=FALSE; // flag shows on-line recording operation
    BOOL PipeSelect=TRUE;
    BYTE* isobuf=NULL;      // temporary buffer holding received 64-byte packet
    PULONG isobuf2=NULL;
    USHORT buffer[1024];    // buffer holds CH1 data to be drawn on the screen
71  USHORT buffer2[1024];   // buffer holds CH2 data to be drawn on the screen
    char* recbuffer1;       //      temporary buffer used while recording CH1
    char* recbuffer2;       //      temporary buffer used while recording CH2
    int reccount=0; // used in recording operation

76  // firmware data in Intel hex format
    INTEL_HEX_RECORD anchor[] = {
       1,0x7F92,0,{0x01},
       3,0x0,0,{0x02,0x02,0x00},
       3,0xb,0,{0x02,0x02,0xe1},
81     3,0x1b,0,{0x02,0x02,0x98},
       3,0x43,0,{0x02,0x05,0x00},
       3,0x53,0,{0x02,0x03,0x5a},
       3,0x5b,0,{0x02,0x03,0x10},
       16,0x500,0,{0x02,0x03,0xf7,0x00,0x02,0x03,0xf8,0x00,0x02,0x03,0xf9,0x00,0x02,0x03,0xfa,0x00},
86     16,0x510,0,{0x02,0x03,0xfb,0x00,0x02,0x03,0xfc,0x00,0x02,0x03,0xfd,0x00,0x02,0x03,0xfe,0x00},
       16,0x520,0,{0x02,0x03,0xff,0x00,0x02,0x04,0x00,0x00,0x02,0x04,0x01,0x00,0x02,0x04,0x02,0x00},
       16,0x530,0,{0x02,0x04,0x03,0x00,0x02,0x04,0x04,0x00,0x02,0x04,0x05,0x00,0x02,0x04,0x06,0x00},
       16,0x540,0,{0x02,0x04,0x32,0x00,0x02,0x04,0x33,0x00,0x02,0x04,0x34,0x00,0x02,0x04,0x35,0x00},
       8,0x550,0,{0x02,0x04,0x76,0x00,0x02,0x04,0x77,0x00},
91     16,0x200,0,{0x79,0xc3,0x7a,0x00,0x7b,0x00,0x7c,0x00,0x7d,0x00,0x7e,0x00,0x90,0x7f,0xaf,0xe0},
       16,0x210,0,{0xd2,0xe0,0xf0,0x90,0x7f,0xad,0x74,0x50,0xf0,0xd2,0xe8,0xd2,0xaf,0x90,0x7f,0xd1},
       16,0x220,0,{0x74,0x40,0xf0,0x90,0x7f,0xcd,0x74,0x40,0xf0,0x90,0x7f,0x94,0x74,0x30,0xf0,0x90},
       16,0x230,0,{0x7f,0x95,0x74,0x00,0xf0,0x90,0x7f,0x93,0x74,0x00,0xf0,0x90,0x7f,0x9d,0x74,0xcf},
       16,0x240,0,{0xf0,0x90,0x7f,0x9c,0x74,0x00,0xf0,0x90,0x7f,0x9e,0x74,0x00,0xf0,0x90,0x7f,0x9a},
96     16,0x250,0,{0xe0,0xc2,0xe1,0x90,0x7f,0x97,0xf0,0x90,0x7f,0x9a,0xe0,0xc2,0xe0,0x90,0x7f,0x97},
       16,0x260,0,{0xf0,0x90,0x7f,0x9a,0xe0,0xc2,0xe2,0x90,0x7f,0x97,0xf0,0x7a,0x00,0xba,0x40,0x26},
       16,0x270,0,{0xee,0xf4,0xfe,0xbe,0x00,0x11,0x90,0x7f,0xbc,0xe0,0x20,0xe1,0xf9,0x7a,0x00,0x90},
       16,0x280,0,{0x7f,0xbd,0x74,0x40,0xf0,0x80,0xe6,0x90,0x7f,0xb8,0xe0,0x20,0xe1,0xf9,0x7a,0x00},
       16,0x290,0,{0x90,0x7f,0xb9,0x74,0x40,0xf0,0x80,0xd5,0xc0,0x86,0xc0,0x82,0xc0,0x84,0xc0,0x85},
101    16,0x2a0,0,{0xc0,0xe0,0x90,0x7f,0x9a,0xe0,0xd2,0xe1,0x90,0x7f,0x97,0xf0,0x71,0xd6,0x90,0x7f},
       16,0x2b0,0,{0x9a,0xe0,0xc2,0xe1,0x90,0x7f,0x97,0xf0,0x71,0xd6,0xd2,0xea,0x90,0x7f,0x9a,0xe0},
       16,0x2c0,0,{0x30,0xe0,0x0a,0x75,0x8c,0xf4,0x75,0x8a,0x00,0xd2,0x8c,0xc2,0x8d,0x89,0x8d,0x75},
       16,0x2d0,0,{0x8b,0x03,0xd2,0x8e,0xc2,0x8f,0xd0,0xe0,0xd0,0x85,0xd0,0x84,0xd0,0x82,0xd0,0x86},
       16,0x2e0,0,{0x32,0xc0,0x86,0xc0,0x82,0xc0,0x84,0xc0,0x85,0xc0,0xe0,0x90,0x7f,0x9a,0xe0,0xd2},
106    16,0x2f0,0,{0xe1,0x90,0x7f,0x97,0xf0,0x71,0xd6,0x90,0x7f,0x9a,0xe0,0xc2,0xe1,0x90,0x7f,0x97},
       16,0x300,0,{0xf0,0x71,0xd6,0xd2,0xea,0xd0,0xe0,0xd0,0x85,0xd0,0x84,0xd0,0x82,0xd0,0x86,0x32},
       16,0x310,0,{0xc0,0x86,0xc0,0x82,0xc0,0x84,0xc0,0x85,0xc0,0xe0,0x90,0x7f,0x9a,0xe0,0xd2,0xe2},
       16,0x320,0,{0x90,0x7f,0x97,0x71,0xd6,0x90,0x7f,0x9a,0xf0,0xd2,0xe1,0x90,0x7f,0x97,0xf0,0x71},
       16,0x330,0,{0xd6,0x90,0x7f,0x9a,0xf0,0xc2,0xe1,0x90,0x7f,0x97,0xf0,0x71,0xd6,0x71,0xd6,0x90},
111    16,0x340,0,{0x7f,0x9a,0xf0,0xd2,0xe1,0x90,0x7f,0x97,0xf0,0xe5,0x91,0xc2,0xe7,0xf5,0x91,0xd0},
       16,0x350,0,{0xe0,0xd0,0x85,0xd0,0x84,0xd0,0x82,0xd0,0x86,0x32,0xc0,0x86,0xc0,0x82,0xc0,0x84},
       8,0x360,0,{0xc0,0x85,0xc0,0xe0,0x90,0x7f,0x9a,0xe0},
       16,0x368,0,{0xb2,0xe0,0x90,0x7f,0x97,0xf0,0x90,0x7f,0x99,0xe0,0xfc,0x90,0x7f,0x9b,0xe0,0xfd},
       16,0x378,0,{0x90,0x7f,0x9a,0xe0,0xc2,0xe1,0x90,0x7f,0x97,0xf0,0x71,0xd6,0x71,0xd6,0x90,0x7f},
116    16,0x388,0,{0x9a,0xe0,0xc2,0xe2,0x90,0x7f,0x97,0xf0,0xbe,0x00,0x19,0x90,0x7f,0xb8,0xe0,0x20},
       16,0x398,0,{0xe1,0xf9,0xea,0xfb,0x0b,0x90,0x7d,0xff,0xa3,0xdb,0xfd,0xec,0xf0,0xa3,0xed,0xf0},
       16,0x3a8,0,{0x0a,0x0a,0x80,0x17,0x90,0x7f,0xbc,0xe0,0x20,0xe1,0xf9,0xea,0xfb,0x0b,0x90,0x7c},
       16,0x3b8,0,{0xff,0xa3,0xdb,0xfd,0xec,0xf0,0xa3,0xed,0xf0,0x0a,0x0a,0xe5,0x91,0xc2,0xe6,0xf5},
       16,0x3c8,0,{0x91,0xc2,0xea,0xd0,0xe0,0xd0,0x85,0xd0,0x84,0xd0,0x82,0xd0,0x86,0x32,0x00,0x00},
121    16,0x3d8,0,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
       16,0x3e8,0,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x22,0x32},
       16,0x3f8,0,{0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0x32,0xc0,0x86},
       16,0x408,0,{0xc0,0x82,0xc0,0x84,0xc0,0x85,0xc0,0xe0,0x90,0x7c,0xc0,0xe0,0xf9,0xe5,0x91,0xc2},
       16,0x418,0,{0xe4,0xf5,0x91,0x90,0x7f,0xaa,0x74,0x10,0xf0,0x90,0x7f,0xcd,0x74,0x40,0xf0,0xd0},
```

```
126     16,0x428,0,{0xe0,0xd0,0x85,0xd0,0x84,0xd0,0x82,0xd0,0x86,0x32,0x32,0x32,0x32,0xc0,0x86,0xc0},
        16,0x438,0,{0x82,0xc0,0x84,0xc0,0x85,0xc0,0xe0,0x90,0x7b,0xc0,0xe0,0xb4,0x10,0x13,0xd2,0xab},
        16,0x448,0,{0xd2,0xa9,0xd2,0xeb,0x75,0x89,0x00,0x75,0x8d,0x00,0x75,0x8b,0x00,0xd2,0x8e,0xc2},
        16,0x458,0,{0x8f,0xe5,0x91,0xc2,0xe4,0xf5,0x91,0x90,0x7f,0xaa,0x74,0x40,0xf0,0x90,0x7f,0xd1},
        16,0x468,0,{0x74,0x40,0xf0,0xd0,0xe0,0xd0,0x85,0xd0,0x84,0xd0,0x82,0xd0,0x86,0x32,0x32,0x32},
131     1,0x7F92,0,{0x0}};

    int next= 0;
    int samplesdisplayed=2; // variable for drawing data on screen at diff. freq.

136 double voltagescale1=1; // variable for adjusting full—scale voltage of CH1
    double voltagescale2=1; // variable for adjusting full—scale voltage of CH2

    // Function prototypes:

141 int f(int x);   // calculates the corresponding point on screen for CH1 data
    int g(int x);   // calculates the corresponding point on screen for CH2 data
    int WindowCreate(void); // creates main window
    void menubar(HWND hWnd);        // creates menu of the window
    int WaitMessages(void); // waits for messages sent to window
146 void anchordownload(void);      // downloads firmware into Ez—USB RAM
    void BeginReadProc(void);       // initialize USB connection and allocate buffers
    void WriteOutProc (BYTE command);       // sends commands to Ez—USB
    void ChangeSamplingRate (BYTE rate);    // sends the new sampling rate info to Ez—USB
    ULONG __stdcall READ_ISO_BUFFER(LPVOID lpParameter);    // reads data from device
151 void EndReadProc(void);         // clears pipes and buffers
    BOOLEAN bOpenDriver (HANDLE * phDeviceHandle, PCHAR devname); // Opens Ez—USB driver


    LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
156
    // Main function of this Win32 app
    INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, INT iCmdShow)
       {

161    WindowCreate();
       menubar(hWnd);
       WaitMessages();
       return 0;
       }
166

    // create and customize window

    int WindowCreate (void)
171 {
           wndclass.style = CS_HREDRAW | CS_VREDRAW;
       wndclass.lpfnWndProc = WndProc;
       wndclass.cbClsExtra = 0;
       wndclass.cbWndExtra = 0;
176    wndclass.hInstance = hInstance;
       wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
       wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
       wndclass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
       wndclass.lpszMenuName= NULL;
181    wndclass.lpszClassName = szAppName;
       if (! RegisterClass(&wndclass))
          {
          MessageBox(NULL, "Could not create window.", "Error", 0);
          return 0;
186       }

       hWnd = CreateWindow(szAppName, szWndName, WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,   CW_USEDEFAULT, 580, 400, NULL, NULL, hInstance, NULL);

191    for (int i=0; i<1024; i++)
       {
              buffer[i]=32768;
```

```
            buffer2[i]=32768;
    }

    ShowWindow(hWnd, SW_SHOWMAXIMIZED);
    UpdateWindow(hWnd);

    return 0;
}

// create and customize menu bar

void menubar(HWND hWnd)
{
    /* create the menus */
    hMenu     = CreateMenu();
    hFileMenu = CreateMenu();
    hOptionsMenu = CreateMenu();
        hSamplingRate = CreateMenu();
        hVoltageScale = CreateMenu();
        hVoltageScale1 = CreateMenu();
        hVoltageScale2 = CreateMenu();

    /* fill up the file menu */
    item.cbSize      = sizeof(MENUITEMINFO);
    item.fMask       = MIIM_ID | MIIM_TYPE | MIIM_SUBMENU | MIIM_CHECKMARKS | MIIM_STATE;
    item.fType       = MFT_STRING;
    item.hSubMenu    = NULL;

        item.wID        = 'd';
    item.dwTypeData = "D&ownload Firmware";
    item.cch        = strlen("D&ownload Firmware");
    InsertMenuItem(hFileMenu, 0, FALSE, &item);

    item.wID        = 'x';
    item.dwTypeData = "E&xit";
    item.cch        = strlen("E&xit");
    InsertMenuItem(hFileMenu, 0, FALSE, &item);

        /* now do the sampling rate menu */

        item.wID        = '1';
    item.dwTypeData = "f=250 Hz";
    item.cch        = strlen("f=250 Hz");
    InsertMenuItem(hSamplingRate, 0, FALSE, &item);

        item.wID        = '2';
    item.dwTypeData = "f=500 Hz";
    item.cch        = strlen("f=500 Hz");
    InsertMenuItem(hSamplingRate, 1, FALSE, &item);

    item.wID        = '3';
    item.dwTypeData = "f=1 Khz";
    item.cch        = strlen("f=1 KHz");
    InsertMenuItem(hSamplingRate, 2, FALSE, &item);

        item.wID        = '4';
    item.dwTypeData = "f=2 Khz";
    item.cch        = strlen("f=2 KHz");
    InsertMenuItem(hSamplingRate, 3, FALSE, &item);

        /* now do the CH1 voltage scale menu */

        item.wID        = '5';
    item.dwTypeData = "-5V to 5V";
    item.cch        = strlen("-5V to 5V");
    InsertMenuItem(hVoltageScale1, 0, FALSE, &item);

        item.wID        = '6';
```

```
        item.dwTypeData = "—2V to 2V";
        item.cch        = strlen("—2V to 2V");
        InsertMenuItem(hVoltageScale1, 1, FALSE, &item);

266     item.wID        = '7';
        item.dwTypeData = "—1V to 1V";
        item.cch        = strlen("—1V to 1V");
        InsertMenuItem(hVoltageScale1, 2, FALSE, &item);

271         item.wID        = '8';
        item.dwTypeData = "—500mV to 500mV";
        item.cch        = strlen("—500mV to 500mV");
        InsertMenuItem(hVoltageScale1, 3, FALSE, &item);

276         item.wID        = '9';
        item.dwTypeData = "—100mV to 100mV";
        item.cch        = strlen("—100mV to 100mV");
        InsertMenuItem(hVoltageScale1, 4, FALSE, &item);

281         /* now do the CH2 voltage scale menu */

            item.wID        = '10';
        item.dwTypeData = "—5V to 5V";
        item.cch        = strlen("—5V to 5V");
286     InsertMenuItem(hVoltageScale2, 0, FALSE, &item);

            item.wID        = '11';
        item.dwTypeData = "—2V to 2V";
        item.cch        = strlen("—2V to 2V");
291     InsertMenuItem(hVoltageScale2, 1, FALSE, &item);

        item.wID        = '12';
        item.dwTypeData = "—1V to 1V";
        item.cch        = strlen("—1v to 1v");
296     InsertMenuItem(hVoltageScale2, 2, FALSE, &item);

            item.wID        = '13';
        item.dwTypeData = "—500mV to 500mV";
        item.cch        = strlen("—500mV to 500mV");
301     InsertMenuItem(hVoltageScale2, 3, FALSE, &item);

            item.wID        = '14';
        item.dwTypeData = "—100mV to 100mV";
        item.cch        = strlen("—100mV to 100mV");
306     InsertMenuItem(hVoltageScale2, 4, FALSE, &item);

        /* now do the options menu */

            item.wID        = 'r';
311     item.dwTypeData = "Start/Stop &Reading";
        item.cch        = strlen("Start/Stop &Reading");
        InsertMenuItem(hOptionsMenu, 0, FALSE, &item);

            item.wID        = 's';
316     item.dwTypeData = "Start/Stop R&ecording";
        item.cch        = strlen("Start/Stop R&ecording");
        InsertMenuItem(hOptionsMenu, 1, FALSE, &item);

            item.wID        = 'c';
321     item.dwTypeData = "&Change Sampling Rate";
        item.cch        = strlen("&Change Sampling Rate");
        item.hSubMenu   = hSamplingRate;
        InsertMenuItem(hOptionsMenu, 2, FALSE, &item);

326         item.wID        = '15';
        item.dwTypeData = "&CH1";
        item.cch        = strlen("&CH1");
        item.hSubMenu   = hVoltageScale1;
```

```
        InsertMenuItem(hVoltageScale, 0, FALSE, &item);
331
            item.wID        = '16';
        item.dwTypeData = "&CH2";
        item.cch        = strlen("&CH2");
        item.hSubMenu   = hVoltageScale2;
336     InsertMenuItem(hVoltageScale, 1, FALSE, &item);

            item.wID        = 'b';
        item.dwTypeData = "&Change Full-Scale Voltage Range";
        item.cch        = strlen("&Change Full-Scale Voltage Range");
341     item.hSubMenu   = hVoltageScale;
        InsertMenuItem(hOptionsMenu, 3, FALSE, &item);

        /* now do the main menu */
        item.wID        = 0;
346     item.dwTypeData = "&File";
        item.cch        = strlen("&File");
        item.hSubMenu   = hFileMenu;
        InsertMenuItem(hMenu, 0, FALSE, &item);
        item.wID        = 0;
351     item.dwTypeData = "&Options";
        item.cch        = strlen("&Options");
        item.hSubMenu   = hOptionsMenu;
        InsertMenuItem(hMenu, 1, FALSE, &item);

356     /* attach the menu to the window */
        SetMenu(hWnd, hMenu);

        /* use the draw menu as a popup menu */
        hPopup = hOptionsMenu;
361
            CheckMenuItem(hSamplingRate,2,MF_BYPOSITION | MF_CHECKED);
            CheckMenuItem(hVoltageScale1,0,MF_BYPOSITION | MF_CHECKED);
            CheckMenuItem(hVoltageScale2,0,MF_BYPOSITION | MF_CHECKED);
            DrawMenuBar(hWnd);
366
    }

    // this function downloads anchor to Ez-USB RAM
    void anchordownload(void)
371 {
            DWORD nBytesanc;
            char tempout[500];

            // Open Driver To Download Firmware
376         if (!bOpenDriver (&hDevice, pcDriverName))
       {
            MessageBox(hWnd,"Data Analysis - Failed to Open Driver. Cannot download firmware",NULL,NULL);
        hDevice = NULL;
        return;
381     }

            int i;
            for (i=0; i<54; i++)
            {
386
                    ANCHOR_DOWNLOAD_CONTROL anchorload;
                    anchorload.Offset=anchor[i].Address;

                    BOOLEAN bResult=DeviceIoControl (hDevice,
391                      IOCTL_EZUSB_ANCHOR_DOWNLOAD,
                         &anchorload,
                         sizeof(ANCHOR_DOWNLOAD_CONTROL),
                         anchor[i].Data,
                         anchor[i].Length,
396                      &nBytesanc,
                         NULL);
```

```
                if (bResult!= TRUE)
                {
401             MessageBox(hWnd,"Data Analysis — Failed to Download Firmware. Cannot write",NULL,NULL);
                hDevice = NULL;
            return;
                }
        }
406     sprintf(tempout,"Downloaded firmware into Ez—USB RAM");
        SetWindowText(hWnd,tempout);
        hDevice=NULL;
        return;
}
411

// this function waits for messages sent to window

int WaitMessages(void)
416 {
        while (GetMessage(&msg, NULL, 0, 0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
421  }

   return (int) msg.wParam;
}

426
void CALLBACK TimeProc(UINT uID, UINT uMsg, DWORD dwUser, DWORD dw1, DWORD dw2)
{
        return;
}
431



// This function processes the various messages for the window.
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
436   {

        BYTE command=16;
        POINT point;
        HDC hDC;
441     PAINTSTRUCT ps;
        HPEN hOldPen, hNewPen;
        int x;
        int y;

446   switch (message)
        {
        case WM_CREATE:
          return 0;

451     case WM_PAINT:

          hDC = BeginPaint(hWnd, &ps);
          hNewPen = CreatePen(PS_SOLID, 1, RGB(0, 180, 180));   // aqua color (mixes green and blue)
          hOldPen = (HPEN)SelectObject(hDC, hNewPen);
456
          // Print a message:
                  TextOut(hDC, axisx1—30, axisy1—25—128, "ECG Signal", 10);
                  TextOut(hDC, axisx2—45, axisy2—25—128, "Respiratory Signal", 18);

461       // Draw the axes:
          MoveToEx(hDC, 0, axisy1, NULL);
          LineTo(hDC,1024, axisy1);
          MoveToEx(hDC, axisx1, axisy1—128, NULL);
          LineTo(hDC, axisx1, axisy1+128);
```

