DESIGN AND IMPLEMENTATION OF AN EMBEDDED SYSTEM ARCHITECTURE FOR MULTICHANNEL BIOMEDICAL SIGNAL ACQUSITION

by

İsmail Hakkı KÖSE

February 2011

DESIGN AND IMPLEMENTATION OF AN EMBEDDED SYSTEM ARCHITECTURE FOR MULTICHANNEL BIOMEDICAL SIGNAL ACQUSITION

by

İsmail Hakkı KÖSE

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Electronics Engineering

February 2011
Istanbul, Turkey

APPROVAL PAGE

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Assoc. Prof. Onur TOKER
Head of Department

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

Assoc. Prof. Onur TOKER
Supervisor

Examining Committee Members

Assoc. Prof. Onur TOKER …….

Prof. Dr. Sadık KARA …….

Prof. Dr. Halil Rıdvan ÖZ …….

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assoc. Prof. Nurullah ARSLAN
Director

DESIGN AND IMPLEMENTATION OF AN EMBEDDED SYSTEM ARCHITECTURE FOR MULTICHANNEL BIOMEDICAL SIGNAL ACQUSITION

İsmail Hakkı KÖSE

M. S. Thesis –Electrical Electronics Engineering
February 2011

Supervisor: Assoc. Prof. Onur TOKER

ABSTRACT

In this thesis, we design and implement modular, flexible and convenient embedded system architecture, which makes possible with dynamic number of channels for biomedical signal acquisition. Firstly, it is used system with electronic components to be ensured easily and open-source coded application. Secondly, it is cheaper than commercial devices, and it is preferable with less costs values in biomedical researches. Computer connection interface is based on USB HID class that is supported with all modern operating systems of today. With the developed computer based user interface and the embedded system architecture was realized biomedical signal collection up to six channels.

Keywords: USB HID, Multichannel Biosignal Acquisition, Useful and Modular Data Collection system

ÇOK KANALLI BİYOMEDİKAL SİNYAL ALICI İÇİN GÖMÜLÜ SİSTEM MİMARİSİ TASARIMI VE GERÇEKLEŞTİRİLMESİ

İsmail Hakkı KÖSE

Yüksek Lisans Tezi – Elektrik Elektronik Mühendisliği
Şubat 2011

Tez Yöneticisi: Doç. Dr. Onur TOKER

ÖZ

Gerçekleştirilen bu sistem, çoklu biyomedikal sinyal toplama uygulaması için dinamik kanal sayısı olanağını sunan, modüler, esnek ve kullanışlı bir gömülü sistem mimarisi önermektedir. Kolay temin edilebilir elektronik bileşenlerinin kullanımıyla ve açık kaynak kodlu bir uygulama olması yönüyle orijinal bir sistemdir. Ticari cihazlara göre oldukça ucuz olması ve biyomedikal araştırma maliyetlerini azaltması yönüyle önerilebilir bir sistemdir. Bilgisayarla bağlantı arayüzü olarak günümüzün bütün modern işletim sistemlerinin desteklediği USB'nin HID sınıfı üzerine temellendirilmiştir. Geliştirilen bilgisayar tabanlı kullanıcı arayüzü program ve gömülü sistem mimarisiyle 6 kanal'a kadar sinyal toplama uygulaması gerçekleştirilmiştir.

Anahtar Kelimeler: USB HID, Çok kanallı biyosinyal toplama, Kullanışlı ve modüler veri toplama sistemi

Dedicated to my parents and eldest brother

ACKNOWLEDGEMENT

First and foremost, I express sincere appreciation to my supervisor Assoc. Prof. Onur TOKER for his guidance, motivation, understanding, continuous support, enlightening experience, suggestions and comments through my thesis work. I would like to thank to Prof. Dr. Halil Rıdvan ÖZ for his valuable comments, suggestions, patience, comments and understanding. Sincere thanks are also extended to the committee members Prof. Dr. Sadık KARA for his valuable suggestions, motivations and comments. I am also grateful that they taught me the importance of patience.

I would like to thank Dr. Hüseyin KAVAS, Assist. Prof. Dr. Abdullah Said ERDOĞAN, Shahriar SHAMIL UULU, Uğur ÇELİK, Ali Kemal ŞAHİN, Ömer IŞIK, H. İbrahim ÇAKAR and Mustafa Selman YILDIRIM for their valuable suggestion and contributions. In addition, thank you to all my friends and colleagues at Fatih University, current and old, for their friendship and collaboration and companionship throughout this journey in graduate school life.