```
466
                MoveToEx(hDC, 0, axisy2, NULL);
            LineTo(hDC,1024, axisy2);
            MoveToEx(hDC, axisx2, axisy2—128, NULL);
            LineTo(hDC, axisx2, axisy2+128);
471

                // Draw the grid:
                for (x=0; x<1024; x+=8)
                {
476                     for (y=axisy1—128; y<=axisy1+128; y+=8)
                            SetPixel(hDC,x,y, RGB(0, 180, 180));
                        for (y=axisy2—128; y<=axisy2+128; y+=8)
                            SetPixel(hDC,x,y, RGB(0, 180, 180));
                }
481


               // Draw the graph of data on channels:
                hNewPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255)); //blue color
486         hOldPen = (HPEN)SelectObject(hDC, hNewPen);
            MoveToEx(hDC, 0, axisy1 — f(0),NULL);
                for (x = 1; x <1024; x++)
                {
                        LineTo(hDC, x, axisy1 — f(x));
491                     MoveToEx(hDC, x, axisy1 — f(x), NULL);
                }

                MoveToEx(hDC, 0, axisy2 — g(0),NULL);
                for (x = 1; x <1024; x++)
496             {
                        LineTo(hDC, x, axisy2 — g(x));
                        MoveToEx(hDC, x, axisy2 — g(x), NULL);
                }

501         EndPaint(hWnd, &ps);
            return 0;

          case WM_DESTROY:
            PostQuitMessage(0);
506         return 0;

             case WM_COMMAND:

                    switch(LOWORD(wParam))
511                 {
                        case '1':                               // sampling rate f=250 Hz
                            command=7;
                            samplesdisplayed=8;
                            ChangeSamplingRate (command);
516                         BeginReadProc();
                            CheckMenuItem(hSamplingRate,0,MF_BYPOSITION | MF_CHECKED);
                            CheckMenuItem(hSamplingRate,1,MF_BYPOSITION | MF_UNCHECKED);
                            CheckMenuItem(hSamplingRate,2,MF_BYPOSITION | MF_UNCHECKED);
                            CheckMenuItem(hSamplingRate,3,MF_BYPOSITION | MF_UNCHECKED);
521                         DrawMenuBar(hWnd);
                            SetWindowText(hWnd,"Receiving Data");

                        return 0;

526                     case '2':
                            command=132;            // sampling rate f=500 Hz
                            samplesdisplayed=4;
                            ChangeSamplingRate (command);
                            BeginReadProc();
531                         CheckMenuItem(hSamplingRate,0,MF_BYPOSITION | MF_UNCHECKED);
                            CheckMenuItem(hSamplingRate,1,MF_BYPOSITION | MF_CHECKED);
                            CheckMenuItem(hSamplingRate,2,MF_BYPOSITION | MF_UNCHECKED);
```

```
                CheckMenuItem(hSamplingRate,3,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
                SetWindowText(hWnd,"Receiving Data");
            return 0;

            case '3':
                command=195;            // sampling rate f=1 Khz
                samplesdisplayed=2;
                ChangeSamplingRate (command);
                BeginReadProc();
                CheckMenuItem(hSamplingRate,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hSamplingRate,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hSamplingRate,2,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hSamplingRate,3,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
                SetWindowText(hWnd,"Receiving Data");
            return 0;

            case '4':
                command=226;            // sampling rate f=2 Khz
                samplesdisplayed=1;
                ChangeSamplingRate (command);
                BeginReadProc();
                CheckMenuItem(hSamplingRate,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hSamplingRate,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hSamplingRate,2,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hSamplingRate,3,MF_BYPOSITION | MF_CHECKED);
                DrawMenuBar(hWnd);
                SetWindowText(hWnd,"Receiving Data");

            return 0;

            case '5':
                voltagescale1=1.0;      // CH1 5V full—voltage scale
                CheckMenuItem(hVoltageScale1,0,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hVoltageScale1,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,2,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

            case '6':
                voltagescale1=2.5;      // CH1 2V full—voltage scale
                CheckMenuItem(hVoltageScale1,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,1,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hVoltageScale1,2,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

            case '7':
                voltagescale1=5.0;      // CH1 1V full—voltage scale
                CheckMenuItem(hVoltageScale1,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,2,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hVoltageScale1,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

            case '8':
                voltagescale1=10.0;     // CH1 500mV full—voltage scale
                CheckMenuItem(hVoltageScale1,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,2,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,3,MF_BYPOSITION | MF_CHECKED);
```

```
                CheckMenuItem(hVoltageScale1,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

606         case '9':
                voltagescale1=50.0;     // CH1 100mV full-voltage scale
                CheckMenuItem(hVoltageScale1,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,2,MF_BYPOSITION | MF_UNCHECKED);
611             CheckMenuItem(hVoltageScale1,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale1,4,MF_BYPOSITION | MF_CHECKED);
                DrawMenuBar(hWnd);
            return 0;

616         case '10':
                voltagescale2=1.0;      // CH2 5V full-voltage scale
                CheckMenuItem(hVoltageScale2,0,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hVoltageScale2,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,2,MF_BYPOSITION | MF_UNCHECKED);
621             CheckMenuItem(hVoltageScale2,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

626         case '11':
                voltagescale2=2.5;      // CH2 2V full-voltage scale
                CheckMenuItem(hVoltageScale2,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,1,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hVoltageScale2,2,MF_BYPOSITION | MF_UNCHECKED);
631             CheckMenuItem(hVoltageScale2,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

636         case '12':
                voltagescale2=5.0;      // CH2 1V full-voltage scale
                CheckMenuItem(hVoltageScale2,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,2,MF_BYPOSITION | MF_CHECKED);
641             CheckMenuItem(hVoltageScale2,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

646         case '13':
                voltagescale2=10.0;     // CH2 500mV full-voltage scale
                CheckMenuItem(hVoltageScale2,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,2,MF_BYPOSITION | MF_UNCHECKED);
651             CheckMenuItem(hVoltageScale2,3,MF_BYPOSITION | MF_CHECKED);
                CheckMenuItem(hVoltageScale2,4,MF_BYPOSITION | MF_UNCHECKED);
                DrawMenuBar(hWnd);
            return 0;

656         case '14':
                voltagescale2=50.0;     // CH2 100mV full-voltage scale
                CheckMenuItem(hVoltageScale2,0,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,1,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,2,MF_BYPOSITION | MF_UNCHECKED);
661             CheckMenuItem(hVoltageScale2,3,MF_BYPOSITION | MF_UNCHECKED);
                CheckMenuItem(hVoltageScale2,4,MF_BYPOSITION | MF_CHECKED);
                DrawMenuBar(hWnd);
            return 0;

666         case 's':
                if (StopRead==FALSE)
                {
                        if (Rec==FALSE)
```

```
                                        {
671                                             CheckMenuItem(hOptionsMenu,1,MF_BYPOSITION | MF_CHECKED);
                                                DrawMenuBar(hWnd);

                                                recbuffer1=(char*) malloc(10*1024*1024*sizeof(unsigned short)); //record memory
                                                recbuffer2=(char*) malloc(10*1024*1024*sizeof(unsigned short)); //record memory
676                                             reccount=0;
                                                datareceived=0;
                                                output1=fopen("output1.txt","w"); //open output files
                                                output2=fopen("output2.txt","w");
                                                Rec=TRUE;       //set record flag
681                                     }
                                        else
                                        {
                                                Rec=FALSE; //reset record flag
                                                CheckMenuItem(hOptionsMenu,1,MF_BYPOSITION | MF_UNCHECKED);
686                                             DrawMenuBar(hWnd);


                                                fwrite(recbuffer1,1,reccount,output1);
                                                fwrite(recbuffer2,1,reccount,output2);
691                                             fclose(output1); //close output files
                                                fclose(output2);
                                                free(recbuffer1); //free record buffer
                                                free(recbuffer2); //free record buffer
                                        }
696                             }
                                else
                                {
                                        return 0;
                                }
701                     return 0;

                        case 'd':                               // download firmware into device
                                anchordownload();
                        return 0;
706
                        case 'x':                               // exit
                                PostQuitMessage(0);
                        return 0;

711                     case 'r':                               // start/stop reading data
                                if (StopRead==TRUE)
                                {
                                        StopRead=FALSE;
                                        BeginReadProc();
716                                     CheckMenuItem(hOptionsMenu,0,MF_BYPOSITION | MF_CHECKED);
                                        DrawMenuBar(hWnd);
                                        WriteOutProc(command);
                                        DWORD dwThreadId, dwThrdParam = 1;
                                        hRead= CreateThread(
721                                                                             NULL,
                                                                                0,
                                                                                READ_ISO_BUFFER,
                                                                                &dwThrdParam,
                                                                                CREATE_SUSPENDED,
726                                                                             &dwThreadId
                                                                        );
                                        SetThreadPriority(hRead,15);
                                        ResumeThread(hRead);
                                }
731                             else
                                {
                                        StopRead=TRUE;
                                        EndReadProc();
                                        CheckMenuItem(hOptionsMenu,0,MF_BYPOSITION | MF_UNCHECKED);
736                                     DrawMenuBar(hWnd);
                                        SetWindowText(hWnd,"Data Analysis");
```

```
                                        UpdateWindow(hWnd);
                                }
                        return 0;
741                   }
                      return 0;

                    case WM_RBUTTONDOWN:
                            point.x = LOWORD(lParam);
746                         point.y = HIWORD(lParam);
                            ClientToScreen(hWnd, &point);
                            TrackPopupMenu(hPopup, TPM_LEFTALIGN, point.x, point.y,0, hWnd, NULL);
                    return 0;

751           break;

            }

       return DefWindowProc(hWnd, message, wParam, lParam);
756    }


       /* f(x) & g(x)are the functions that calculate the corresponding y value */
       /* for each x-pixel given (0<=x<=1024) */
761
       int f(int x)
       {
               double y;
               char tempbuf[8];
766            sprintf(tempbuf,"%d",buffer[x]);
               y= atoi(tempbuf)*1.0;
               y=(y/65536.0-0.5)*512*voltagescale1;
               if (y>128)
               {
771                    y=128;
               }
               if (y<-128)
               {
                      y=-128;
776            }
               return (int)y;
       }


781    int g(int x)
       {
               double y;
               char tempbuf[8];
               sprintf(tempbuf,"%d",buffer2[x]);
786            y= atoi(tempbuf)*1.0;
               y=(y/65536.0-0.5)*512*voltagescale2;
               if (y>128)
               {
                      y=128;
791            }
               if (y<-128)
               {
                      y=-128;
               }
796
               return y;
       }


801    /* BeginReadProc initializes USB connection... */
       /* opens driver, sets interface, allocates memory for the read I/O buffer */

       void BeginReadProc (void)
       {
```

```
806     if (bOpenDriver (&hDevice, pcDriverName) == TRUE)
        {
         SetWindowText(hWnd,"Data Analysis — Opened Driver Successfully to Read");
        }
        else
811     {
             MessageBox(hWnd,"Data Analysis — Failed to Open Driver. Cannot read",NULL,NULL);
             StopRead=TRUE;
         hDevice=NULL;
         return;
816     }

        SET_INTERFACE_IN setint;
        setint.interfaceNum=Interface;
        setint.alternateSetting=AltSetting;
821     DWORD nBytesint;

        BOOLEAN bResult=DeviceIoControl (hDevice,    // Set Ez—USB interface
                         IOCTL_Ezusb_SETINTERFACE,
                         &setint,
826                      sizeof(SET_INTERFACE_IN),
                         NULL,
                         0,
                         &nBytesint,
                         NULL);
831     if (bResult!= TRUE)
        {
             MessageBox(hWnd,"Data Analysis — Failed to Set Interface. Cannot read",NULL,NULL);
         hDevice=NULL;
         return;
836     }


        // allocate reading buffers
        ULONG bytesToRead = PackCount * (PackSize + sizeof(USBD_ISO_PACKET_DESCRIPTOR));
841     isobuf= (BYTE*) malloc(bytesToRead);
        isobuf2=(PULONG) malloc(sizeof(ULONG));

        datareceived=0;
        reccount=0;
846
        if (!isobuf)
        {
           MessageBox(hWnd,"Data Analysis — Memory Allocation Failed",NULL,NULL);
           hDevice=NULL;
851        return;
        }

        InvalidateRect(hWnd, NULL, TRUE);
    }
856

    /* This function switches between two bulk pipes,
            reset the selected pipe and read 64—byte packet,
            check for record flag and write to rec buffer if set,
861         and update window while stopRead flag is false.
    */

    ULONG __stdcall READ_ISO_BUFFER (LPVOID lpParameter)
    {
866         if (hDevice==NULL)
            {
                    return 0;
            }

871     SetWindowText(hWnd,"Receiving Data");

        BOOLEAN bResult= FALSE;
```

```
        int i,j;
        char temp[256];      // buffer used to write text on the window bar
876     int count=0; // counter for updating graphs
        RECT region1={{0},{40},{1024},{296}};        // first graph region
        RECT region2={{0},{424},{1024},{680}};       // second graph region
        BYTE bytesToRead= PackSize;

881     while (!StopRead)
        {

                    ULONG pipenum;

886                 BULK_TRANSFER_CONTROL  BulkControl;

                    //change the pipe to read from
                    if (PipeSelect)
                    {
891                         BulkControl.pipeNum=PipeNumber1;
                            pipenum=PipeNumber1;
                            PipeSelect=FALSE;
                    }
                    else
896                 {
                            BulkControl.pipeNum=PipeNumber2;
                            pipenum=PipeNumber2;
                            PipeSelect=TRUE;
                    }
901
               // reset the pipe
                    DWORD nBytes;
                    DWORD nBytespipe;

906             bResult = DeviceIoControl (hDevice,
                        IOCTL_Ezusb_RESETPIPE,
                        &pipenum,
                        sizeof(ULONG),
                        NULL,
911                     0,
                        &nBytespipe,
                        NULL);
                    bResult=TRUE;
               if (bResult != TRUE)
916            {
                  MessageBox(hWnd,"Data Analysis — Pipe Reset Failed",NULL,NULL);
                  hDevice=NULL;
                            return 0;
               }
921
                // read 64—byte packet by bulk transfer

            bResult = DeviceIoControl (hDevice,
                        IOCTL_EZUSB_BULK_READ,
926                     &BulkControl,
                        sizeof(BULK_TRANSFER_CONTROL),
                        isobuf,
                        bytesToRead,
                        &nBytes,
931                     NULL);

                next= 1024 — (samplesdisplayed);

            if (bResult != TRUE)
936         {
                        DeviceIoControl (hDevice,
                                                  IOCTL_EZUSB_GET_LAST_ERROR,
                                                 NULL,
                                                 0,
941                                              isobuf2,
```

```
                                                sizeof(ULONG),
                                                &nBytes,
                                                NULL);
                        char tempbuf[100];
946                     sprintf(tempbuf,"Read Bulk Data Failed with Error %d",isobuf2);
                        SetWindowText(hWnd,tempbuf);
                return 0;
        }
            else
951         {
                    if (Rec==TRUE) // if record flag is set, save data to disk
                    {
                            for (i=0; i<PackSize/4; i++)
                            {
956                         union v {unsigned short ni; BYTE d[sizeof(unsigned short)];};
                            union v v1;
                            v1.d[1]=isobuf[4*i];
                            v1.d[0]=isobuf[4*i+1];
                            union v v2;
961                         v2.d[1]=isobuf[4*i+2];
                            v2.d[0]=isobuf[4*i+3];
                            sprintf(&recbuffer1[reccount],"%5d\n",v1.ni);
                            sprintf(&recbuffer2[reccount],"%5d\n",v2.ni);
                            reccount=reccount+6;
966                         }

                            datareceived=datareceived+bytesToRead/4;
                            if (datareceived>1048576)
                            {
971                             sprintf(temp,"Recorded %6.1fM samples/channel",datareceived/1048576.0);
                            }
                            else if (datareceived>1024)
                            {
                                    sprintf(temp,"Recorded %6.1fK samples/channel",datareceived/1024.0);
976                         }
                            else
                            {
                                    sprintf(temp,"Recorded %5.0f samples/channel",datareceived);
                            }
981
                            SetWindowText(hWnd,temp);
                            if (reccount>=180000)
                            {
                                    fwrite(recbuffer1,1,reccount,output1);
986                                 fwrite(recbuffer2,1,reccount,output2);
                                    reccount=0;
                            }
                    }
                    else
991                 {
                            SetWindowText(hWnd,"Receiving Data");
                    }

                    for (i=0; i<next; i++)
996                 {
                            buffer[i]=buffer[i+(samplesdisplayed)];
                            buffer2[i]=buffer2[i+(samplesdisplayed)];
                    }

1001                j=(PackSize/4)/samplesdisplayed;

                    for (i=0; i<(samplesdisplayed); i++)
                    {
                            union v {unsigned short ni; BYTE d[sizeof(unsigned short)];};
1006                        union v vn;
                            vn.d[1]=isobuf[4*j*i];
                            vn.d[0]=isobuf[4*j*i+1];
                            union v vm;
```

```
                        vm.d[1]=isobuf[4*j*i+2];
                        vm.d[0]=isobuf[4*j*i+3];
                        buffer[next+i]=vn.ni;
                        buffer2[next+i]=vm.ni;
                }

        }

        if (samplesdisplayed==8)
        {

                if (count==1)
                {
                      InvalidateRect(hWnd, &region1, TRUE);
                      UpdateWindow(hWnd);
                }
                if (count==2)
                {
                        InvalidateRect(hWnd, &region2, TRUE);
                        UpdateWindow(hWnd);
                        count=0;
                }
                else
                count=count+1;
                if (count>2)
                {
                        count=0;
                }
        }

        if (samplesdisplayed==4)
        {

                if (count==2)
                {
                      InvalidateRect(hWnd, &region1, TRUE);
                      UpdateWindow(hWnd);
                }
                if (count==4)
                {
                        InvalidateRect(hWnd, &region2, TRUE);
                        UpdateWindow(hWnd);
                        count=0;
                }
                else
                count=count+1;
                if (count>4)
                {
                        count=0;
                }
        }

        if (samplesdisplayed==2)
        {

                if (count==4)
                {
                      InvalidateRect(hWnd, &region1, TRUE);
                      UpdateWindow(hWnd);
                }

                if (count==8)
                {
                        InvalidateRect(hWnd, &region2, TRUE);
                        UpdateWindow(hWnd);
                        count=0;
                }
                else
```

```
                    count=count+1;
                    if (count>8)
                    {
1081                        count=0;
                    }
            }

            if (samplesdisplayed==1)
1086        {

                    if (count==8)
                    {
                        InvalidateRect(hWnd, &region1, TRUE);
1091                    UpdateWindow(hWnd);
                    }

                    if (count==16)
                    {
1096                    InvalidateRect(hWnd, &region2, TRUE);
                        UpdateWindow(hWnd);
                        count=0;
                    }
                    else
1101                count=count+1;
            }

        }
            return 0;
1106    }

    // function that ends the reading process and clear pending I/O
    void EndReadProc(void)
    {
1111        if (hDevice==NULL)
            {
                return;
            }

1116        DWORD nBytespipe;
            ULONG pipenum=PipeNumber1;

            BOOL bResult = DeviceIoControl (hDevice,
                        IOCTL_Ezusb_ABORTPIPE,
1121                    &pipenum,
                        sizeof(ULONG),
                        NULL,
                        0,
                        &nBytespipe,
1126                    NULL);
            if (bResult != TRUE)
            {
            MessageBox(hWnd,"Data Analysis — Pipe Abort Failed",NULL,NULL);
            hDevice=NULL;
1131            return;
         }

            pipenum=PipeNumber2;
            bResult = DeviceIoControl (hDevice,
1136                    IOCTL_Ezusb_ABORTPIPE,
                        &pipenum,
                        sizeof(ULONG),
                        NULL,
                        0,
1141                    &nBytespipe,
                        NULL);
            if (bResult != TRUE)
            {
            MessageBox(hWnd,"Data Analysis — Pipe Abort Failed",NULL,NULL);
```

```
1146          hDevice=NULL;
                      return;
          }

              free(isobuf);
1151          free(isobuf2);
              hDevice=NULL;
              return;
      }

1156
      // Sends command to Ez—USB
      void WriteOutProc (BYTE command)
      {
              if (hDevice==NULL)
1161          {
                      return;
              }

              BYTE* writebuf;
1166          BOOLEAN bResult= FALSE;

              BYTE bytesToWrite=1;
              writebuf=(BYTE*) malloc(bytesToWrite);

1171          ULONG pipenum;
              BULK_TRANSFER_CONTROL   BulkControl;

              pipenum=PipeOut;

1176          BulkControl.pipeNum=pipenum;

          // reset the pipe
              DWORD nBytes;
              DWORD nBytespipe;
1181
          bResult = DeviceIoControl (hDevice,
                                              IOCTL_Ezusb_RESETPIPE,
                          &pipenum,
                          sizeof(ULONG),
1186                      NULL,
                          0,
                          &nBytespipe,
                          NULL);

1191      if (bResult != TRUE)
          {
              MessageBox(hWnd,"Send Command — Pipe Reset Failed",NULL,NULL);
              hDevice=NULL;
                      return;
1196          }

              writebuf[0]=command;

              bResult = DeviceIoControl (hDevice,
1201                        IOCTL_EZUSB_BULK_WRITE,
                          &BulkControl,
                          sizeof(BULK_TRANSFER_CONTROL),
                          writebuf,
                          bytesToWrite,
1206                      &nBytes,
                          NULL);
              if (!bResult)
              {
                      hDevice=NULL;
1211                  return;
              }
```