I want to thank my colleagues Yaşar USTAMEHMETOĞLU, Maxim SHYLOV and other member of IT department for their encouragement, friendship, collaborations and supports. A sincere thanks is also to the head of IT department Vahdet ŞAHİN for his support, understanding and patience.

At last but not least, I express my thanks and appreciation to my mother, my father, my eldest brother Süleyman and all family members for their understanding, motivation, patience and supporting me through all these years. I owe everything in my life to them.

To all of you, I thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION

μC	: Microcontroller
A	: ampere
ADC	: Analog Digital Converter
API	: Application Interface Program
ASIC	: Application System Integrated Circuits
B	: Byte
Bps	: bits per second
C18	: MPLAB C compiler for PIC18 CPUs
COTS	: Commercial-Off-The-Shelf
CPU	: Central Processing Unit
CUDA	: Compute Unified Device Architecture
DAC	: Digital Analog Converter
DAQ:	Data Acquisition
DAS	: Data Acquisition System
DSP	: Digital Signal Processor
ECG	: Electrocardiogram
EEG	: Electroencephalography
EMG	: Electromyography
EMIF	: External Memory IF
ENG	: Electroneurogram
F	: Farad
FIFO	: First Input First Output
FPGA	: Field Programmable Gate Array
GPL	: General Public License

GPU	: Graphic Processor Unit
HID	: Human Interface Device
Hz	: Hertz (cycles per second)
IP	: Intellectual Property
KHz	: Kilo Hertz
kSPS	: kilo Sample Per Second
kΩ	: Kilo Ohm
LED	: Light Emitted Diot
mA	: milliampere
MCU	: Microcontroller Control Unit
MHz	: Mega Hertz
MIPS	: Million Instructions per Second
ms	: millisecond
mV	: millivolt
NRZI	: Non Returning Zero Inverted
OOP	: Object Oriented Pascal
OOP	: Object Oriented Programming
OTG	: On The Go
PC	: Personal Computer
PCB	: Print Circuit Board
PID	: Product Identification
PLD	: Programmable Logic Device
PLL	: Phase Locked Loop
RAM	: Random Access Memory
ROM	: Read Only Memory
s	: second
SPAD	: Single Photon Avalanche Diode
UCON	: USB Control registers
UCFG	: USB Configuration registers

USTAT	: USB Transfer Status registers
UADDR	: USB Device Address registers
UEPn	: Endpoint Enable registers zero through fifteen
UPUEN	: USB On-Chip Pull-up Enable bit
USB	: Universal Serial Bus
USB–IF	: Universal Serial Bus Implementers Forum
V	: Volt
VID	: Vendor Identification
WAV	: Wave Form Audio
Ω	: Ohm

CHAPTER 1

INTRODUCTION

This study is aimed to develop flexible, modular and multichannel embedded system architecture for biomedical and biotechnology research and signal acquisition based projects. This system was implemented for multichannel signal collection studies. Many signal collector devices for biomedical researching may need to different filters such as Low Pass Filters, High Pass Filters, Notch Filters, Comb Filters and different amplifiers.

The connection interface to the PC was selected as USB, which is supported by major operating systems. To digitize signal received from human body, processor's internal unipolar 10-bit ADC was used. The scope of this study is digitizing bio signal by ADC, data transfer to the PC by USB HID communication and computer program, which manages independent signal acquisition devices, signal plotting and storing on the PC. As a general, biosignal's frequency is lower than 2 KHz. Therefore, for many signal acquisition studies in biomedical application, the sampling frequency, which is lower than 2 KHz, is sufficient. HID data transfer capacity is limited. The most important advantage of HID is supported by almost all operating systems without developing a driver for communication over USB. Moreover, it does not need installing new driver to the PC. Operating systems perform all processes without user settings.

Device controller program on the host computer side was developed in Delphi environment. During this time, JEDI component was used to implementing HID USB Class. JEDI is a software class which is open source and it hides details aspect of the operating system, it makes transparency against low level programming JEDI

component has supported a very important contribution to facilitating and accelerating the software development process.

The signal acquisition and device controller program on computer side was developed by using object oriented Pascal in Delphi environment. USB is a complex protocol and there are many details. Therefore, to ease developing program on computer side in short time, object oriented programming and using a class, which makes transparency to operating system drivers, is very important.

Data acquisition devices have very important role in biomedical researches, disease diagnosis and on time implementation of appropriate treatment methods. There are a large number of commercial products developed for data acquisition on the markets. Biomedical commercial data collecting devices are expensive, and it leads to development of low-cost, portable and modular data collecting devices. In biomedical research projects, the most expensive expending is ensuring the data acquisition units, sensors and software. In our project, the motivation is to achieve data acquisition and channel multiplexing for biomedical applications with low-cost.

Designing data acquisition system with serving for many needs in a unique form requires a lot of time and effort. This topic has many sub expertise working such as; embedded system design and software, analog circuit designs that can get biosignals with maximum efficiency, the digital signal processing, multithreading management of multiple data acquisition devices, USB communication system and computer software with user interface components.

The proposed and the target architecture in project is designing a system with dynamic channel number and compatible with all popular operating systems via using the connection method which is flexible, portable and known by everyone and combining signal processing and analog circuit design. We developed a lab-made firmware by using electronic component as easy as possible to be obtained and open sources. It allows to communicate with each device and get the data then offer a useful output as a graphical and data with user-friendly interface program.

The selected communication interface for DAQ devices management is USB-based HID interface. Be compatible with all operating systems, no need to install the

HID driver outside and easy multi-channel properties make practical use and applicability of it increasing.

CHAPTER 2

LITERATURE REVIEW

2.1. DATA ACQUISITION

Data acquisition is the process of gathering information about the physical conditions of real-world phenomena. This data is perceived as a special numerical data-collecting device and transferred to the computer by various communication interfaces. Scientists and engineers use this numerical data to their analysis. Analysis and inferences close to the truth is dependent on maximum level accuracy and minimum error. Gathering such a high quality data is only possible with usage of special data acquisition systems.

Generally, a data acquisition system is designed based on a host computer with variety hardware communication interfaced such as PCI, VME, USB, CAMAC buses, etc. These interfaces are involved an appropriate analog to digital converters (ADCs) to translate the acquisition signal to the numerical values. The major responsibility of the host computer is to present as a readability format on the computer screen, graphical presentations, some computations for signal processing, etc. (Aiello, 1998)

Data in real world does not directly perceive form a computer. These data are found in the various form of energy such as temperature, motion, pressure, distance, weight, or acoustic, electric, optical. The data in this form can be obtained by using an appropriate sensor, and related with time or position information, then can be recorded to the computer.

Transfer of the measurable physical event to the computer begins with its converting to the electrical signal. At this stage, the physical parameter is converted to

an electrical parameter such as resistance, current, voltage, capacity by using sensor or transducer. This electrical parameter's order is changed to the readable order of electrical equipment.

However, it is now in the form of continuous analog signal so it is not ready to be saved and transferred to the computer. A continuous signal can be recorded on the computer just in case that they have to be changed to time dependent discrete signals and to be digital. This conversion operation is provided by Analog to Digital Converters (ADC). The analog signals are usually converted to a digital format by the meaning of ADC in a few seconds. DAC is also used for converting a digital signal to the analog one.

Analog data with certain time intervals and precision is converted to a certain range of numerical values. The similarity of shape of resulting digital signal to the analog signal is dependent to the conversion frequency of continuous signal into discrete signal depends on the frequency of.

Figure 2.1 shows a digitized form of triangular-shaped state of an analog signal. As the digitalization frequency increases, the saw teeth go smaller, otherwise the signal's shape becomes noisier. (Austerlitz, 2002)

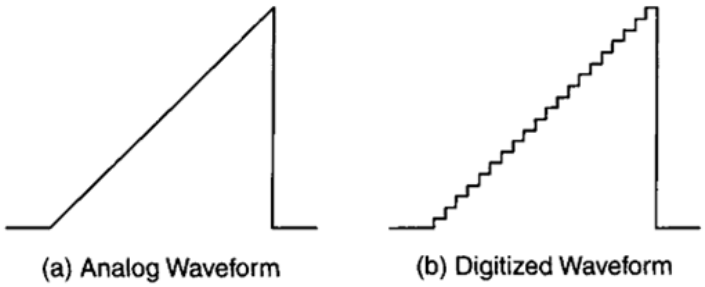


Figure 2. 1 Comparison of certain frequencies of analog signal to digital signal (Austerlitz, 2002)

To detect a signal, the signal data should be in the level of human perception. Therefore, the signal is strengthened and filtered to reduce the noise. As a result, some signal processing and analysis methods are used to make the signal a computer readable graphical or numerical form.

There are a lot of communication interface to transfer of collected data in human body to the computer. USB, Ethernet, PCI, serial and parallel port interfaces are widespread communication ways. Now, almost all of user-based desktops or business computers have USB, Ethernet, and PCI drivers. Data acquisition devices are made by using a variety of such interfaces.