```
            bResult = DeviceIoControl (hDevice,
                        IOCTL_Ezusb_ABORTPIPE,
1216                    &pipenum,
                        sizeof(ULONG),
                        NULL,
                        0,
                        &nBytespipe,
1221                    NULL);
            if (!bResult)
            {
            MessageBox(hWnd,"Send Command — Pipe Abort Failed",NULL,NULL);
            hDevice=NULL;
1226                return;
        }

            free(writebuf);

1231        return;
    }

    // Changes sampling rate of Ez—USB
    void ChangeSamplingRate (BYTE rate)
1236 {
            if (hDevice==NULL)
            {
                return;
            }
1241
            BYTE* writebuf;
            BOOLEAN bResult= FALSE;

            BYTE bytesToWrite=1;
1246        writebuf=(BYTE*) malloc(bytesToWrite);

            ULONG pipenum;
            BULK_TRANSFER_CONTROL   BulkControl;

1251        pipenum=SamplingPipe;

            BulkControl.pipeNum=pipenum;

        // reset the pipe
1256        DWORD nBytes;
            DWORD nBytespipe;

        bResult = DeviceIoControl (hDevice,
                                        IOCTL_Ezusb_RESETPIPE,
1261                    &pipenum,
                        sizeof(ULONG),
                        NULL,
                        0,
                        &nBytespipe,
1266                    NULL);

        if (bResult != TRUE)
        {
            MessageBox(hWnd,"Change Sampling Rate — Pipe Reset Failed",NULL,NULL);
1271        hDevice=NULL;
                return;
            }

            writebuf[0]=rate;
1276
            bResult = DeviceIoControl (hDevice,
                        IOCTL_EZUSB_BULK_WRITE,
                        &BulkControl,
                        sizeof(BULK_TRANSFER_CONTROL),
1281                    writebuf,
```

```
                        bytesToWrite,
                        &nBytes,
                        NULL);

1286          bResult = DeviceIoControl (hDevice,
                        IOCTL_Ezusb_ABORTPIPE,
                        &pipenum,
                        sizeof(ULONG),
                        NULL,
1291                    0,
                        &nBytespipe,
                        NULL);
              if (bResult != TRUE)
              {
1296          MessageBox(hWnd,"Data Analysis — Pipe Abort Failed",NULL,NULL);
              hDevice=NULL;
                      return;
          }

1301          free(writebuf);
              return;
      }


1306

      // Open the Driver
      BOOLEAN bOpenDriver (HANDLE * phDeviceHandle, PCHAR devname)
      {
1311     char completeDeviceName[64] = "";
         char pcMsg[64] = "";

         strcat (completeDeviceName,
            "\\\\.\\"
1316        );

         strcat (completeDeviceName,
                         devname
               );
1321
         *phDeviceHandle = CreateFile(   completeDeviceName,
            GENERIC_WRITE,
            FILE_SHARE_WRITE,
            NULL,
1326        OPEN_EXISTING,
            0,
            NULL);

         if (*phDeviceHandle == INVALID_HANDLE_VALUE) {
1331        return (FALSE);
         } else {
            return (TRUE);
         }

1336
      }//OpenDevice
```

## B.4 The Digital Signal Processing Software

Listing B.3: Software in Matlab M-File format

```
close all;
clear all;
clc;

fs=1000;

ecg = textread('tunca91.txt');
ecg=ecg/65536*20-10;
ecg1=ecg;

res = textread('tunca92.txt');
res=res/65536*20-10;
res1=res;

tinit=0;
tfinal=(size(ecg,1)-1)/1000;
t=linspace(tinit,tfinal,size(ecg,1));
t1=t;

figure(1);subplot(4,1,1);plot(t,res); ylabel('Raw Resp');title('Respiratory Filtering Process')
axis([20 30 min(res) max(res)]);
figure(2);subplot(3,1,1);plot(t,ecg); ylabel('Raw ECG');title('ECG Filtering Process');
axis([23 25 min(ecg) max(ecg)]);

hpecgn=fs/2;
hpecg=ones(1,hpecgn);
hpecg=-hpecg/hpecgn;
hpecg(hpecgn/2+1)=1+hpecg(hpecgn/2+1);
ecg = filter(hpecg,[1],ecg);
ecg = ecg((1.5*size(hpecg,2)+1):size(ecg,1));

hpresn=10*fs;
hpres=ones(1,hpresn);
hpres=-hpres/hpresn;
hpres(hpresn/2+1)=1+hpres(hpresn/2+1);
res = filter(hpres,[1],res);
res = res((1.5*size(hpres,2)+1):size(res,1));

t = t((size(hpres,2)+1):(size(t,2)-0.5*size(hpres,2)));
ecg = ecg((size(hpres,2)-size(hpecg,2)+1):(size(ecg,1)-0.5*size(hpres,2)+0.5*size(hpecg,2)));

figure(1);subplot(4,1,2);plot(t,res); ylabel('Highpass filtered');
axis([20 30 min(res) max(res)]);
figure(2);subplot(3,1,2);plot(t,ecg); ylabel('Highpass filtered');
axis([23 25 min(ecg) max(ecg)]);

nn= fs/50;
n=ones(1,nn);
n=n/nn;
ecg = filter(n,[1],ecg);
res = filter(n,[1],res);
ecg = ecg((1.5*size(n,2)+1):size(ecg,1));
res = res((1.5*size(n,2)+1):size(res,1));
t = t((size(n,2)+1):(size(t,2)-0.5*size(n,2)));
figure(1);subplot(4,1,3);plot(t,res); ylabel('50Hz notch');
axis([20 30 min(res) max(res)]);
figure(2);subplot(3,1,3);plot(t,ecg); ylabel('Lowpass Filtered');xlabel('time (sec)');
axis([23 25 min(ecg) max(ecg)]);
```

```
lpresn=fs/3.125;
lpres=ones(1,lpresn);
lpres=lpres/lpresn;
res = filter(lpres,[1],res);
res = res((1.5*size(lpres,2)+1):size(res,1));
t = t((size(lpres,2)+1):(size(t,2)-0.5*size(lpres,2)));
ecg = ecg((size(lpres,2)+1):(size(ecg,1)-0.5*size(lpres,2)));

figure(1);subplot(4,1,4);plot(t,res); ylabel('Lowpass filtered');xlabel('time (sec)');
axis([20 30 min(res) max(res)]);

%%%%%%%%%%%%%%%%%%%%%% DF1 Method for QRS Detection %%%%%%%%%%%%%%%%%

yfilt=[2 1 0 -1 -2];    % find the first derivative of ECG signal
yfilt=yfilt/10;

ydiff = filter(yfilt,[1],ecg);
ydiff = ydiff((1.5*size(yfilt,2)+1):size(ydiff,1));
tdiff = t((size(yfilt,2)+1):(size(t,2)-0.5*size(yfilt,2)));
ecg = ecg((size(yfilt,2)+1):(size(ecg,1)-0.5*size(yfilt,2)));

segments=floor(size(ydiff,1)/(fs*3));
for i=1:segments
    slopemaxarray(i)=max(ydiff((i-1)*fs*3+1:i*fs*3));   % find maximum values of segments
end
slopeth=0.7*median(slopemaxarray);  % define slope threshold as 0.7 of median of maximums
rr=1;
trlast=0;
for i=3:1:size(ydiff,1)
    if (ydiff(i)>slopeth) & (ydiff(i-1)>slopeth) & (ydiff(i-2)>slopeth) & (tdiff(i-2)>trlast+0.3)
        tr(rr)=tdiff(i-2); % find time of R-waves
        trlast=tdiff(i-2);
        ti(rr)=i-2;
        rr=rr+1;
    end
end

for i=1:size(ti,2)
    rwave(i)=ecg(ti(i));
end

trdiff=diff(tr);    % time intervals between R-waves
tr=tr(2:size(tr,2));    % discard first r-wave
ti=ti(2:size(ti,2));
rwave=rwave(2:size(rwave,2));

tdiff=tdiff(ti(1):ti(size(ti,2)));
ecg=ecg(ti(1):ti(size(ti,2)));
ydiff=ydiff(ti(1):ti(size(ti,2)));

tdiff=tdiff-tr(1);
tr=tr-tr(1);
figure(3); plot(tdiff,ecg,'black'); hold on; plot(tdiff,5*ydiff,'red');stem(tr,rwave,'o--');hold off;
legend('Filtered ECG Signal','First Derivative (x5)','R-Wave');
axis([23.5 24.3 -0.4 1]); grid on; xlabel('time(sec)');ylabel('Amplitude (V)');
thrv = tr(1):0.25:tr(size(tr,2));  % create time axis for interpolated hrv signal
hrv=interp1(tr,trdiff,thrv); % interpolate R-R interval signal

figure(4);plot(thrv,hrv); title('Heart Rate Variability');
xlabel('Time (sec)');ylabel('RR Intervals (sec)'); grid on;

hrv2=hrv-mean(hrv);
hrvmean=mean(hrv),     %mean of HRV signal
hrvstd=std(hrv),    % standard deviation of HRV signal
[P,f]=pcov(hrv2,50,512,4); % power spectral density of HRV signal
```

```
%%%%%%%%%%%%%%%% Respiration Variability detection %%%%%%%%%%%%%%

respfilt=[2 1 0 -1 -2];
respfilt=respfilt/10;

respdiff = filter(respfilt,[1],res);    % first derivative of respiratory signal
respdiff = respdiff((1.5*size(respfilt,2)+1):size(respdiff,1));
tresp = t((size(respfilt,2)+1):(size(t,2)-0.5*size(respfilt,2)));
res = res((size(respfilt,2)+1):(size(res,1)-0.5*size(respfilt,2)));

rc=1;
texlast=0;

respsegments=floor(size(respdiff,1)/(fs*10));

for i=1:respsegments
    respmaxarray(i)=max(respdiff((i-1)*fs*10+1:i*fs*10));
end
respth=0.2*median(respmaxarray);

for i=3:1:size(respdiff,1)
    if (respdiff(i)>respth) & (respdiff(i-1)>respth) & (respdiff(i-2)>respth) & (res(i)<0) & (tresp(i)>texlast+1.7)
        tex(rc)=tresp(i-2); % find time of expiration
        texlast=tresp(i-2);
        ti2(rc)=i-2;
        rc=rc+1;
    end
end

for i=1:size(ti2,2)
    resppoint(i)=res(ti2(i));
end

figure(6);
plot(tresp,res,'black');hold on;plot(tresp,100*respdiff,'red'); stem(tex,resppoint,'o--');hold off;
legend('Respiratory Signal','First Derivative (x100)','Respiratory Fiducial Points');
% axis([50 60 min(res) max(res)]);

texdiff=diff(tex);  % respiration intervals
tex=tex(2:size(tex,2));
tresp=tresp(ti2(2):ti2(size(ti2,2)));
respdiff = respdiff(ti2(2):ti2(size(ti2,2)));
res = res(ti2(2):ti2(size(ti2,2)));

tresp2 = tex(1):0.25:tex(size(tex,2));  % create time axis for interpolated respiration signal
respv2=interp1(tex,texdiff,tresp2);    % interpolate respiration interval function

respv3=respv2-mean(respv2);
respmean=mean(respv2),     %mean of respiratory signal
respstd=std(respv2),    % standard deviation of respiratory signal

figure(5); plot(f,P); axis([0 0.5 min(P) max(P)]); xlabel('frequency (Hz)');
title('Power Spectral Density of HRV'); grid on;
clc;
respmean,
respstd,
hrvmean,
hrvstd,

figure(7); plot(tresp2,respv2);
title('Respiratory Signal Variability'); grid on;
xlabel('Time(sec)');
ylabel('Instantaneous Respiration Period (sec)');

figure(8); hist(respv2,40); grid on;
xlabel('Instantaneous Respiration Period (sec)');
ylabel('Histogram of Respiratory Signal Variability');
```

```
%%%%%%%%%%%%%%%%%% filter responses %%%%%%%%%%%%%%%%%%%%%%%%

% c=linspace(0,1000,64000);
%
% ftha=abs(fft(hpecg,64000));
% fthb=angle(fft(hpecg,64000))/pi;
% ftna=abs(fft(n,64000));
% ftnb=angle(fft(n,64000))/pi;
% ftlresa=abs(fft(lpres,64000));
% ftlresb=angle(fft(lpres,64000));
%
%%% Frequency response of ECG highpass filter %%%%%
% figure;
% subplot(2,1,1);
% plot(c,ftha);
% axis([0 20 0 1.5]);
% xlabel('frequency (Hz)');
% ylabel('|Xhp(jw)|');
% grid on;
% hold on; stem(1.5,0.707,'bo-.'); hold off;
% subplot(2,1,2);
% plot(c,fthb);
% axis([0 20 -1 1]);
% xlabel('frequency (Hz)');
% ylabel('Phase of Xhp(jw) (*pi rad)');
% grid on;
%
%%% Frequency response of 50Hz notch filter %%%%%
% figure;
% subplot(2,1,1);
% plot(c,ftna);
% axis([0 125 0 1]);
% xlabel('frequency (Hz)');
% ylabel('|Xn(jw)|');
% grid on;
% hold on; stem(22.1,0.707,'bo-.'); hold off;
% subplot(2,1,2);
% plot(c,ftnb);
% axis([0 125 -1 1]);
% xlabel('frequency (Hz)');
% ylabel('Phase of Xn(jw) (*pi rad)');
% grid on;
%
%%% Frequency response of respiratory lowpass filter %%%%%
%
% figure;
% subplot(2,1,1);
% plot(c,ftlresa);
% axis([0 20 0 1]);
% xlabel('frequency (Hz)');
% ylabel('|Xlp(jw)|');
% grid on;
% hold on; stem(1.4,0.707,'bo-.'); hold off;
% subplot(2,1,2);
% plot(c,ftlresb);
% axis([0 20 -1 1]);
% xlabel('frequency (Hz)');
% ylabel('Phase of Xlp(jw) (*pi rad)');
% grid on;
```

# Appendix C

# Technical References

This chapter includes technical references for some of the circuit elements used in the design.

## C.1 OP07 Technical Reference

# Ultralow Offset Voltage Operational Amplifiers

# OP07

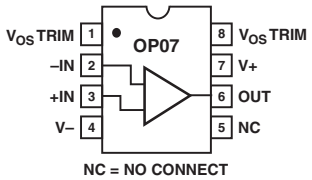
**FEATURES**
**Low $V_{OS}$: 75 μV Max**
**Low $V_{OS}$ Drift: 1.3 μV/°C Max**
**Ultra-Stable vs. Time: 1.5 μV/Month Max**
**Low Noise: 0.6 μV p-p Max**
**Wide Input Voltage Range: ±14 V**
**Wide Supply Voltage Range: 3 V to 18 V**
**Fits 725,108A/308A, 741, AD510 Sockets**
**125°C Temperature-Tested Dice**

**APPLICATIONS**
**Wireless Base Station Control Circuits**
**Optical Network Control Circuits**
**Instrumentation**
**Sensors and Controls**
- **Thermocouples**
- **RTDs**
- **Strain Bridges**
- **Shunt Current Measurements**

**Precision Filters**

**PIN CONNECTIONS**

**Epoxy Mini-Dip (P-Suffix)**
**8-Pin SO (S-Suffix)**




## GENERAL DESCRIPTION

The OP07 has very low input offset voltage (75 μV max for OP07E) which is obtained by trimming at the wafer stage. These low offset voltages generally eliminate any need for external nulling. The OP07 also features low input bias current (±4 nA for OP07E) and high open-loop gain (200 V/mV for OP07E). The low offsets and high open-loop gain make the OP07 particularly useful for high-gain instrumentation applications.

The wide input voltage range of ±13 V minimum combined with high CMRR of 106 dB (OP07E) and high input impedace provides high accuracy in the noninverting circuit configuration. Excellent linearity and gain accuracy can be maintained even at high closed-loop gains. Stability of offsets and gain with time or variations in temperature is excellent. The accuracy and stability of the OP07, even at high gain, combined with the freedom from external nulling have made the OP07 an industry standard for instrumentation applications.

The OP07 is available in two standard performance grades. The OP07E is specified for operation over the 0°C to 70°C range, and OP07C over the –40°C to +85°C temperature range.

The OP07 is available in epoxy 8-lead Mini-DIP and 8-lead SOIC. It is a direct replacement for 725,108A, and OP05 amplifiers; 741-types may be directly replaced by removing the 741's nulling potentiometer. For improved specifications, see the OP177 or OP1177. For ceramic DIP and TO-99 packages and standard micro circuit (SMD) versions, see the OP77.

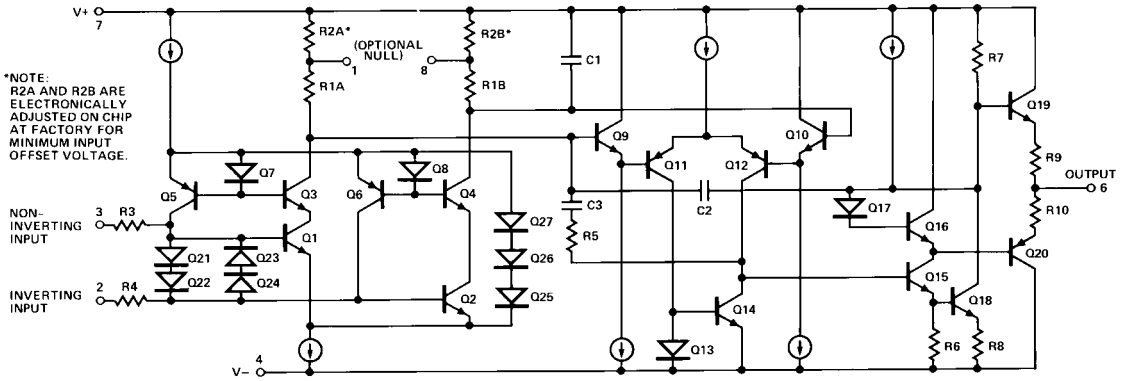



*Figure 1. Simplified Schematic*

REV. A



**One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.**
**Tel: 781/329-4700** **www.analog.com**
**Fax: 781/326-8703** 

# OP07–SPECIFICATIONS

## OP07E ELECTRICAL CHARACTERISTICS ($V_S$ = ±15 V, $T_A$ = 25°C, unless otherwise noted.)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| INPUT CHARACTERISTICS | | | | | | |
| Input Offset Voltage[1] | $V_{OS}$ | | | 30 | 75 | µV |
| Long-Term $V_{OS}$ Stability[2] | $V_{OS}$/Time | | | 0.3 | 1.5 | µV/Mo |
| Input Offset Current | $I_{OS}$ | | | 0.5 | 3.8 | nA |
| Input Bias Current | $I_B$ | | | ±1.2 | ±4.0 | nA |
| Input Noise Voltage | $e_n$ p-p | 0.1 Hz to 10 Hz[3] | | 0.35 | 0.6 | µV p-p |
| Input Noise Voltage Density | $e_n$ | $f_O$ = 10 Hz | | 10.3 | 18.0 | nV√Hz |
| | | $f_O$ = 100 Hz[3] | | 10.0 | 13.0 | nV√Hz |
| | | $f_O$ = 1 kHz | | 9.6 | 11.0 | nV√Hz |
| Input Noise Current | $I_n$ p-p | | | 14 | 30 | pA p-p |
| Input Noise Current Density | $I_n$ | $f_O$ = 10 Hz | | 0.32 | 0.80 | pA√Hz |
| | | $f_O$ = 100 Hz[3] | | 0.14 | 0.23 | pA√Hz |
| | | $f_O$ = 1 kHz | | 0.12 | 0.17 | pA√Hz |
| Input Resistance—Differential Mode[4] | $R_{IN}$ | | 15 | 50 | | mΩ |
| Input Resistance—Common-Mode | $R_{INCM}$ | | | 160 | | GΩ |
| Input Voltage Range | IVR | | ±13 | ±14 | | V |
| Common-Mode Rejection Ratio | CMRR | $V_{CM}$ = ±13 V | 106 | 123 | | dB |
| Power Supply Rejection Ratio | PSRR | $V_S$ = ±3 V to ±18 V | | 5 | 20 | µV/V |
| Large-Signal Voltage Gain | $A_{VO}$ | $R_L$ ≥ 2 kΩ, $V_O$ = ±10 V | 200 | 500 | | V/mV |
| | | $R_L$ ≥ 500 Ω, $V_O$ = ±0.5 V, $V_S$ = ±3 V[4] | 150 | 400 | | V/mV |
| OUTPUT CHARACTERISTICS | | | | | | |
| Output Voltage Swing | $V_O$ | $R_L$ ≥ 10 kΩ | ±12.5 | ±13.0 | | V |
| | | $R_L$ ≥ 2 kΩ | ±12.0 | ±12.8 | | V |
| | | $R_L$ ≥ 1 kΩ | ±10.5 | ±12.0 | | V |
| DYNAMIC PERFORMANCE | | | | | | |
| Slew Rate | SR | $R_L$ ≥ 2 kΩ[3] | 0.1 | 0.3 | | V/µs |
| Closed-Loop Bandwidth | BW | $A_{VOL}$ = 1[5] | 0.4 | 0.6 | | MHz |
| Closed-Loop Output Resistance | $R_O$ | $V_O$ = 0, $I_O$ = 0 | | 60 | | Ω |
| Power Consumption | $P_d$ | $V_S$ = ±15 V, No Load | | 75 | 120 | mW |
| | | $V_S$ = ±13 V, No Load | | 4 | 6 | mW |
| Offset Adjustment Range | | $R_P$ = 20 kΩ | | ±4 | | mV |

NOTES

[1]Input offset voltage measurements are performed by automated test equipment approximately 0.5 seconds after application of power.

[2]Long-term input offset voltage stability refers to the averaged trend time of VOS vs. Time over extended periods after the first 30 days of operation. Excluding the initial hour of operation, changes in VOS during the first 30 operating days are typically 2.5 µV refer to the typical performance curves. Parameter is sample tested.

[3]Sample tested.

[4]Guaranteed by design.

[5]Guaranteed but not tested.

Specifications subject to change without notice.

## OP07C ELECTRICAL CHARACTERISTICS ($V_S$ = ±15 V, $T_A$ = 25°C, unless otherwise noted.)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| INPUT CHARACTERISTICS | | | | | | |
| Input Offset Voltage[1] | $V_{OS}$ | | | 60 | 150 | µV |
| Long-Term $V_{OS}$ Stability[2] | $V_{OS}$/Time | | | 0.4 | 2.0 | µV/Mo |
| Input Offset Current | $I_{OS}$ | | | 0.8 | 6.0 | nA |
| Input Bias Current | $I_B$ | | | ±1.8 | ±7.0 | nA |
| Input Noise Voltage | $e_n$ p-p | 0.1 Hz to 10 Hz[3] | | 0.38 | 0.65 | µV p-p |
| Input Noise Voltage Density | $e_n$ | $f_O$ = 10 Hz | | 10.5 | 20.0 | nV√Hz |
| | | $f_O$ = 100 Hz[3] | | 10.2 | 13.5 | nV√Hz |
| | | $f_O$ = 1 kHz | | 9.8 | 11.5 | nV√Hz |
| Input Noise Current | $I_n$ p-p | | | 15 | 35 | pA p-p |
| Input Noise Current Density | $I_n$ | $f_O$ = 10 Hz | | 0.35 | 0.90 | pA√Hz |
| | | $f_O$ = 100 Hz[3] | | 0.15 | 0.27 | pA√Hz |
| | | $f_O$ = 1 kHz | | 0.13 | 0.18 | pA√Hz |
| Input Resistance- Differential Mode[4] | $R_{IN}$ | | 8 | 33 | | mΩ |
| Input Resistance- Common-Mode | $R_{INCM}$ | | | 120 | | GΩ |
| Input Voltage Range | IVR | | ±13 | ±14 | | V |
| Common-Mode Rejection Ratio | CMRR | $V_{CM}$ = ±13 V | 100 | 120 | | dB |
| Power Supply Rejection Ratio | PSRR | $V_S$ = ±3 V to ±18 V | | 7 | 32 | µV/V |
| Large-Signal Voltage Gain | $A_{VO}$ | $R_L$ ≥ 2 kΩ, $V_O$ = ±10 V | 120 | 400 | | V/mV |
| | | $R_L$ ≥ 500 Ω, $V_O$ = ±0.5 V, $V_S$ = ±3 V[4] | 100 | 400 | | V/mV |
| OUTPUT CHARACTERISTICS | | | | | | |
| Output Voltage Swing | $V_O$ | $R_L$ ≥ 10 kΩ | ±12.0 | ±13.0 | | V |
| | | $R_L$ ≥ 2 kΩ | ±11.5 | ±12.8 | | V |
| | | $R_L$ ≥ 1 kΩ | | ±12.0 | | V |
| DYNAMIC PERFORMANCE | | | | | | |
| Slew Rate | SR | $R_L$ ≥ 2 kΩ[3] | 0.1 | 0.3 | | V/µs |
| Closed-Loop Bandwidth | BW | $A_{VOL}$ = 1[5] | 0.4 | 0.6 | | MHz |
| Closed-Loop Output Resistance | $R_O$ | $V_O$ = 0, $I_O$ = 0 | | 60 | | Ω |
| Power Consumption | $P_d$ | $V_S$ = ±15 V, No Load | | 80 | 150 | mW |
| | | $V_S$ = ±13 V, No Load | | 4 | 8 | mW |
| Offset Adjustment Range | | $R_P$ = 20 kΩ | | ±4 | | mV |

NOTES

[1]Input offset voltage measurements are performed by automated test equipment approximately 0.5 seconds after application of power.

[2]Long-term input offset voltage stability refers to the averaged trend time of VOS vs. Time over extended periods after the first 30 days of operation. Excluding the initial hour of operation, changes in VOS during the first 30 operating days are typically 2.5 µV refer to the typical performance curves. Parameter is sample tested.

[3]Sample tested.

[4]Guaranteed by design.

[5]Guaranteed but not tested.

Specifications subject to change without notice.

# OP07–SPECIFICATIONS

## OP07E ELECTRICAL CHARACTERISTICS ($V_S$ = ±15 V, 0°C ≤ $T_A$ ≤ 70°C, unless otherwise noted.)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| INPUT CHARACTERISTICS | | | | | | |
| Input Offset Voltage[1] | $V_{OS}$ | | | 45 | 130 | µV |
| Voltage Drift without External Trim[2] | $TCV_{OS}$ | | | 0.3 | 1.3 | µV/°C |
| Voltage Drift with External Trim[3] | $TCV_{OSN}$ | $R_P$ = 20 kΩ | | 0.3 | 1.3 | µV/°C |
| Input Offset Current | $I_{OS}$ | | | 0.9 | 5.3 | nA |
| Input Offset Current Drift | $TCI_{OS}$ | | | 8 | 35 | pA/°C |
| Input Bias Current | $I_B$ | | | ±1.5 | ±5.5 | nA |
| Input Bias Current Drift | $TCI_B$ | | | 13 | 35 | pA/°C |
| Input Voltage Range | IVR | | ±13 | ±13.5 | | V |
| Common-Mode Rejection Ratio | CMRR | $V_{CM}$ = ±13 V | 103 | 123 | | dB |
| Power Supply Rejection Ratio | PSRR | $V_S$ = ±3 V to ±18 V | | 7 | 32 | µV/V |
| Large-Signal Voltage Gain | $A_{VO}$ | $R_L$ ≥ 2 kΩ, $V_O$ = ±10 V | 180 | 450 | | V/mV |
| OUTPUT CHARACTERISTICS | | | | | | |
| Output Voltage Swing | $V_O$ | $R_L$ ≥ 10 kΩ | ±12 | ±12.6 | | V |

NOTES
[1]Input offset voltage measurements are performed by automated test equipment approximately 0.5 seconds after application of power.
[2]Guaranteed by design.
[3]Sample tested.

Specifications subject to change without notice.

## OP07C ELECTRICAL CHARACTERISTICS ($V_S$ = ±15 V, −40°C ≤ $T_A$ ≤ +85°C, unless otherwise noted.)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| INPUT CHARACTERISTICS | | | | | | |
| Input Offset Voltage[1] | $V_{OS}$ | | | 85 | 250 | µV |
| Voltage Drift without External Trim[2] | $TCV_{OS}$ | | | 0.5 | 1.8 | µV/°C |
| Voltage Drift with External Trim[3] | $TCV_{OSN}$ | $R_P$ = 20 kΩ | | 0.4 | 1.8 | µV/°C |
| Input Offset Current | $I_{OS}$ | | | 1.6 | 8.0 | nA |
| Input Offset Current Drift | $TCI_{OS}$ | | | 12 | 50 | pA/°C |
| Input Bias Current | $I_B$ | | | ±2.2 | ±9.0 | nA |
| Input Bias Current Drift | $TCI_B$ | | | 18 | 50 | pA/°C |
| Input Voltage Range | IVR | | ±13 | ±13.5 | | V |
| Common-Mode Rejection Ratio | CMRR | $V_{CM}$ = ±13 V | 97 | 120 | | dB |
| Power Supply Rejection Ratio | PSRR | $V_S$ = ±3 V to ±18 V | | 10 | 51 | µV/V |
| Large-Signal Voltage Gain | $A_{VO}$ | $R_L$ ≥ 2 kΩ, $V_O$ = ±10 V | 100 | 400 | | V/mV |
| OUTPUT CHARACTERISTICS | | | | | | |
| Output Voltage Swing | $V_O$ | $R_L$ ≥ 10 kΩ | ±11 | ±12.6 | | V |

NOTES
[1]Input offset voltage measurements are performed by automated test equipment approximately 0.5 seconds after application of power.
[2]Guaranteed by design.
[3]Sample tested.

Specifications subject to change without notice.

**ABSOLUTE MAXIMUM RATINGS***

Supply Voltage ($V_S$) . . . . . ±22 V
Input Voltage* . . . . . ±22 V
Differential Input Voltage . . . . . ±30 V
Output Short-Circuit Duration . . . . . Indefinite
Storage Temperature Range
S, P Packages . . . . . –65°C to +125°C
Operating Temperature Range
OP07E . . . . . 0°C to 70°C
OP07C . . . . . –40°C to +85°C
Junction Temperature Range . . . . . 150°C
Lead Temperature Range (Soldering, 60 sec) . . . . . 300°C

*For supply voltages less than ±22 V, the absolute maximum input voltage is equal to the supply voltage.

| Package Type | $\theta_{JA}$* | $\theta_{JC}$ | Units |
|---|---|---|---|
| 8-Lead Plastic DIP (P) | 103 | 43 | °C/W |
| 8-Lead SOIC (S) | 158 | 43 | °C/W |

*$\theta_{JA}$ is specified for worst case conditions, i.e., $\theta_{JA}$ is specified for device in socket for P-DIP package, $\theta_{JA}$ is specified for device soldered to printed circuit board for SO package.

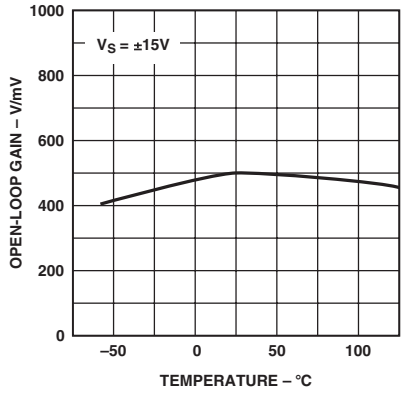
**ORDERING GUIDE**

| Model | Temperature Range | Package Description | Package Option | Branding Information |
|---|---|---|---|---|
| OP07EP | 0°C to 70°C | 8-Lead Epoxy DIP | P-8 | |
| OP07CP | –40°C to 85°C | 8-Lead Epoxy DIP | P-8 | |
| OP07CS | –40°C to 85°C | 8-Lead SOIC | S-8 | |

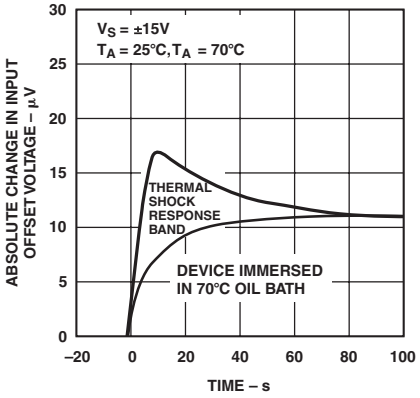
**CAUTION**

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although the OP07 features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high-energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

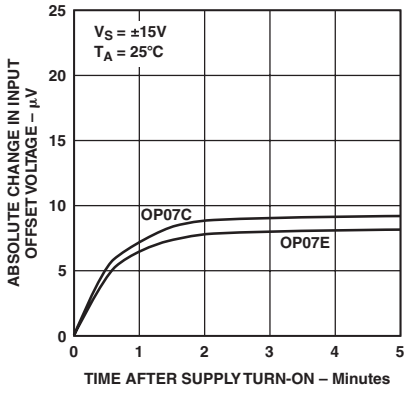
# OP07 – Typical Performance Characteristics



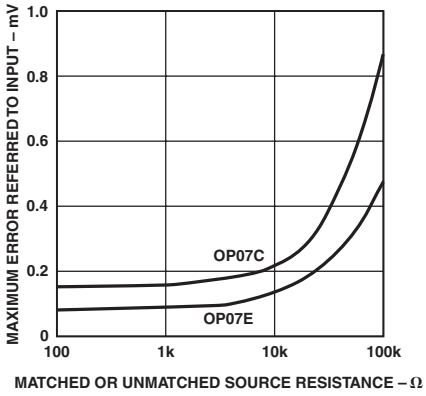

*TPC 1. Open-Loop Gain vs. Temperature*



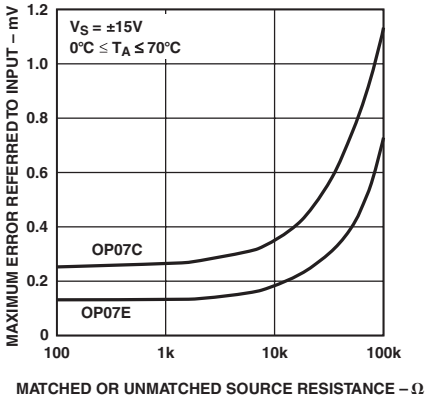

*TPC 2. Offset Voltage Change Due to Thermal Shock*



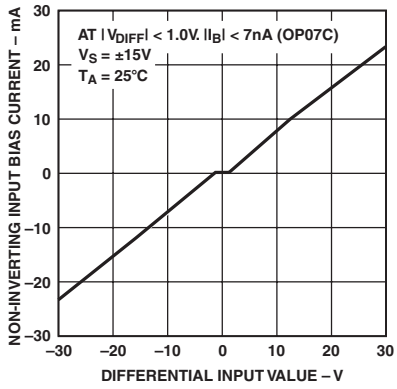

*TPC 3. Warm-Up Drift*



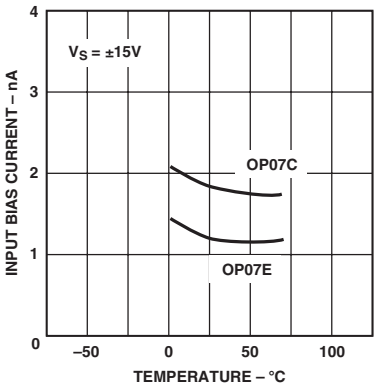

*TPC 4. Maximum Error vs. Source Resistance*



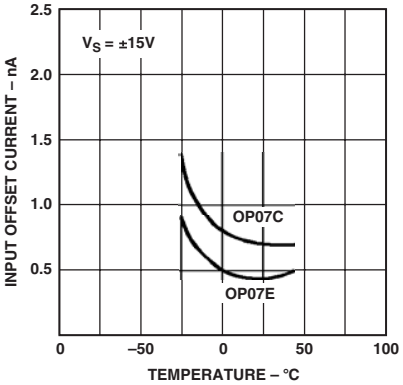

*TPC 5. Maximum Error vs. Source Resistance*



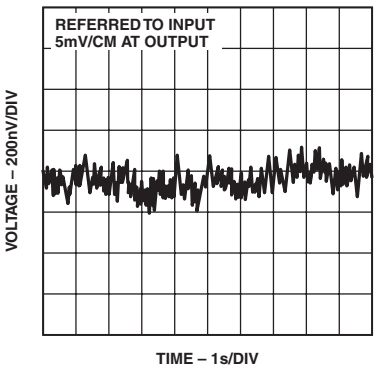

*TPC 6. Input Bias Current vs. Differential Input Voltage*



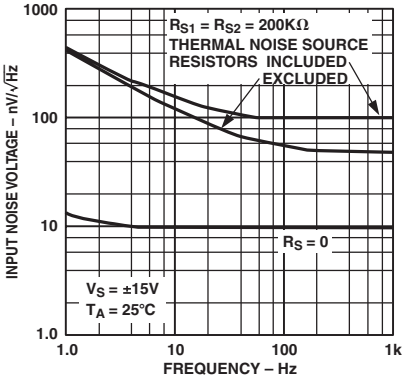