By developing technology, production costs of computers are decreasing and usage in our society is pervading. Now, computers are not only in private laboratories but also everywhere. In 30 years ago, computers are much more slowly than today's computers. In addition, the cost is very much above the current prices. Day by day, electronic components are becoming smaller and their performances are increasing. Moreover, by the decreasing cost, performance is increasing. (Austerlitz, 2002)

Data acquisition units can be obtained by just computers connections with environmental hardware. Widely used ordinary desktop computers, laptop computers or tablet PCs are sufficient for biomedical data acquisition application. In fact, it is extremely decreased cost of data acquisition that even ordinary desktop computers have been seen for the high-cost industrial data acquisition applications. So industrial PC's are starting to be used for this aim. When the USB-based HID is the communication method of data acquisition application, type of used computer is becoming very important. Because almost all the leading operating system support the HID.

When deciding on the type of computer used in DAQ applications, the important parameters are the signal sampling frequency, resolution and capacity of data at the same. Additionally, there are other factors effecting to decision of type of computer.

Data acquisition unit's hardware capacity is important, and efficient working of multi-channel data acquisition managing program is very important. It is required that without consuming unnecessary processor cycle, they should be well written to give high performance. In intensive signal processing and floating-point application, graphics card processor can be used the mathematical calculation in addition to central processor.

CUDA can be used with NVIDIA graphics cards, so the high performance required calculations and algorithms could be run on the GPU.

DAQ computer applications not only used in recording the data numerically from external world. Analysis of the received signal in various ways, depending on various parameters in graphical or table representation of common applications. Without such functions, DAQ is just complex data recorder.

Even with the integration of DAQ systems and expert systems, a helpful intelligent system can be obtained. Interpretation of data from the body with disease is increasing trend in medical help. The usage of DAQ is very common apart from the biomedical fields. DAQ devices commonly used in industry for process control. Accordance with the received signals are interpreted by the computer, and so on are decided. Than desired changes in the external world brought under control by actuator. (Austerlitz, 2002)

2.2. Multi-Channel Counter

Another study is the FPGA/DSP based multi-channel counter. This system was implemented Commercial-Off-The-Shelf (COTS) FPGA/DSP based board and it enables up to 64 input channels and counting rate of 45 MHz pulses. This system was intended to implement for photon counting applications based on single-photon avalanche diode (SPAD) arrays. In the proposed system, the collected counting results are transmitted over a high-speed IEEE 1394 serial link (Baronti, 2009).

The majority feature of besides the basic function of counting the incoming pulses, it is able to handle collected information in real-time. High performance and configurable feature may be useful in astronomical researches, which observe fast and slow phenomena, with either high or low luminescence (Baronti, 2009).

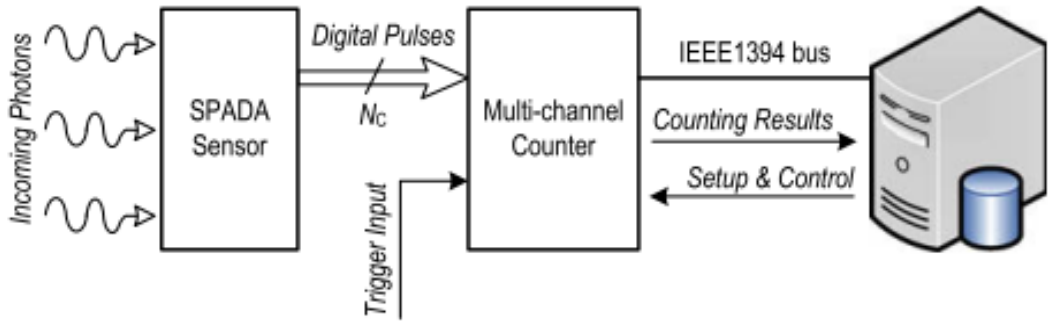


Figure 2. 2 A typical application scenario for astronomical observation (Baronti, 2009)

It is illustrated a typical astronomical experiments for the proposed architecture in figure 2.2. The multi-channel counter can handle incoming pulses in a continuous mode without dead-times between two consecutive the digital signals (Baronti, 2009).