*TPC 7. Input Bias Current vs. Temperature*



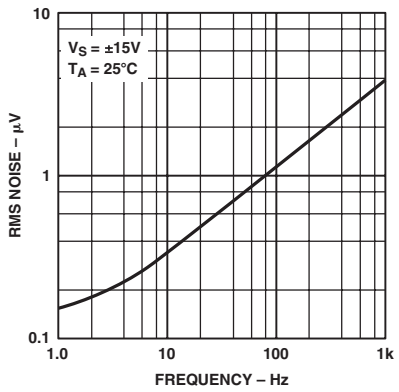

*TPC 8. Input Offset Current vs. Temperature*



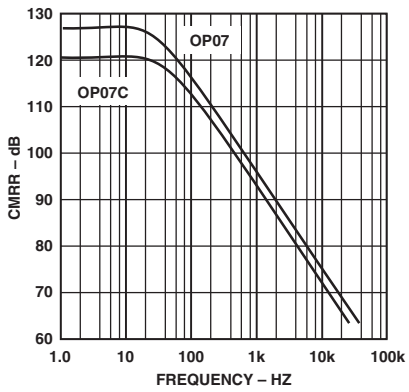

*TPC 9. Low Frequency Noise*



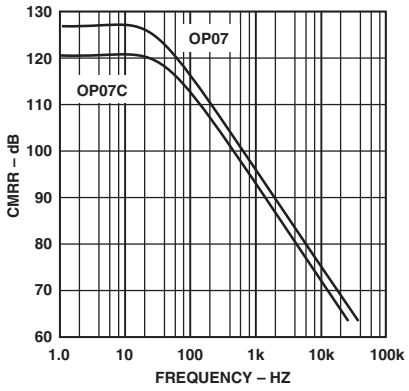

*TPC 10. Total Input Noise Voltage vs. Frequency*



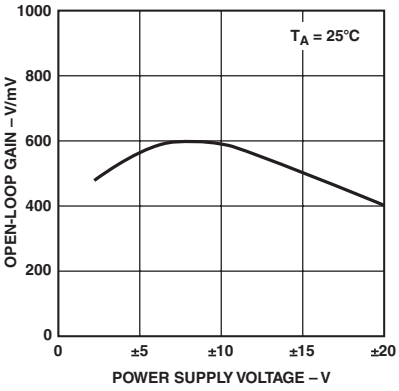

*TPC 11. Input Wideband Noise vs Bandwidth (0.1 Hz to Frequency Indicated)*



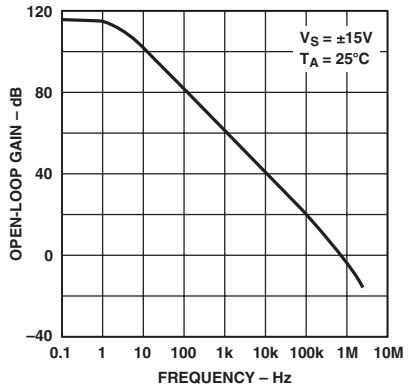

*TPC 12. CMRR vs. Frequency*



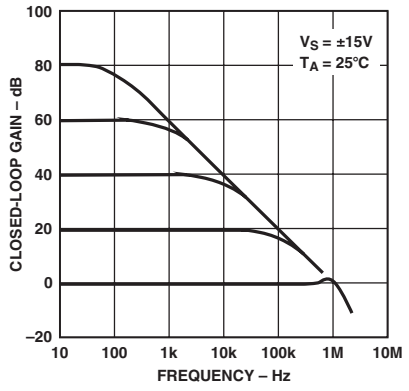

*TPC 13. PSRR vs. Frequency*



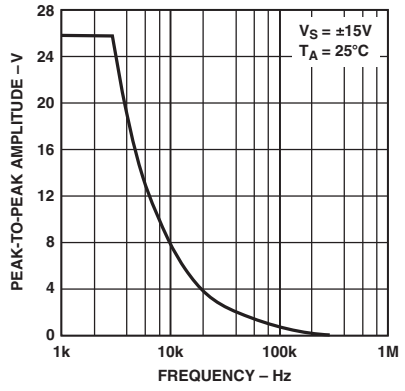

*TPC 14. Open-Loop Gain vs Power Supply Voltage*



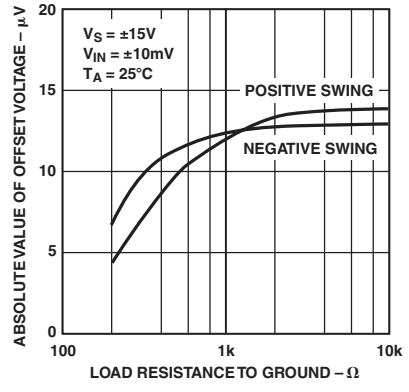

*TPC 15. Open-Loop Frequency Response*



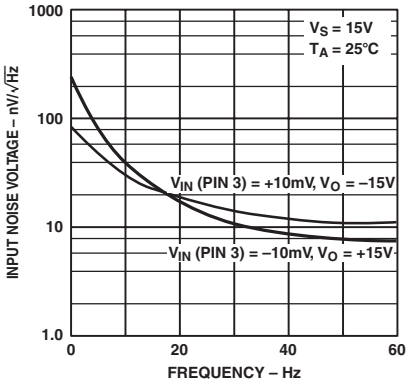

*TPC 16. Closed-Loop Response for Various Gain Configurations*



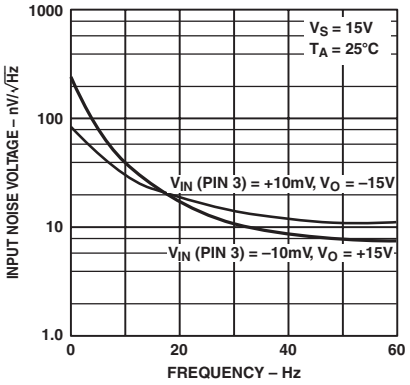

*TPC 17. Maximum Output Swing vs. Frequency*



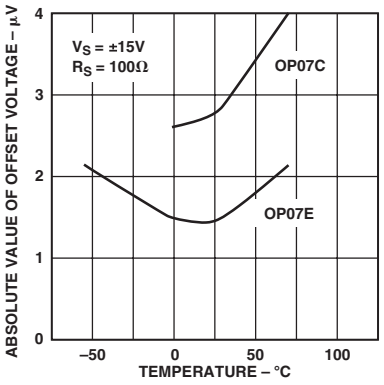

*TPC 18. Maximum Output Voltage vs. Load Resistance*



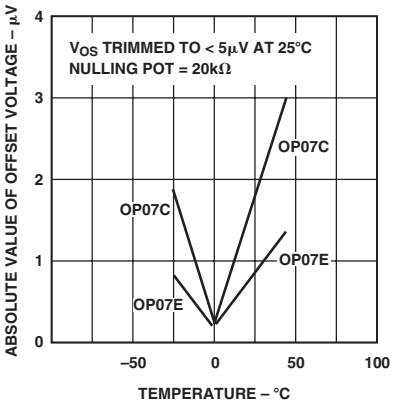

*TPC 19. Power Consumption vs. Power Supply*



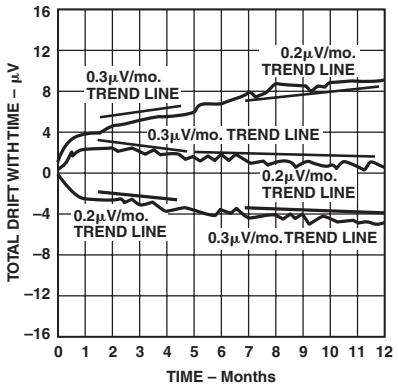

*TPC 20. Output Short-Circuit Current vs. Time*




*TPC 21. Untrimmed Offset Voltage vs. Temperature*




*TPC 22. Trimmed Offset Voltage vs. Temperature*




*TPC 23. Offset Voltage Stability vs. Time*

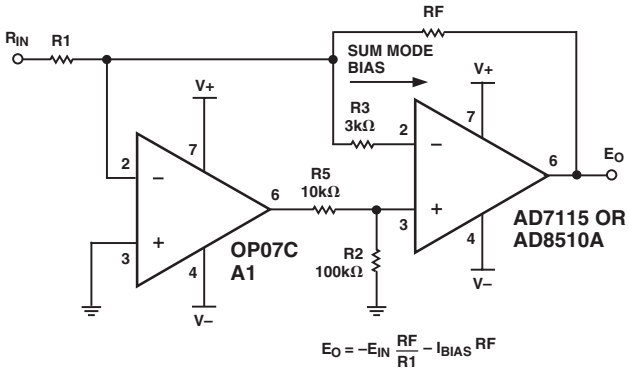



*Figure 2. Typical Offset Voltage Test Circuit*

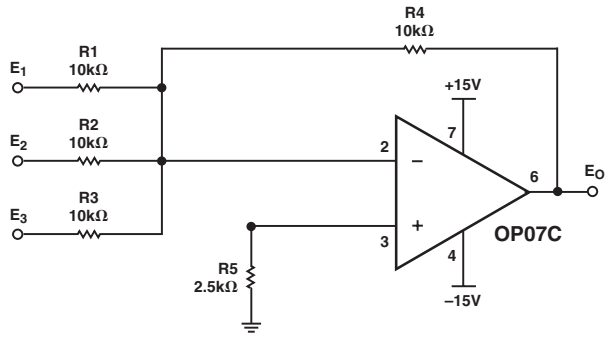



*Figure 3. Typical Low-Frequency Noise Circuit*

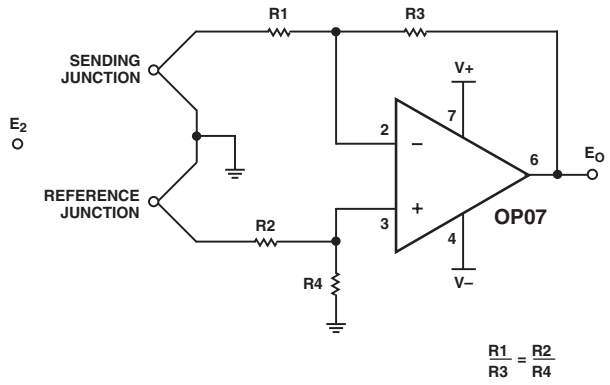



*Figure 4. Optional Offset Nulling Circuit*

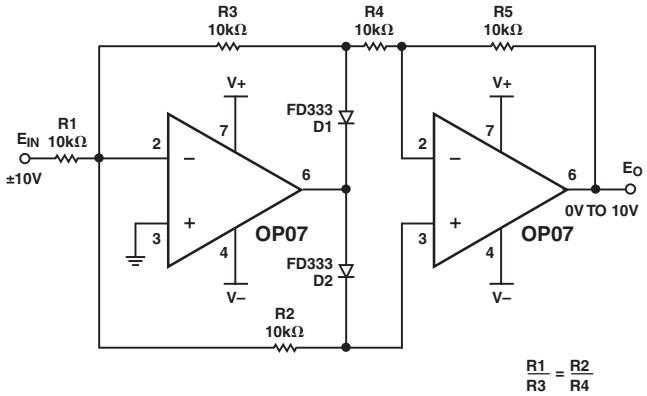
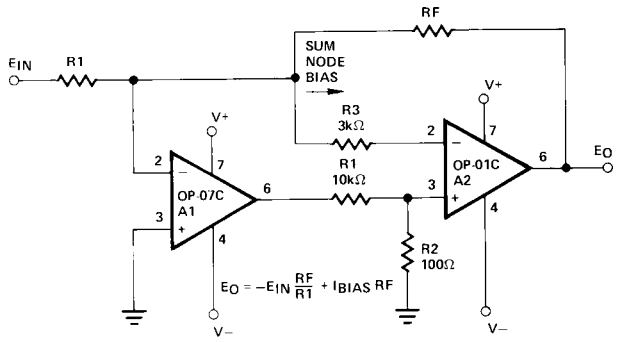
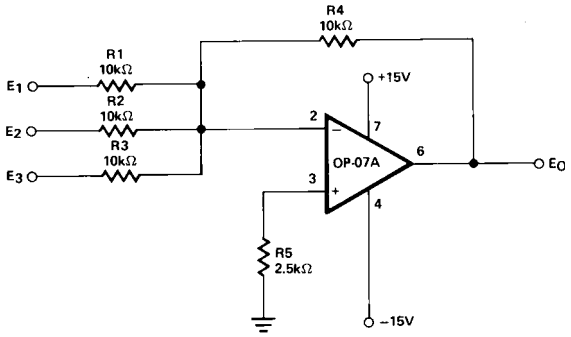



*Figure 5. Burn-In circuit*




PINOUTS SHOWN FOR J, P, AND Z PACKAGES

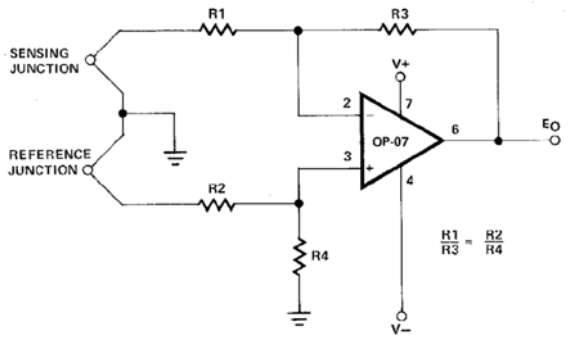
*Figure 6. High-Speed, Low VOS Composite Amplifier*




PINOUTS SHOWN FOR J, P, AND Z PACKAGES

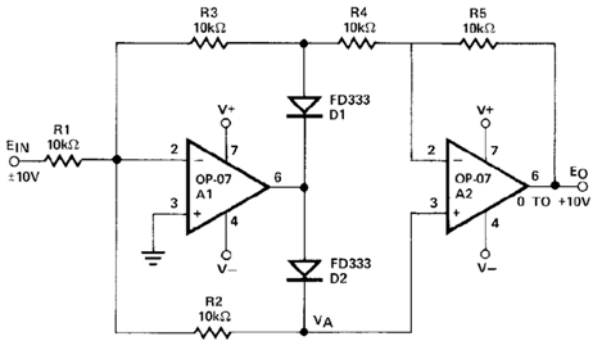
*Figure 7. Adjustment-Free Precision Summing Amplifier*

TYPICAL APPLICATIONS




PINOUTS SHOWN FOR J, P, AND Z PACKAGES

*Figure 8. High-Stability Thermocouple Amplifier*




PINOUTS SHOWN FOR J, P, AND Z PACKAGES

*Figure 9. Precision Absolute-Value Circuit*

**APPLICATIONS INFORMATION**

OP07 series units may be substituted directly into 725, 108A/308A* and OP05 sockets with or without removal of external compensation or nulling components. Additionally, the OP07 may be used in unnulled 741 type sockets. However, if conventional 741 nulling circuitry is in use, it should be modified or removed to enable proper OP07 operation. OP07 offset voltage may be nulled to zero through use of a potentiometer (see offset nulling circuit diagram).

**PRECISION ABSOLUTE-VALUE CIRCUIT**

The OP07 provides stable operation with load capacitance of up to 500 pF and ±10 V swings; larger capacitances should be decoupled with a 50 Q decoupling resistor.

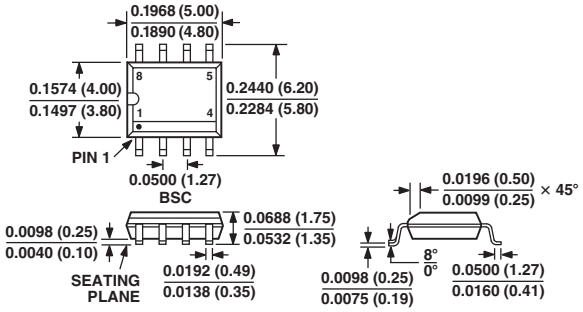
Stray thermoelectric voltages generated by dissimilar metals at the contacts to the input terminals can degrade drift performance. Therefore, best operation will be obtained when both input contacts are maintained at the same temperature, preferably close to the package temperature.

*TO-99 Package only

**OUTLINE DIMENSIONS**

Dimensions shown in inches and (mm).

**8-Lead SO DIP**
**(S-Suffix)**




# Revision History

| Location | Page |
|---|---|

**Data Sheet changed from REV. 0 to REV. A.**

## C.2 AD574A Technical Reference

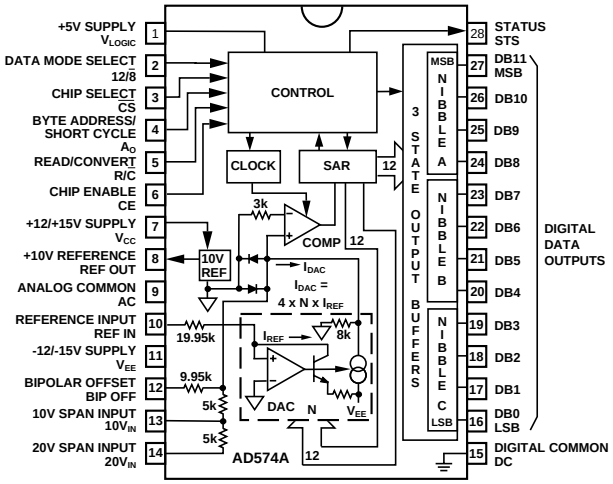
# Complete 12-Bit A/D Converter

# AD574A*

**FEATURES**

**Complete 12-Bit A/D Converter with Reference and Clock**
**8- and 16-Bit Microprocessor Bus Interface**
**Guaranteed Linearity Over Temperature**
**0°C to +70°C – AD574AJ, K, L**
**–55°C to +125°C – AD574AS, T, U**
**No Missing Codes Over Temperature**
**35 μs Maximum Conversion Time**
**Buried Zener Reference for Long-Term Stability and Low Gain T.C. 10 ppm/°C max AD574AL**
**12.5 ppm/°C max AD574AU**
**Ceramic DIP, Plastic DIP or PLCC Package**
**Available in Higher Speed, Pinout-Compatible Versions (15 μs AD674B, 80 μs AD774B; 10 μs (with SHA) AD1674)**
**Available in Versions Compliant with MIL-STD-883 and JAN QPL**

**BLOCK DIAGRAM AND PIN CONFIGURATION**




## PRODUCT DESCRIPTION

The AD574A is a complete 12-bit successive-approximation analog-to-digital converter with 3-state output buffer circuitry for direct interface to an 8- or 16-bit microprocessor bus. A high precision voltage reference and clock are included on-chip, and the circuit guarantees full-rated performance without external circuitry or clock signals.

The AD574A design is implemented using Analog Devices' Bipolar/$I^2L$ process, and integrates all analog and digital functions on one chip. Offset, linearity and scaling errors are minimized by active laser-trimming of thin-film resistors at the wafer stage. The voltage reference uses an implanted buried Zener for low noise and low drift. On the digital side, $I^2L$ logic is used for the successive-approximation register, control circuitry and 3-state output buffers.

The AD574A is available in six different grades. The AD574AJ, K, and L grades are specified for operation over the 0°C to +70°C temperature range. The AD574AS, T, and U are specified for the –55°C to +125°C range. All grades are available in a 28-pin hermetically-sealed ceramic DIP. Also, the J, K, and L grades are available in a 28-pin plastic DIP and PLCC, and the J and K grades are available in ceramic LCC.

The S, T, and U grades in ceramic DIP or LCC are available with optional processing to MIL-STD-883C Class B; the T and U grades are available as JAN QPL. The Analog Devices' Military Products Databook should be consulted for details on /883B testing of the AD574A.



REV. B



## PRODUCT HIGHLIGHTS

1. The AD574A interfaces to most 8- or 16-bit microprocessors. Multiple-mode three-state output buffers connect directly to the data bus while the read and convert commands are taken from the control bus. The 12 bits of output data can be read either as one 12-bit word or as two 8-bit bytes (one with 8 data bits, the other with 4 data bits and 4 trailing zeros).
2. The precision, laser-trimmed scaling and bipolar offset resistors provide four calibrated ranges: 0 volts to +10 volts and 0 volts to +20 volts unipolar, –5 volts to +5 volts and –10 volts to +10 volts bipolar. Typical bipolar offset and full-scale calibration errors of ±0.1% can be trimmed to zero with one external component each.
3. The internal buried Zener reference is trimmed to 10.00 volts with 0.2% maximum error and 15 ppm/°C typical T.C. The reference is available externally and can drive up to 1.5 mA beyond the requirements of the reference and bipolar offset resistors.
4. AD674B (15 μs) and AD774B (8 μs) provide higher speed, pin compatibility; AD1674 (10 μs) includes on-chip Sample-Hold Amplifier (SHA).

**One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.**
**Tel: 617/329-4700** **Fax: 617/326-8703**

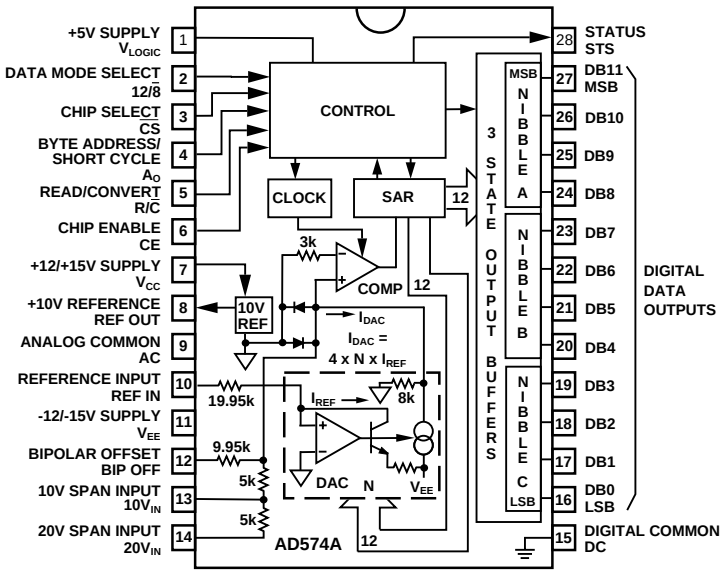
# AD574A–SPECIFICATIONS (@ +25°C with $V_{CC}$ = +15 V or +12 V, $V_{LOGIC}$ = +5 V, $V_{EE}$ = –15 V or –12 V unless otherwise noted)

| Model | AD574AJ Min | AD574AJ Typ | AD574AJ Max | AD574AK Min | AD574AK Typ | AD574AK Max | AD574AL Min | AD574AL Typ | AD574AL Max | Units |
|---|---|---|---|---|---|---|---|---|---|---|
| RESOLUTION | | | 12 | | | 12 | | | 12 | Bits |
| LINEARITY ERROR @ +25°C | | | ±1 | | | ±1/2 | | | ±1/2 | LSB |
| $T_{MIN}$ to $T_{MAX}$ | | | ±1 | | | ±1/2 | | | ±1/2 | LSB |
| DIFFERENTIAL LINEARITY ERROR (Minimum Resolution for Which No Missing Codes are Guaranteed) | | | | | | | | | | |
| $T_{MIN}$ to $T_{MAX}$ | 11 | | | 12 | | | 12 | | | Bits |
| UNIPOLAR OFFSET (Adjustable to Zero) | | | ±2 | | | ±1 | | | ±1 | LSB |
| BIPOLAR OFFSET (Adjustable to Zero) | | | ±4 | | | ±4 | | | ±2 | LSB |
| FULL-SCALE CALIBRATION ERROR (With Fixed 50 Ω Resistor from REF OUT to REF IN) (Adjustable to Zero) | | | 0.25 | | | 0.25 | | | 0.125 | % of FS |
| TEMPERATURE RANGE | 0 | | +70 | 0 | | +70 | 0 | | +70 | °C |
| TEMPERATURE COEFFICIENTS (Using Internal Reference) | | | | | | | | | | |
| $T_{MIN}$ to $T_{MAX}$ | | | | | | | | | | |
| Unipolar Offset | | | ±2 (10) | | | ±1 (5) | | | ±1 (5) | LSB (ppm/°C) |
| Bipolar Offset | | | ±2 (10) | | | ±1 (5) | | | ±1 (5) | LSB (ppm/°C) |
| Full-Scale Calibration | | | ±9 (50) | | | ±5 (27) | | | ±2 (10) | LSB (ppm/°C) |
| POWER SUPPLY REJECTION | | | | | | | | | | |
| Max Change in Full-Scale Calibration | | | | | | | | | | |
| $V_{CC}$ = 15 V ± 1.5 V or 12 V ± 0.6 V | | | ±2 | | | ±1 | | | ±1 | LSB |
| $V_{LOGIC}$ = 5 V ± 0.5 V | | | ±1/2 | | | ±1/2 | | | ±1/2 | LSB |
| $V_{EE}$ = –15 V ± 1.5 V or –12 V ± 0.6 V | | | ±2 | | | ±1 | | | ±1 | LSB |
| ANALOG INPUT | | | | | | | | | | |
| Input Ranges | | | | | | | | | | |
| Bipolar | –5 | | +5 | –5 | | +5 | –5 | | +5 | Volts |
| | –10 | | +10 | –10 | | +10 | –10 | | +10 | Volts |
| Unipolar | 0 | | +10 | 0 | | +10 | 0 | | +10 | Volts |
| | 0 | | +20 | 0 | | +20 | 0 | | +20 | Volts |
| Input Impedance | | | | | | | | | | |
| 10 Volt Span | 3 | 5 | 7 | 3 | 5 | 7 | 3 | 5 | 7 | kΩ |
| 20 Volt Span | 6 | 10 | 14 | 6 | 10 | 14 | 6 | 10 | 14 | kΩ |
| DIGITAL CHARACTERISTICS[1] ($T_{MIN}$–$T_{MAX}$) | | | | | | | | | | |
| Inputs[2] (CE, $\overline{CS}$, R/$\overline{C}$, $A_0$) | | | | | | | | | | |
| Logic "1" Voltage | +2.0 | | +5.5 | +2.0 | | +5.5 | +2.0 | | +5.5 | Volts |
| Logic "0" Voltage | –0.5 | | +0.8 | –0.5 | | +0.8 | –0.5 | | +0.8 | Volts |
| Current | –20 | | +20 | –20 | | +20 | –20 | | +20 | µA |
| Capacitance | | 5 | | | 5 | | | 5 | | pF |
| Output (DB11–DB0, STS) | | | | | | | | | | |
| Logic "1" Voltage ($I_{SOURCE}$ ≤ 500 µA) | +2.4 | | | +2.4 | | | +2.4 | | | Volts |
| Logic "0" Voltage ($I_{SINK}$ ≤ 1.6 mA) | | | +0.4 | | | +0.4 | | | +0.4 | Volts |
| Leakage (DB11–DB0, High-Z State) | –20 | | +20 | –20 | | +20 | –20 | | +20 | µA |
| Capacitance | | 5 | | | 5 | | | 5 | | pF |
| POWER SUPPLIES | | | | | | | | | | |
| Operating Range | | | | | | | | | | |
| $V_{LOGIC}$ | +4.5 | | +5.5 | +4.5 | | +5.5 | +4.5 | | +5.5 | Volts |
| $V_{CC}$ | +11.4 | | +16.5 | +11.4 | | +16.5 | +11.4 | | +16.5 | Volts |
| $V_{EE}$ | –11.4 | | –16.5 | –11.4 | | –16.5 | –11.4 | | –16.5 | Volts |
| Operating Current | | | | | | | | | | |
| $I_{LOGIC}$ | | 30 | 40 | | 30 | 40 | | 30 | 40 | mA |
| $I_{CC}$ | | 2 | 5 | | 2 | 5 | | 2 | 5 | mA |
| $I_{EE}$ | | 18 | 30 | | 18 | 30 | | 18 | 30 | mA |
| POWER DISSIPATION | | 390 | 725 | | 390 | 725 | | 390 | 725 | mW |
| INTERNAL REFERENCE VOLTAGE | 9.98 | 10.0 | 10.02 | 9.98 | 10.0 | 10.02 | 9.99 | 10.0 | 10.01 | Volts |
| Output Current (Available for External Loads)[3] (External Load Should not Change During Conversion) | | | 1.5 | | | 1.5 | | | 1.5 | mA |
| PACKAGE OPTIONS[4] | | | | | | | | | | |
| Ceramic (D-28) | | AD574ASD | | | AD574AKD | | | AD574ALD | | |
| Plastic (N-28) | | AD574AJN | | | AD574AKN | | | AD574ALN | | |
| PLCC (P-28A) | | AD574AJP | | | AD574AKP | | | | | |
| LCC (E-28A) | | AD574AJE | | | AD574AKE | | | | | |

NOTES
[1]Detailed Timing Specifications appear in the Timing Section.
[2]12/$\overline{8}$ Input is not TTL-compatible and must be hard wired to $V_{LOGIC}$ or Digital Common.
[3]The reference should be buffered for operation on ±12 V supplies.
[4]D = Ceramic DIP; N = Plastic DIP; P = Plastic Leaded Chip Carrier.
Specifications subject to change without notice.

| Model | AD574AS Min | AD574AS Typ | AD574AS Max | AD574AT Min | AD574AT Typ | AD574AT Max | AD574AU Min | AD574AU Typ | AD574AU Max | Units |
|---|---|---|---|---|---|---|---|---|---|---|
| RESOLUTION | | | 12 | | | 12 | | | 12 | Bits |
| LINEARITY ERROR @ +25°C | | | ±1 | | | ±1/2 | | | ±1/2 | LSB |
| $T_{MIN}$ to $T_{MAX}$ | | | ±1 | | | ±1 | | | ±1 | LSB |
| DIFFERENTIAL LINEARITY ERROR (Minimum Resolution for Which No Missing Codes are Guaranteed) $T_{MIN}$ to $T_{MAX}$ | 11 | | | 12 | | | 12 | | | Bits |
| UNIPOLAR OFFSET (Adjustable to Zero) | | | ±2 | | | ±1 | | | ±1 | LSB |
| BIPOLAR OFFSET (Adjustable to Zero) | | | ±4 | | | ±4 | | | ±2 | LSB |
| FULL-SCALE CALIBRATION ERROR (With Fixed 50 Ω Resistor from REF OUT to REF IN) (Adjustable to Zero) | | | 0.25 | | | 0.25 | | | 0.125 | % of FS |
| TEMPERATURE RANGE | –55 | | +125 | –55 | | +125 | –55 | | +125 | °C |
| TEMPERATURE COEFFICIENTS (Using Internal Reference) ($T_{MIN}$ to $T_{MAX}$) | | | | | | | | | | |
| Unipolar Offset | | | ±2 (5) | | | ±1 (2.5) | | | ±1 (2.5) | LSB (ppm/°C) |
| Bipolar Offset | | | ±4 (10) | | | ±2 (5) | | | ±1 (2.5) | LSB (ppm/°C) |
| Full-Scale Calibration | | | ±20 (50) | | | ±10 (25) | | | ±5 (12.5) | LSB (ppm/°C) |
| POWER SUPPLY REJECTION Max Change in Full-Scale Calibration | | | | | | | | | | |
| $V_{CC}$ = 15 V ± 1.5 V or 12 V ± 0.6 V | | | ±2 | | | ±1 | | | ±1 | LSB |
| $V_{LOGIC}$ = 5 V ± 0.5 V | | | ±1/2 | | | ±1/2 | | | ±1/2 | LSB |
| $V_{EE}$ = –15 V ± 1.5 V or –12 V ± 0.6 V | | | ±2 | | | ±1 | | | ±1 | LSB |
| ANALOG INPUT | | | | | | | | | | |
| Input Ranges | | | | | | | | | | |
| Bipolar | –5 | | +5 | –5 | | +5 | –5 | | +5 | Volts |
| | –10 | | +10 | –10 | | +10 | –10 | | +10 | Volts |
| Unipolar | 0 | | +10 | 0 | | +10 | 0 | | +10 | Volts |
| | 0 | | +20 | 0 | | +20 | 0 | | +20 | Volts |
| Input Impedance | | | | | | | | | | |
| 10 Volt Span | 3 | 5 | 7 | 3 | 5 | 7 | 3 | 5 | 7 | kΩ |
| 20 Volt Span | 6 | 10 | 14 | 6 | 10 | 14 | 6 | 10 | 14 | kΩ |
| DIGITAL CHARACTERISTICS[1] ($T_{MIN}$–$T_{MAX}$) | | | | | | | | | | |
| Inputs[2] (CE, $\overline{CS}$, R/$\overline{C}$, $A_0$) | | | | | | | | | | |
| Logic "1" Voltage | +2.0 | | +5.5 | +2.0 | | +5.5 | +2.0 | | +5.5 | Volts |
| Logic "0" Voltage | –0.5 | | +0.8 | –0.5 | | +0.8 | –0.5 | | +0.8 | Volts |
| Current | –20 | | +20 | –20 | | +20 | –20 | | +20 | µA |
| Capacitance | | 5 | | | 5 | | | 5 | | pF |
| Output (DB11–DB0, STS) | | | | | | | | | | |
| Logic "1" Voltage ($I_{SOURCE} \leq 500$ µA) | +2.4 | | | +2.4 | | | +2.4 | | | Volts |
| Logic "0" Voltage ($I_{SINK} \leq 1.6$ mA) | | | +0.4 | | | +0.4 | | | +0.4 | Volts |
| Leakage (DB11–DB0, High-Z State) | –20 | | +20 | –20 | | +20 | –20 | | +20 | µA |
| Capacitance | | 5 | | | 5 | | | 5 | | pF |
| POWER SUPPLIES Operating Range | | | | | | | | | | |
| $V_{LOGIC}$ | +4.5 | | +5.5 | +4.5 | | +5.5 | +4.5 | | +5.5 | Volts |
| $V_{CC}$ | +11.4 | | +16.5 | +11.4 | | +16.5 | +11.4 | | +16.5 | Volts |
| $V_{EE}$ | –11.4 | | –16.5 | –11.4 | | –16.5 | –11.4 | | –16.5 | Volts |
| Operating Current | | | | | | | | | | |
| $I_{LOGIC}$ | | 30 | 40 | | 30 | 40 | | 30 | 40 | mA |
| $I_{CC}$ | | 2 | 5 | | 2 | 5 | | 2 | 5 | mA |
| $I_{EE}$ | | 18 | 30 | | 18 | 30 | | 18 | 30 | mA |
| POWER DISSIPATION | | 390 | 725 | | 390 | 725 | | 390 | 725 | mW |
| INTERNAL REFERENCE VOLTAGE | 9.98 | 10.0 | 10.02 | 9.98 | 10.0 | 10.02 | 9.99 | 10.0 | 10.01 | Volts |
| Output Current (Available for External Loads)[3] (External Load Should not Change During Conversion) | | | 1.5 | | | 1.5 | | | 1.5 | mA |
| PACKAGE OPTION[4] | | | | | | | | | | |
| Ceramic (D-28) | | AD574ASD | | | AD574ATD | | | AD574AUD | | |

NOTES
[1]Detailed Timing Specifications appear in the Timing Section.
[2]12/$\overline{8}$ Input is not TTL-compatible and must be hard wired to $V_{LOGIC}$ or Digital Common.
[3]The reference should be buffered for operation on ±12 V supplies.
[4]D = Ceramic DIP.
Specifications subject to change without notice.




*AD574A Block Diagram and Pin Configuration*

**ABSOLUTE MAXIMUM RATINGS***

(Specifications apply to all grades, except where noted)

$V_{CC}$ to Digital Common . . . . . 0 V to +16.5 V
$V_{EE}$ to Digital Common . . . . . 0 V to –16.5 V
$V_{LOGIC}$ to Digital Common . . . . . 0 V to +7 V
Analog Common to Digital Common . . . . . ±1 V
Control Inputs (CE, $\overline{CS}$, $A_0$ 12/$\overline{8}$, R/$\overline{C}$) to
Digital Common . . . . . –0.5 V to $V_{LOGIC}$ + 0.5 V
Analog Inputs (REF IN, BIP OFF, 10 $V_{IN}$) to
Analog Common . . . . . $V_{EE}$ to $V_{CC}$
20 $V_{IN}$ to Analog Common . . . . . ±24 V
REF OUT . . . . . Indefinite Short to Common
Momentary Short to $V_{CC}$
Chip Temperature . . . . . 175°C
Power Dissipation . . . . . 825 mW
Lead Temperature (Soldering, 10 sec). . . . . +300°C
Storage Temperature (Ceramic) . . . . . –65°C to +150°C
(Plastic) . . . . . –25°C to +100°C

*Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**ORDERING GUIDE**

| Model[1] | Temperature Range | Linearity Error Max ($T_{MIN}$ to $T_{MAX}$) | Resolution No Missing Codes ($T_{MIN}$ to $T_{MAX}$) | Max Full Scale T.C. (ppm/°C) |
|---|---|---|---|---|
| AD574AJ(X) | 0°C to +70°C | ±1 LSB | 11 Bits | 50.0 |
| AD574AK(X) | 0°C to +70°C | ±1/2 LSB | 12 Bits | 27.0 |
| AD574AL(X) | 0°C to +70°C | ±1/2 LSB | 12 Bits | 10.0 |
| AD574AS(X)[2] | –55°C to +125°C | ±1 LSB | 11 Bits | 50.0 |
| AD574AT(X)[2] | –55°C to +125°C | ±1 LSB | 12 Bits | 25.0 |
| AD574AU(X)[2] | –55°C to +125°C | ±1 LSB | 12 Bits | 12.5 |

NOTES
[1]X = Package designator. Available packages are: D (D-28) for all grades. E (E-28A) for J and K grades and /883B processed S, T and U grades. N (N-28) for J, K, and L grades. P (P-28A) for PLCC in J, K grades. Example: AD574AKN is K grade in plastic DIP.
[2]For details on grade and package offerings screened in accordance with MIL-STD-883, refer to Analog Devices Military Products Databook.

## THE AD574A OFFERS GUARANTEED MAXIMUM LINEARITY ERROR OVER THE FULL OPERATING TEMPERATURE RANGE

### DEFINITIONS OF SPECIFICATIONS

LINEARITY ERROR

Linearity error refers to the deviation of each individual code from a line drawn from "zero" through "full scale". The point used as "zero" occurs 1/2 LSB (1.22 mV for 10 volt span) before the first code transition (all zeros to only the LSB "on"). "Full scale" is defined as a level 1 1/2 LSB beyond the last code transition (to all ones). The deviation of a code from the true straight line is measured from the middle of each particular code.

The AD574AK, L, T, and U grades are guaranteed for maximum nonlinearity of ±1/2 LSB. For these grades, this means that an analog value which falls exactly in the center of a given code width will result in the correct digital output code. Values nearer the upper or lower transition of the code width may produce the next upper or lower digital output code. The AD574AJ and S grades are guaranteed to ±1 LSB max error. For these grades, an analog value which falls within a given code width will result in either the correct code for that region or either adjacent one.

Note that the linearity error is not user-adjustable.

DIFFERENTIAL LINEARITY ERROR (NO MISSING CODES)

A specification which guarantees no missing codes requires that every code combination appear in a monotonic increasing sequence as the analog input level is increased. Thus every code must have a finite width. For the AD574AK, L, T, and U grades, which guarantee no missing codes to 12-bit resolution, all 4096 codes must be present over the entire operating temperature ranges. The AD574AJ and S grades guarantee no missing codes to 11-bit resolution over temperature; this means that all code combinations of the upper 11 bits must be present; in practice very few of the 12-bit codes are missing.

UNIPOLAR OFFSET

The first transition should occur at a level 1/2 LSB above analog common. Unipolar offset is defined as the deviation of the actual transition from that point. This offset can be adjusted as discussed on the following two pages. The unipolar offset temperature coefficient specifies the maximum change of the transition point over temperature, with or without external adjustment.

BIPOLAR OFFSET

In the bipolar mode the major carry transition (0111 1111 1111 to 1000 0000 0000) should occur for an analog value 1/2 LSB below analog common. The bipolar offset error and temperature coefficient specify the initial deviation and maximum change in the error over temperature.

QUANTIZATION UNCERTAINTY

Analog-to-digital converters exhibit an inherent quantization uncertainty of ±1/2 LSB. This uncertainty is a fundamental characteristic of the quantization process and cannot be reduced for a converter of given resolution.

LEFT-JUSTIFIED DATA

The data format used in the AD574A is left-justified. This means that the data represents the analog input as a fraction of full-scale, ranging from 0 to $\frac{4095}{4096}$. This implies a binary point to the left of the MSB.

FULL-SCALE CALIBRATION ERROR

The last transition (from 1111 1111 1110 to 1111 1111 1111) should occur for an analog value 1 1/2 LSB below the nominal full scale (9.9963 volts for 10.000 volts full scale). The full-scale calibration error is the deviation of the actual level at the last transition from the ideal level. This error, which is typically 0.05% to 0.1% of full scale, can be trimmed out as shown in Figures 3 and 4.

TEMPERATURE COEFFICIENTS

The temperature coefficients for full-scale calibration, unipolar offset, and bipolar offset specify the maximum change from the initial (25°C) value to the value at $T_{MIN}$ or $T_{MAX}$.

POWER SUPPLY REJECTION

The standard specifications for the AD574A assume use of +5.00 V and ±15.00 V or ±12.00 V supplies. The only effect of power supply error on the performance of the device will be a small change in the full-scale calibration. This will result in a linear change in all lower order codes. The specifications show the maximum full-scale change from the initial value with the supplies at the various limits.

CODE WIDTH

A fundamental quantity for A/D converter specifications is the code width. This is defined as the range of analog input values for which a given digital output code will occur. The nominal value of a code width is equivalent to 1 least significant bit (LSB) of the full-scale range or 2.44 mV out of 10 volts for a 12-bit ADC.

## CIRCUIT OPERATION

The AD574A is a complete 12-bit A/D converter which requires no external components to provide the complete successive-approximation analog-to-digital conversion function. A block diagram of the AD574A is shown in Figure 1.

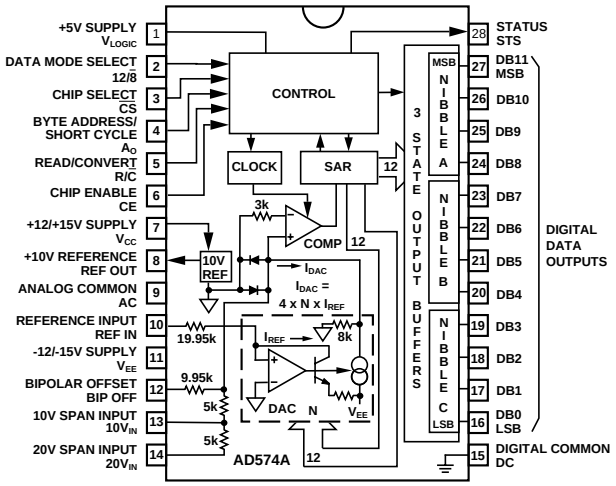



*Figure 1. Block Diagram of AD574A 12-Bit A-to-D Converter*

When the control section is commanded to initiate a conversion (as described later), it enables the clock and resets the successive-approximation register (SAR) to all zeros. Once a conversion cycle has begun, it cannot be stopped or restarted and data is not available from the output buffers. The SAR, timed by the clock, will sequence through the conversion cycle and return an end-of-convert flag to the control section. The control section will then disable the clock, bring the output status flag low, and enable control functions to allow data read functions by external command.

During the conversion cycle, the internal 12-bit current output DAC is sequenced by the SAR from the most significant bit (MSB) to least significant bit (LSB) to provide an output current which accurately balances the input signal current through the 5 kΩ (or 10 kΩ) input resistor. The comparator determines whether the addition of each successively-weighted bit current causes the DAC current sum to be greater or less than the input current; if the sum is less, the bit is left on; if more, the bit is turned off. After testing all the bits, the SAR contains a 12-bit binary code which accurately represents the input signal to within ±1/2 LSB.

The temperature-compensated buried Zener reference provides the primary voltage reference to the DAC and guarantees excellent stability with both time and temperature. The reference is trimmed to 10.00 volts ±0.2%; it can supply up to 1.5 mA to an external load in addition to the requirements of the reference input resistor (0.5 mA) and bipolar offset resistor (1 mA) when the AD574A is powered from ±15 V supplies. If the AD574A is used with ±12 V supplies, or if external current must be supplied over the full temperature range, an external buffer amplifier is recommended. Any external load on the AD574A reference must remain constant during conversion. The thin-film application resistors are trimmed to match the full-scale output current of the DAC. There are two 5 kΩ input scaling resistors to allow either a 10 volt or 20 volt span. The 10 kΩ bipolar offset resistor is grounded for unipolar operation and connected to the 10 volt reference for bipolar operation.

## DRIVING THE AD574 ANALOG INPUT

The internal circuitry of the AD574 dictates that its analog input be driven by a low source impedance. Voltage changes at the current summing node of the internal comparator result in abrupt modulations of the current at the analog input. For accurate 12-bit conversions the driving source must be capable of holding a constant output voltage under these dynamically changing load conditions.

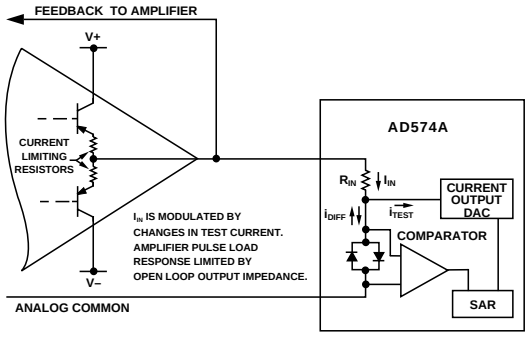



*Figure 2. Op Amp – AD574A Interface*

The output impedance of an op amp has an open-loop value which, in a closed loop, is divided by the loop gain available at the frequency of interest. The amplifier should have acceptable loop gain at 500 kHz for use with the AD574A. To check whether the output properties of a signal source are suitable, monitor the AD574's input with an oscilloscope while a conversion is in progress. Each of the 12 disturbances should subside in 1 µs or less.

For applications involving the use of a sample-and-hold amplifier, the AD585 is recommended. The AD711 or AD544 op amps are recommended for dc applications.

## SAMPLE-AND-HOLD AMPLIFIERS

Although the conversion time of the AD574A is a maximum of 35 µs, to achieve accurate 12-bit conversions of frequencies greater than a few Hz requires the use of a sample-and-hold amplifier (SHA). If the voltage of the analog input signal driving the AD574A changes by more than 1/2 LSB over the time interval needed to make a conversion, then the input requires a SHA.

The AD585 is a high linearity SHA capable of directly driving the analog input of the AD574A. The AD585's fast acquisition time, low aperture and low aperture jitter are ideally suited for high-speed data acquisition systems. Consider the AD574A converter with a 35 µs conversion time and an input signal of 10 V p-p: the maximum frequency which may be applied to achieve rated accuracy is 1.5 Hz. However, with the addition of an AD585, as shown in Figure 3, the maximum frequency increases to 26 kHz.

The AD585's low output impedance, fast-loop response, and low droop maintain 12-bits of accuracy under the changing load conditions that occur during a conversion, making it suitable for use in high accuracy conversion systems. Many other SHAs cannot achieve 12-bits of accuracy and can thus compromise a system. The AD585 is recommended for AD574A applications requiring a sample and hold.

An alternate approach is to use the AD1674, which combines the ADC and SHA on one chip, with a total throughput time of 10 µs.

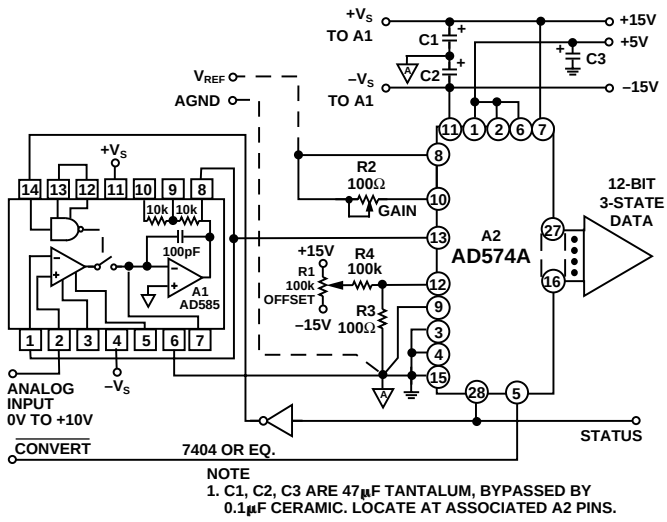



*Figure 3. AD574A with AD585 Sample and Hold*

## SUPPLY DECOUPLING AND LAYOUT CONSIDERATIONS

It is critically important that the AD574A power supplies be filtered, well regulated, and free from high frequency noise. Use of noisy supplies will cause unstable output codes. Switching power supplies are not recommended for circuits attempting to achieve 12-bit accuracy unless great care is used in filtering any switching spikes present in the output. Remember that a few millivolts of noise represents several counts of error in a 12-bit ADC.

Decoupling capacitors should be used on all power supply pins; the +5 V supply decoupling capacitor should be connected directly from Pin 1 to Pin 15 (digital common) and the $+V_{CC}$ and $-V_{EE}$ pins should be decoupled directly to analog common (Pin 9). A suitable decoupling capacitor is a 4.7 μF tantalum type in parallel with a 0.1 μF disc ceramic type.

Circuit layout should attempt to locate the AD574A, associated analog input circuitry, and interconnections as far as possible from logic circuitry. For this reason, the use of wire-wrap circuit construction is not recommended. Careful printed circuit construction is preferred.

## GROUNDING CONSIDERATIONS

The analog common at Pin 9 is the ground reference point for the internal reference and is thus the "high quality" ground for the AD574A; it should be connected directly to the analog reference point of the system. In order to achieve all of the high accuracy performance available from the AD574A in an environment of high digital noise content, the analog and digital commons should be connected together at the package. In some situations, the digital common at Pin 15 can be connected to the most convenient ground reference point; analog power return is preferred.

## UNIPOLAR RANGE CONNECTIONS FOR THE AD574A

The AD574A contains all the active components required to perform a complete 12-bit A/D conversion. Thus, for most situations, all that is necessary is connection of the power supplies (+5 V, +12 V/+15 V and –12 V/–15 V), the analog input, and the conversion initiation command, as discussed on the next page. Analog input connections and calibration are easily accomplished; the unipolar operating mode is shown in Figure 4.

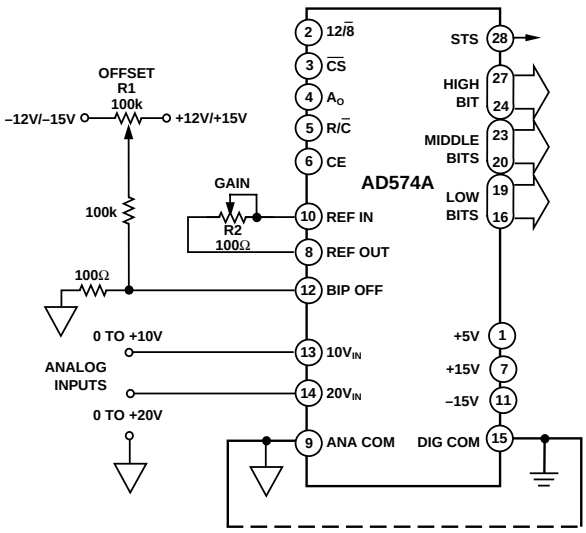



*Figure 4. Unipolar Input Connections*

All of the thin-film application resistors of the AD574A are trimmed for absolute calibration. Therefore, in many applications, no calibration trimming will be required. The absolute accuracy for each grade is given in the specification tables. For example, if no trims are used, the AD574AK guarantees ±1 LSB max zero offset error and ±0.25% (10 LSB) max full-scale error. (Typical full-scale error is ±2 LSB.) If the offset trim is not required, Pin 12 can be connected directly to Pin 9; the two resistors and trimmer for Pin 12 are then not needed. If the full-scale trim is not needed, a 50 Ω ± 1% metal film resistor should be connected between Pin 8 and Pin 10.

The analog input is connected between Pin 13 and Pin 9 for a 0 V to +10 V input range, between 14 and Pin 9 for a 0 V to +20 V input range. The AD574A easily accommodates an input signal beyond the supplies. For the 10 volt span input, the LSB has a nominal value of 2.44 mV; for the 20 volt span, 4.88 mV. If a 10.24 V range is desired (nominal 2.5 mV/bit), the gain trimmer (R2) should be replaced by a 50 Ω resistor, and a 200 Ω trimmer inserted in series with the analog input to Pin 13 for a full-scale range of 20.48 V (5 mV/bit), use a 500 Ω trimmer into Pin 14. The gain trim described below is now done with these trimmers. The nominal input impedance into Pin 13 is 5 kΩ, and 10 kΩ into Pin 14.

## UNIPOLAR CALIBRATION

The AD574A is intended to have a nominal 1/2 LSB offset so that the exact analog input for a given code will be in the middle of that code (halfway between the transitions to the codes above and below it). Thus, the first transition (from 0000 0000 0000 to 0000 0000 0001) will occur for an input level of +1/2 LSB (1.22 mV for 10 V range).

If Pin 12 is connected to Pin 9, the unit will behave in this manner, within specifications. If the offset trim (R1) is used, it should be trimmed as above, although a different offset can be set for a particular system requirement. This circuit will give approximately ±15 mV of offset trim range.

The full-scale trim is done by applying a signal 1 1/2 LSB below the nominal full scale (9.9963 for a 10 V range). Trim R2 to give the last transition (1111 1111 1110 to 1111 1111 1111).

**BIPOLAR OPERATION**

The connections for bipolar ranges are shown in Figure 5. Again, as for the unipolar ranges, if the offset and gain specifications are sufficient, one or both of the trimmers shown can be replaced by a 50 Ω ± 1% fixed resistor. Bipolar calibration is similar to unipolar calibration. First, a signal 1/2 LSB above negative full scale (–4.9988 V for the ±5 V range) is applied and R1 is trimmed to give the first transition (0000 0000 0000 to 0000 0000 0001). Then a signal 1 1/2 LSB below positive full scale (+4.9963 V the ±5 V range) is applied and R2 trimmed to give the last transition (1111 11111110 to 1111 1111 1111).

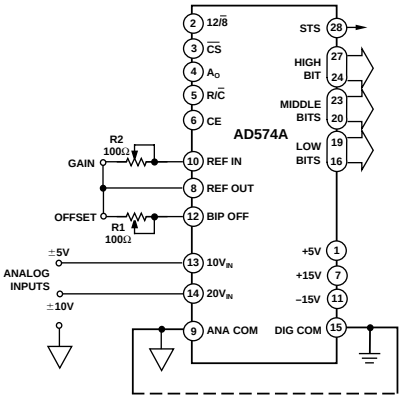


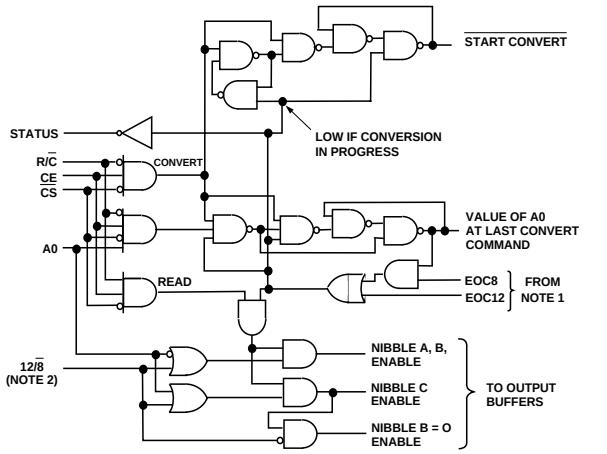

*Figure 5. Bipolar Input Connections*

**CONTROL LOGIC**

The AD574A contains on-chip logic to provide conversion initiation and data read operations from signals commonly available in microprocessor systems. Figure 6 shows the internal logic circuitry of the AD574A.

The control signals CE, $\overline{CS}$, and R/$\overline{C}$ control the operation of the converter. The state of R/$\overline{C}$ when CE and $\overline{CS}$ are both asserted determines whether a data read (R/$\overline{C}$ = 1) or a convert (R/$\overline{C}$ = 0) is in progress. The register control inputs $A_0$ and 12/$\overline{8}$ control conversion length and data format. The $A_0$ line is usually tied to the least significant bit of the address bus. If a conversion is started with $A_0$ low, a full 12-bit conversion cycle is initiated. If $A_0$ is high during a convert start, a shorter 8-bit conversion cycle results. During data read operations, $A_0$ determines whether the three-state buffers containing the 8 MSBs of the conversion result ($A_0$ = 0) or the 4 LSBs ($A_0$ = 1) are enabled. The 12/$\overline{8}$ pin determines whether the output data is to be organized as two 8-bit words (12/$\overline{8}$ tied to DIGITAL COMMON) or a single 12-bit word (12/$\overline{8}$ tied to $V_{LOGIC}$). The 12/$\overline{8}$ pin is not TTL-compatible and must be hard-wired to either $V_{LOGIC}$ or DIGITAL COMMON. In the 8-bit mode, the byte addressed when $A_0$ is high contains the 4 LSBs from the conversion followed by four trailing zeroes. This organization allows the data lines to be overlapped for direct interface to 8-bit buses without the need for external three-state buffers.

It is not recommended that $A_0$ change state during a data read operation. Asymmetrical enable and disable times of the three-state buffers could cause internal bus contention resulting in potential damage to the AD574A.




NOTE 1: WHEN $\overline{START}$ $\overline{CONVERT}$ GOES LOW, THE EOC (END OF CONVERSION) SIGNALS GO LOW. EOC8 RETURNS HIGH AFTER AN 8-BIT CONVERSION CYCLE IS COMPLETE, AND EOC12 RETURNS HIGH WHEN ALL 12-BITS HAVE BEEN CONVERTED. THE EOC SIGNALS PREVENT DATA FROM BEING READ DURING CONVERSIONS.

NOTE 2: 12/$\overline{8}$ IS NOT A TTL-COMPATABLE INPUT AND SHOULD ALWAYS BE WIRED DIRECTLY TO $V_{LOGIC}$ OR DIGITAL COMMON.

*Figure 6. AD574A Control Logic*

An output signal, STS, indicates the status of the converter. STS goes high at the beginning of a conversion and returns low when the conversion cycle is complete.

**Table I. AD574A Truth Table**

| CE | $\overline{CS}$ | R/$\overline{C}$ | 12/$\overline{8}$ | $A_0$ | Operation |
|---|---|---|---|---|---|
| 0 | X | X | X | X | None |
| X | 1 | X | X | X | None |
| 1 | 0 | 0 | X | 0 | Initiate 12-Bit Conversion |
| 1 | 0 | 0 | X | 1 | Initiate 8-Bit Conversion |
| 1 | 0 | 1 | Pin 1 | X | Enable 12-Bit Parallel Output |
| 1 | 0 | 1 | Pin 15 | 0 | Enable 8 Most Significant Bits |
| 1 | 0 | 1 | Pin 15 | 1 | Enable 4 LSBs + 4 Trailing Zeroes |

**TIMING**

The AD574A is easily interfaced to a wide variety of microprocessors and other digital systems. The following discussion of the timing requirements of the AD574A control signals should provide the system designer with useful insight into the operation of the device.

**Table II. Convert Start Timing—Full Control Mode**

| Symbol | Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|---|
| $t_{DSC}$ | STS Delay from CE | | | 400 | ns |
| $t_{HEC}$ | CE Pulse Width | 300 | | | ns |
| $t_{SSC}$ | $\overline{CS}$ to CE Setup | 300 | | | ns |
| $t_{HSC}$ | $\overline{CS}$ Low During CE High | 200 | | | ns |
| $t_{SRC}$ | R/$\overline{C}$ to CE Setup | 250 | | | ns |
| $t_{HRC}$ | R/$\overline{C}$ Low During CE High | 200 | | | ns |
| $t_{SAC}$ | $A_0$ to CE Setup | 0 | | | ns |
| $t_{HAC}$ | $A_0$ Valid During CE High | 300 | | | ns |
| $t_C$ | Conversion Time | | | | |
| | 8-Bit Cycle | 10 | | 24 | µs |
| | 12-Bit Cycle | 15 | | 35 | µs |

Figure 7 shows a complete timing diagram for the AD574A convert start operation. R/$\overline{C}$ should be low before both CE and $\overline{CS}$ are asserted; if R/$\overline{C}$ is high, a read operation will momentarily occur, possibly resulting in system bus contention. Either CE or $\overline{CS}$ may be used to initiate a conversion; however, use of CE is recommended since it includes one less propagation delay than $\overline{CS}$ and is the faster input. In Figure 7, CE is used to initiate the conversion.

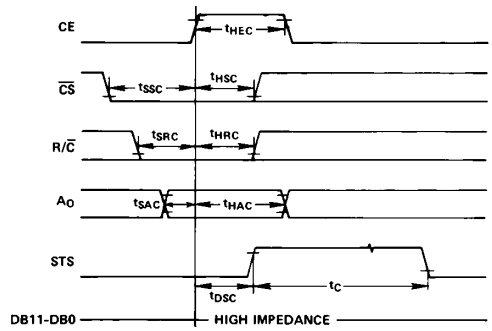



*Figure 7. Convert Start Timing*

Once a conversion is started and the STS line goes high, convert start commands will be ignored until the conversion cycle is complete. The output data buffers cannot be enabled during conversion.

Figure 8 shows the timing for data read operations. During data read operations, access time is measured from the point where CE and R/$\overline{C}$ both are high (assuming $\overline{CS}$ is already low). If $\overline{CS}$ is used to enable the device, access time is extended by 100 ns.

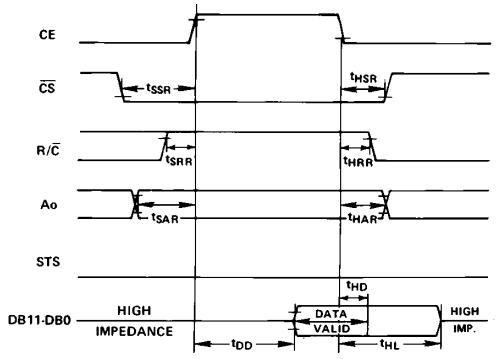


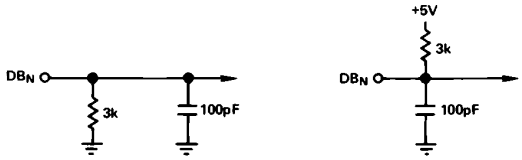

*Figure 8. Read Cycle Timing*

In the 8-bit bus interface mode (12/$\overline{8}$ input wired to DIGITAL COMMON), the address bit, $A_O$, must be stable at least 150 ns prior to $\overline{CE}$ going high and must remain stable during the entire read cycle. If $A_O$ is allowed to change, damage to the AD574A output buffers may result.

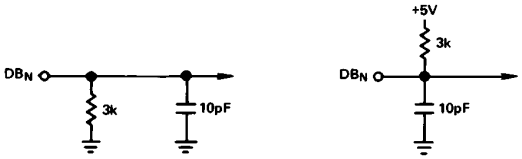
**Table III. Read Timing—Full Control Mode**

| Symbol | Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|---|
| $t_{DD}$[1] | Access Time (from CE) | | | 200 | ns |
| $t_{HD}$ | Data Valid After CE Low | 25 | | | ns |
| $t_{HL}$[2] | Output Float Delay | | | 100 | ns |
| $t_{SSR}$ | $\overline{CS}$ to CE Setup | 150 | | | ns |
| $t_{SRR}$ | R/$\overline{C}$ to CE Setup | 0 | | | ns |
| $t_{SAR}$ | $A_O$ to CE Setup | 150 | | | ns |
| $t_{HSR}$ | $\overline{CS}$ Valid After CE Low | 50 | | | ns |
| $t_{HRR}$ | R/$\overline{C}$ High After CE Low | 0 | | | ns |
| $t_{HAR}$ | $A_O$ Valid After CE Low | 50 | | | ns |

NOTES
[1]$t_{DD}$ is measured with the load circuit of Figure 9 and defined as the time required for an output to cross 0.4 V or 2.4 V.
[2]$t_{HL}$ is defined as the time required for the data lines to change 0.5 V when loaded with the circuit of Figure 10.




*a. High-Z to Logic 1 b. High-Z to Logic 0*
*Figure 9. Load Circuit for Access Time Test*




*a. Logic 1 to High-Z b. Logic 0 to High-Z*
*Figure 10. Load Circuit for Output Float Delay Test*

**"STAND-ALONE" OPERATION**

The AD574A can be used in a "stand-alone" mode, which is useful in systems with dedicated input ports available and thus not requiring full bus interface capability.

In this mode, CE and 12/$\overline{8}$ are wired high, $\overline{CS}$ and $A_O$ are wired low, and conversion is controlled by R/$\overline{C}$. The three-state buffers are enabled when R/$\overline{C}$ is high and a conversion starts when R/$\overline{C}$ goes low. This allows two possible control signals—a high pulse or a low pulse. Operation with a low pulse is shown in Figure 11. In this case, the outputs are forced into the high impedance state in response to the falling edge of R/$\overline{C}$ and return

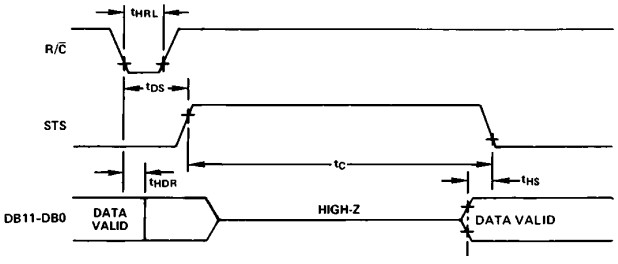



*Figure 11. Low Pulse for R/$\overline{C}$—Outputs Enabled After Conversion*

to valid logic levels after the conversion cycle is completed. The STS line goes high 600 ns after R/$\overline{C}$ goes low and returns low 300 ns after data is valid.

If conversion is initiated by a high pulse as shown in Figure 12, the data lines are enabled during the time when R/$\overline{C}$ is high. The falling edge of R/$\overline{C}$ starts the next conversion, and the data lines return to three-state (and remain three-state) until the next high pulse of R/$\overline{C}$.

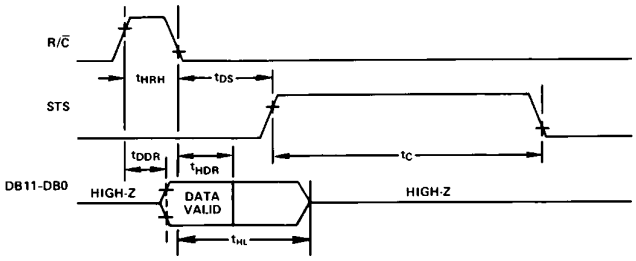



*Figure 12. High Pulse for R/$\overline{C}$—Outputs Enabled While R/$\overline{C}$ High, Otherwise High-Z*

**Table IV. Stand-Alone Mode Timing**

| Symbol | Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|---|
| $t_{HRL}$ | Low R/$\overline{C}$ Pulse Width | 250 | | | ns |
| $t_{DS}$ | STS Delay from R/$\overline{C}$ | | | 600 | ns |
| $t_{HDR}$ | Data Valid After R/$\overline{C}$ Low | 25 | | | ns |
| $t_{HL}$ | Output Float Delay | | | 150 | ns |
| $t_{HS}$ | STS Delay After Data Valid | 300 | | 1000 | ns |
| $t_{HRH}$ | High R/$\overline{C}$ Pulse Width | 300 | | | ns |
| $t_{DDR}$ | Data Access Time | | | 250 | ns |

Usually the low pulse for R/$\overline{C}$ stand-alone mode will be used. Figure 13 illustrates a typical stand-alone configuration for 8086 type processors. The addition of the 74F/S374 latches improves bus access/release times and helps minimize digital feedthrough to the analog portion of the converter.

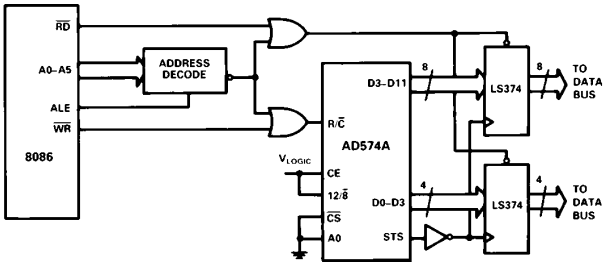



*Figure 13. 8086 Stand-Alone Configuration*

## INTERFACING THE AD574A TO MICROPROCESSORS

The control logic of the AD574A makes direct connection to most microprocessor system buses possible. While it is impossible to describe the details of the interface connections for every microprocessor type, several representative examples will be described here.

## GENERAL A/D CONVERTER INTERFACE CONSIDERATIONS

A typical A/D converter interface routine involves several operations. First, a write to the ADC address initiates a conversion. The processor must then wait for the conversion cycle to complete, since most ADCs take longer than one instruction cycle to complete a conversion. Valid data can, of course, only be read after the conversion is complete. The AD574A provides an output signal (STS) which indicates when a conversion is in progress. This signal can be polled by the processor by reading it through an external three-state buffer (or other input port). The STS signal can also be used to generate an interrupt upon completion of conversion, if the system timing requirements are critical (bear in mind that the maximum conversion time of the AD574A is only 35 microseconds) and the processor has other tasks to perform during the ADC conversion cycle. Another possible time-out method is to assume that the ADC will take 35 microseconds to convert, and insert a sufficient number of "do-nothing" instructions to ensure that 35 microseconds of processor time is consumed.

Once it is established that the conversion is finished, the data can be read. In the case of an ADC of 8-bit resolution (or less), a single data read operation is sufficient. In the case of converters with more data bits than are available on the bus, a choice of data formats is required, and multiple read operations are needed. The AD574A includes internal logic to permit direct interface to 8-bit or 16-bit data buses, selected by connection of the 12/$\overline{8}$ input. In 16-bit bus applications (12/$\overline{8}$ high) the data lines (DB11 through DB0) may be connected to either the 12 most significant or 12 least significant bits of the data bus. The remaining four bits should be masked in software. The interface to an 8-bit data bus (12/$\overline{8}$ low) is done in a left-justified format. The even address (A0 low) contains the 8 MSBs (DB11 through DB4). The odd address (A0 high) contains the 4 LSBs (DB3 through DB0) in the upper half of the byte, followed by four trailing zeroes, thus eliminating bit masking instructions.

It is not possible to rearrange the AD574A data lines for right justified 8-bit bus interface.

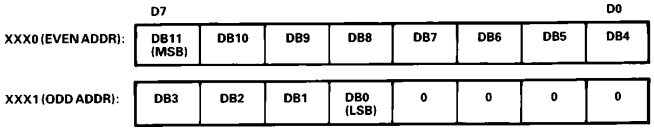



*Figure 14. AD574A Data Format for 8-Bit Bus*

## SPECIFIC PROCESSOR INTERFACE EXAMPLES

### Z-80 System Interface

The AD574A may be interfaced to the Z-80 processor in an I/O or memory mapped configuration. Figure 15 illustrates an I/O or mapped configuration. The Z-80 uses address lines A0–A7 to decode the I/O port address.

An interesting feature of the Z-80 is that during I/O operations a single wait state is automatically inserted, allowing the AD574A to be used with Z-80 processors having clock speeds up to 4MHz. For applications faster than 4 MHz use the wait state generator in Figure 16. In a memory mapped configuration the AD574A may be interfaced to Z-80 processors with clock speeds of up to 2.5 MHz.

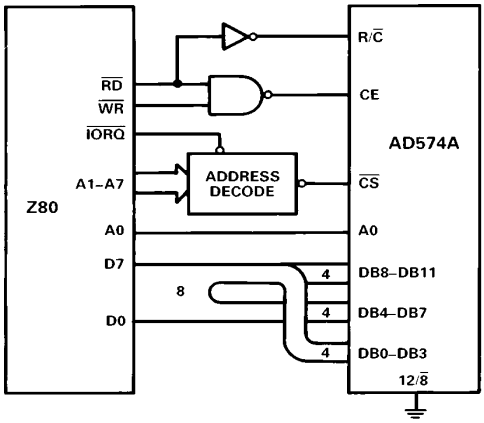



*Figure 15. Z80—AD574A Interface*

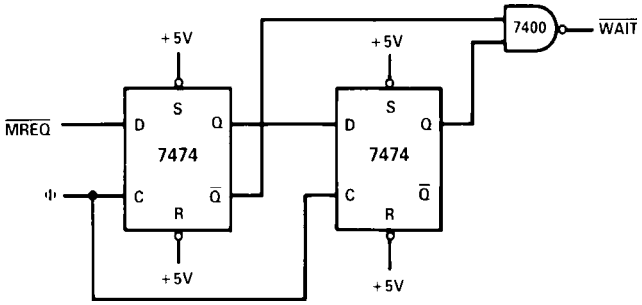



*Figure 16. Wait State Generator*

**IBM PC Interface**

The AD574A appears in Figure 17 interfaced to the 4 MHz 8088 processor of an IBM PC. Since the device resides in I/O space, its address is decoded from only the lower ten address lines and must be gated with AEN (active low) to mask out internal DMA cycles which use the same I/O address space. This active low signal is applied to $\overline{CS}$. $\overline{IOR}$ and $\overline{IOW}$ are used to initiate the conversion and read, and are gated together to drive the chip enable, CE. Because the data bus width is limited to 8 bits, the AD574A data resides in two adjacent addresses selected by A0.

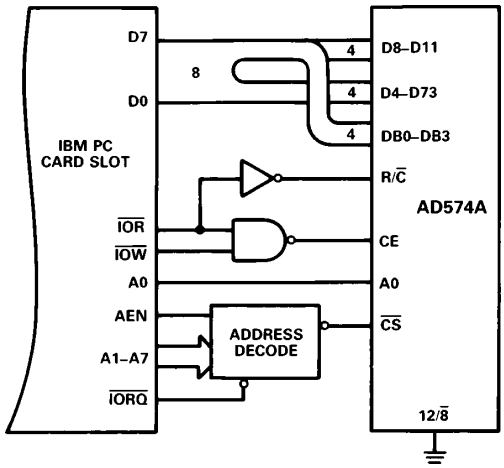



*Figure 17. IBM PC—AD574A Interface*

Note: Due to the large number of options that may be installed in the PC, the I/O bus loading should be limited to one Schottky TTL load. Therefore, a buffer/driver should be used when interfacing more than two AD574As to the I/O bus.

**8086 Interface**

The data mode select pin (12/$\overline{8}$) of the AD574A should be connected to $V_{LOGIC}$ to provide a 12-bit data output. To prevent possible bus contention, a demultiplexed and buffered address/data bus is recommended. In the cases where the 8-bit short conversion cycle is not used, A0 should be tied to digital common. Figure 18 shows a typical 8086 configuration.

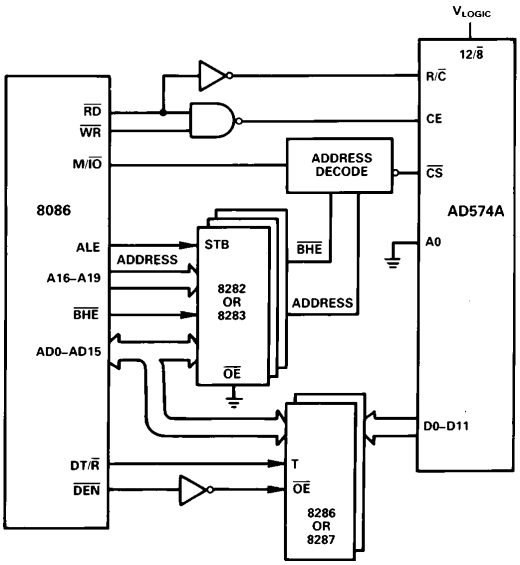



*Figure 18. 8086—AD574A with Buffered Bus Interface*

For clock speeds greater than 4 MHz wait state insertion similar to Figure 16 is recommended to ensure sufficient CE and R/$\overline{C}$ pulse duration.

The AD574A can also be interfaced in a stand-alone mode (see Figure 13). A low going pulse derived from the 8086's $\overline{WR}$ signal logically ORed with a low address decode starts the conversion. At the end of the conversion, STS clocks the data into the three-state latches.

**68000 Interface**

The AD574, when configured in the stand-alone mode, will easily interface to the 4 MHz version of the 68000 microprocessor. The 68000 R/$\overline{W}$ signal combined with a low address decode initiates conversion. The $\overline{UDS}$ or $\overline{LDS}$ signal, with the decoded address, generates the $\overline{DTACK}$ input to the processor, latching in the AD574A's data. Figure 19 illustrates this configuration.

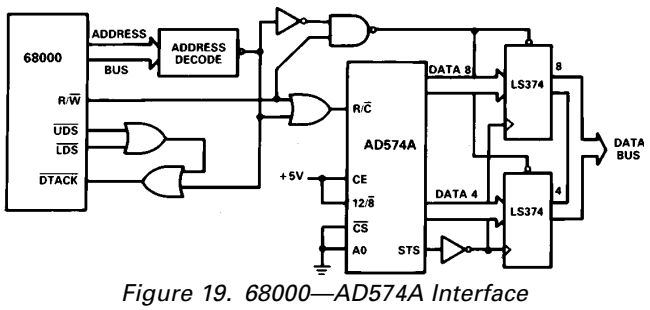
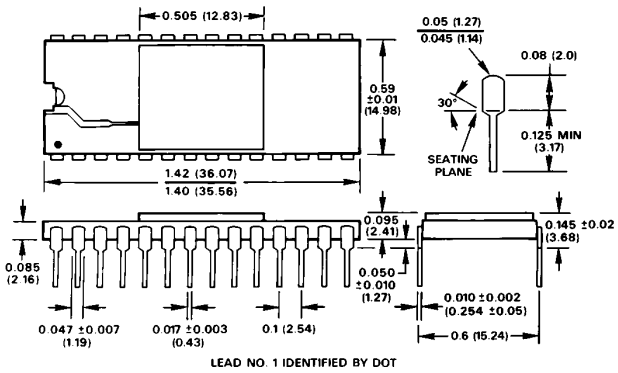



*Figure 19. 68000—AD574A Interface*

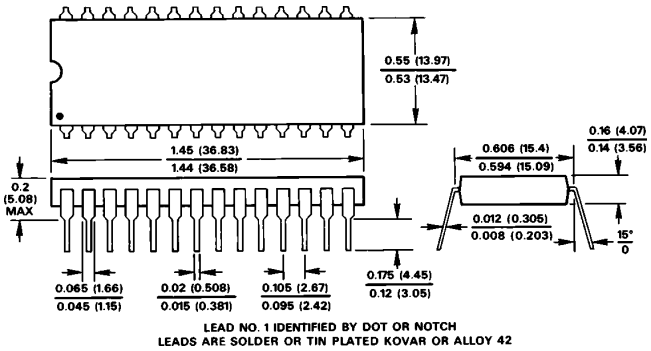
## OUTLINE DIMENSIONS

Dimensions shown in inches and (mm).

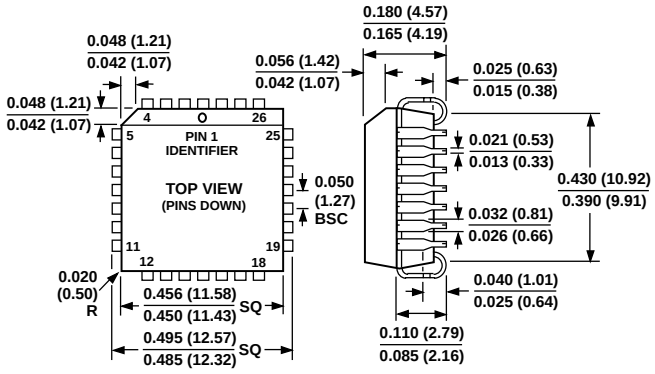
### 28-Pin Ceramic DIP Package (D-28)



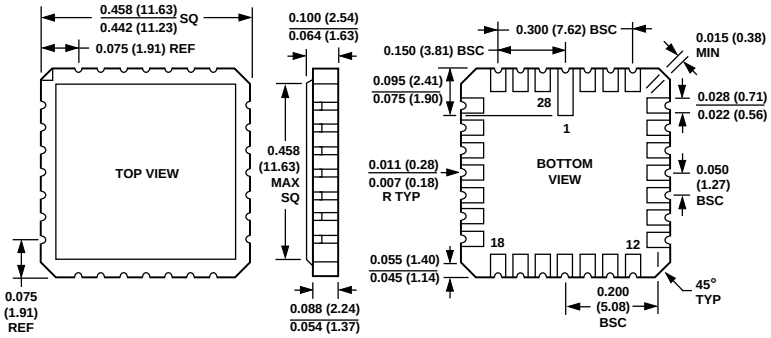

### 28-Lead Plastic DIP Package (N-28A)




### 28-Terminal PLCC Package (P-28A)




### 28–Terminal LCC Package (E-28A)




C704d–10–8/88

PRINTED IN U.S.A.

## C.3 Cypress AN2131QC Technical Reference

# Chapter 13 EZ-USB AC/DC Parameters

## 13.1 Electrical Characteristics

### 13.1.1 Absolute Maximum Ratings

Storage Temperature . . . . . -65°C to +150°C
Ambient Temperature Under Bias . . . . . -40°C to +85°C
Supply Voltage to Ground Potential . . . . . -0.5V to +4.0V
DC Input Voltage to Any Pin . . . . . -0.5V to +5.8V

### 13.1.2 Operating Conditions

Ta (Ambient Temperature Under Bias) . . . . . 0°C to +70°C
Supply Voltage . . . . . +3.0V to +3.6V
Ground Voltage . . . . . 0V
$F_{osc}$ (Oscillator or Crystal Frequency) . . . . . 12 MHz +/- 0.25%

### 13.1.3 DC Characteristics

*Table 13-1. DC Characteristics*

| Symbol | Parameter | Condition | Min | Typ | Max | Unit | Notes |
|---|---|---|---|---|---|---|---|
| $V_{CC}$ | Supply Voltage | | 3.0 | | 3.6 | V | |
| $V_{IH}$ | Input High Voltage | | 2.0 | | 5.25 | V | |
| $V_{IL}$ | Input Low Voltage | | -0.5 | | 0.8 | V | |
| $I_I$ | Input Leakage Current | $0 < V_{IN} < V_{CC}$ | | | ± 10 | µA | Current is higher (-70µA typical) in the switching range between $V_{IL\text{-}MAX}$ and $V_{IH\text{-}MIN}$ |
| $V_{OH}$ | Output High Voltage | $I_{OUT}$ = 1.6 mA | 2.4 | | | V | |
| $V_{OL}$ | Output Low Voltage | $I_{OUT}$ = -1.6 mA | | | 0.4 | V | |
| $C_{IN}$ | Input Pin Capacitance | | | | 10 | pF | |
| $I_{SUSP}$ | Suspend Current | | | 110 | | µA | |
| $I_{CC}$ | Supply Current | 8051 running, connected to USB | | | 50 | mA | |

## C.4 Interrupts

The EZ-USB's interrupt architecture is an enhanced and expanded version of the standard 8051's. The EZ-USB responds to the interrupts shown in Table C-15; interrupt sources that are not present in the standard 8051 are shown in **bold** type.

*Table C-15 EZ-USB Interrupts*

| EZ-USB Interrupt | Source | Interrupt Vector | Natural Priority |
|---|---|---|---|
| IE0 | INT0 Pin | 0x0003 | 1 |
| TF0 | Timer 0 Overflow | 0x000B | 2 |
| IE1 | INT1 Pin | 0x0013 | 3 |
| TF1 | Timer 1 Overflow | 0x001B | 4 |
| RI_0 & TI_0 | USART0 Rx & Tx | 0x0023 | 5 |
| **TF2** | **Timer 2 Overflow** | 0x002B | 6 |
| **Resume** | **$\overline{\text{WAKEUP}}$ or USB Resume** | 0x0033 | 0 (Highest Priority) |
| **RI_1 & TI_1** | **USART1 Rx & Tx** | 0x003B | 7 |
| **USBINT** | **USB** | 0x0043 | 8 |
| **I²CINT** | **I²C Bus** | 0x004B | 9 |
| **IE4** | **INT4 Pin** | 0x0053 | 10 |
| **IE5** | **INT5 Pin** | 0x005B | 11 |
| **IE6** | **INT6 Pin** | 0x0063 | 12 |

The **Natural Priority** column in Table C-15 shows the EZ-USB interrupt priorities. The EZ-USB can assign each interrupt to a high or low priority group; priorities within the groups are resolved using the natural priorities.

## C.5 Interrupt-Control SFRs

The following SFRs are associated with interrupt control:

- IE - SFR 0xA8 (Table C-16)
- IP - SFR 0xB8 (Table C-17)
- EXIF - SFR 0x91 (Table C-18)

- EICON - SFR 0xD8 (Table C-19)
- EIE - SFR 0xE8 (Table C-20)
- EIP - SFR 0xF8 (Table C-21)

The IE and IP SFRs provide interrupt enable and priority control for the standard interrupt unit, as with the standard 8051. Additionally, these SFRs provide control bits for the Serial Port 1 interrupt.

The EXIF, EICON, EIE and EIP Registers provide flags, enable control, and priority control.

*Table C-16 IE Register — SFR 0xA8*

| Bit | Function |
|---|---|
| IE.7 | **EA** - Global interrupt enable. Controls masking of all interrupts except USB wakeup (resume). EA = 0 disables all interrupts except USB wakeup. When EA = 1, interrupts are enabled or masked by their individual enable bits. |
| IE.6 | **ES1** - Enable Serial Port 1 interrupt. ES1 = 0 disables Serial port 1 interrupts (TI_1 and RI_1). ES1 = 1 enables interrupts generated by the TI_1 or RI_1 flag. |
| IE.5 | **ET2** - Enable Timer 2 interrupt. ET2 = 0 disables Timer 2 interrupt (TF2). ET2=1 enables interrupts generated by the TF2 or EXF2 flag. |
| IE.4 | **ES0** - Enable Serial Port 0 interrupt. ES0 = 0 disables Serial Port 0 interrupts (TI_0 and RI_0). ES0=1 enables interrupts generated by the TI_0 or RI_0 flag. |
| IE.3 | **ET1** - Enable Timer 1 interrupt. ET1 = 0 disables Timer 1 interrupt (TF1). ET1=1 enables interrupts generated by the TF1 flag. |
| IE.2 | **EX1** - Enable external interrupt 1. EX1 = 0 disables external interrupt 1 (INT1). EX1=1 enables interrupts generated by the $\overline{\text{INT1}}$ pin. |
| IE.1 | **ET0** - Enable Timer 0 interrupt. ET0 = 0 disables Timer 0 interrupt (TF0). ET0=1 enables interrupts generated by the TF0 flag. |
| IE.0 | **EX0** - Enable external interrupt 0. EX0 = 0 disables external interrupt 0 (INT0). EX0=1 enables interrupts generated by the $\overline{\text{INT0}}$ pin. |

*Table C-17 IP Register — SFR 0xB8*

| Bit | Function |
|---|---|
| IP.7 | **Reserved**. Read as 1. |
| IP.6 | **PS1** - Serial Port 1 interrupt priority control. PS1 = 0 sets Serial Port 1 interrupt (TI_1 or RI_1) to low priority. PS1 = 1 sets Serial port 1 interrupt to high priority. |
| IP.5 | **PT2** - Timer 2 interrupt priority control. PT2 = 0 sets Timer 2 interrupt (TF2) to low priority. PT2 = 1 sets Timer 2 interrupt to high priority. |
| IP.4 | **PS0** - Serial Port 0 interrupt priority control. PS0 = 0 sets Serial Port 0 interrupt (TI_0 or RI_0) to low priority. PS0 = 1 sets Serial Port 0 interrupt to high priority. |
| IP.3 | **PT1** - Timer 1 interrupt priority control. PT1 = 0 sets Timer 1 interrupt (TF1) to low priority. PT1 = 1 sets Timer 1 interrupt to high priority. |
| IP.2 | **PX1** - External interrupt 1 priority control. PX1 = 0 sets external interrupt 1 (INT1) to low priority. PT1 = 1 sets external interrupt 1 to high priority. |
| IP.1 | **PT0** - Timer 0 interrupt priority control. PT0 = 0 sets Timer 0 interrupt (TF0) to low priority. PT0 = 1 sets Timer 0 interrupt to high priority. |
| IP.0 | **PX0** - External interrupt 0 priority control. PX0 = 0 sets external interrupt 0 (INT0) to low priority. PX0 = 1 sets external interrupt 0 to high priority. |

*Table C-18 EXIF Register — SFR 0x91*

| Bit | Function |
|---|---|
| EXIF.7 | **IE5** - External Interrupt 5 flag. IE5 = 1 indicates a falling edge was detected at the INT5 pin. IE5 must be cleared by software. Setting IE5 in software generates an interrupt, if enabled. |
| EXIF.6 | **IE4** - External Interrupt 4 flag. The "INT4" interrupt indicates that a rising edge was detected at the INT4 pin. IE4 must be cleared by software. Setting IE4 in software generates an interrupt, if enabled. |
| EXIF.5 | **I2CINT** - I²C Bus Interrupt flag. I2CINT = 1 indicates an I²C Bus interrupt. I2CINT must be cleared by software. Setting I2CINT in software generates an interrupt, if enabled. |
| EXIF.4 | **USBINT** - USB Interrupt flag. USBINT = 1 indicates an USB interrupt. USBINT must be cleared by software. Setting USBINT in software generates an interrupt, if enabled. |
| EXIF.3 | **Reserved**. Read as 1. |
| EXIF.2-0 | **Reserved**. Read as 0. |

*Table C-19 EICON Register — SFR 0xD8*

| Bit | Function |
|---|---|
| EICON.7 | **SMOD1** - Serial Port 1 baud rate doubler enable. When SMOD1 = 1, the baud rate for Serial Port 1 is doubled. |
| EICON.6 | **Reserved**. Read as 1. |
| EICON.5 | **ERESI** - Enable Resume interrupt. ERESI = 0 disables the Resume interrupt. ERESI = 1 enables interrupts generated by the resume event. |
| EICON.4 | **RESI** - Wakeup interrupt flag. RESI = 1 indicates a false-to-true transition was detected at the WAKEUP pin, or that USB activity has resumed from the suspended state. RESI must be cleared by software before exiting the interrupt service routine, otherwise the interrupt will immediately be reasserted. Setting RESI = 1 in software generates a wakeup interrupt, if enabled. |
| EICON.3 | **INT6** - External interrupt 6. When INT6 = 1, the INT6 pin has detected a low to high transition. INT6 must be cleared by software. Setting this bit in software generates an INT6 interrupt, if enabled. |
| EICON.2-0 | **Reserved**. Read as 0. |

*Table C-20 EIE Register — SFR 0xE8*

| Bit | Function |
|---|---|
| EIE.7-5 | Reserved. Read as 1. |
| EIE.4 | EX6 - Enable external interrupt 6. EX6 = 0 disables external interrupt 6 (INT6). EX6 = 1 enables interrupts generated by the INT6 pin. |
| EIE.3 | EX5 - Enable external interrupt 5. EX5 = 0 disables external interrupt 5 (INT5). EX5 = 1 enables interrupts generated by the $\overline{\text{INT5}}$ pin. |
| EIE.2 | EX4 - Enable external interrupt 4. EX4 = 0 disables external interrupt 4 (INT4). EX4 = 1 enables interrupts generated by the INT4 pin. |
| EIE.1 | EI2C - Enable I²C Bus interrupt (I2CINT). EI2C = 0 disables the I²C Bus interrupt. EI2C = 1 enables interrupts generated by the I²C Bus controller. |
| EIE.0 | EUSB - Enable USB interrupt (USBINT). EUSB = 0 disables USB interrupts. EUSB = 1 enables interrupts generated by the USB Interface. |