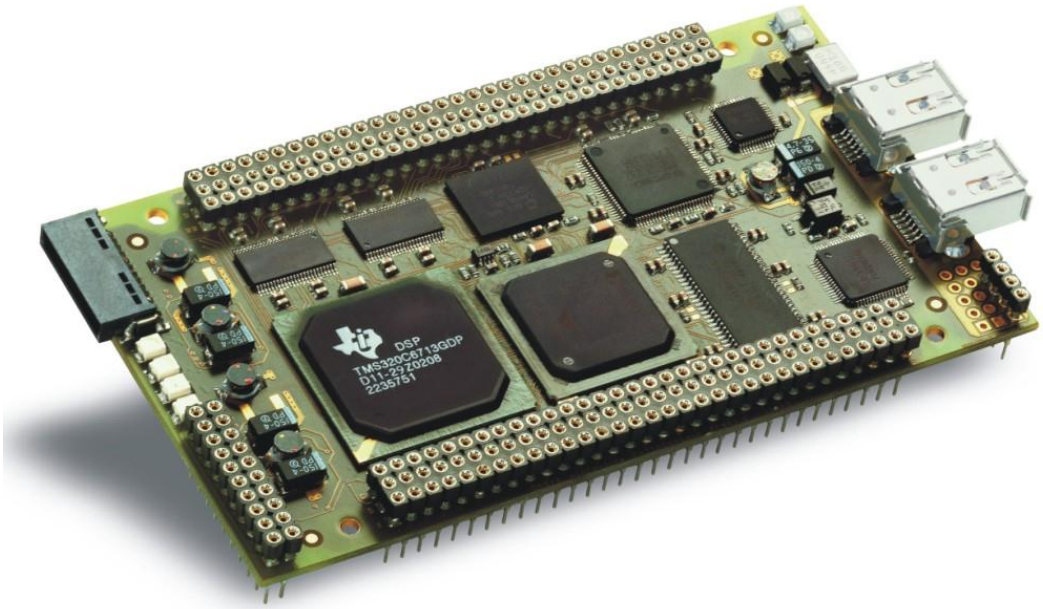


Figure 2. 2 Orsys MicroLine C6713 Compact Development board (Orsys, 2011)

In figure 2.3., Orsys MicroLine C6713 development board that used to implement high performance multi-channel counter is illustrated. In this board are included 1 M gate Xilinx Virtex 2 FPGA, high performance 32 bit floating point Texas Instruments TMS320C6713 DSP, a general purpose TI TSB12LV32 Link-Layer Controller (LLC), a high-speed IEEE 1394 serial interface(Baronti, 2009).

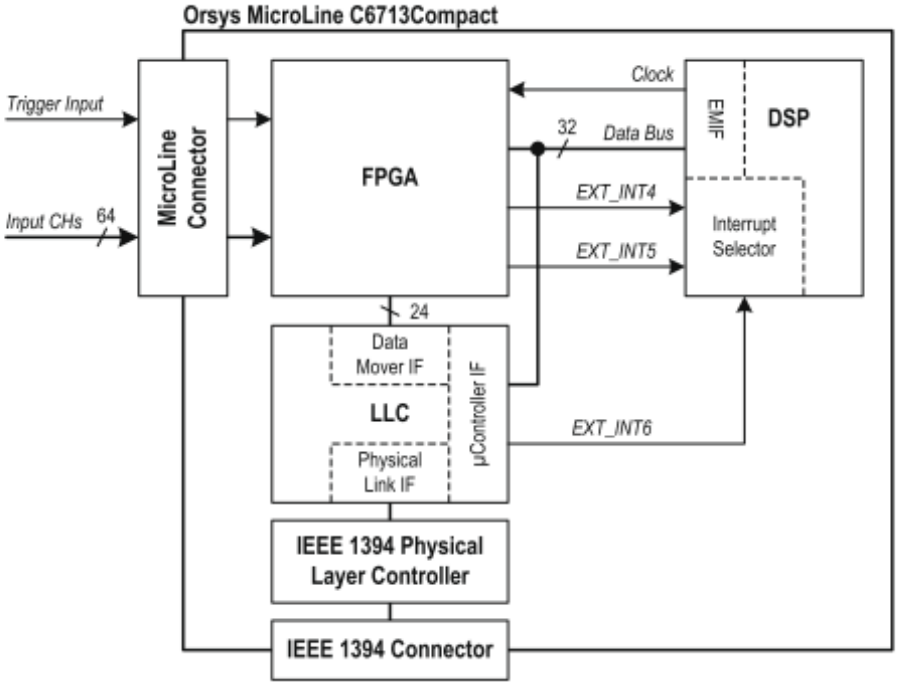


Figure 2. 3 The Architecture of the Orsys MicroLine C6713 Compact (Baronti, 2009)

The External Memory IF (EMIF) of the 32-bit DSP is shared by means of the data bus for both the FPGA and the LLC. LLC block module which includes Physical Link IF, Data Mover IF and Microcontroller (µC) IF, manages both the isochronous and the asynchronous transactions of the IEEE 1394 bus, as shown in figure 2.4. There are MicroLine Connector which routes the 64 input channels directly to the FPGA. This provides implements the acquisition and counting logic (Baronti, 2009).

The FPGA sends packet of counting results to the DSP and the LLC Data Mover IF, when information are ready. Therefore, the DSP can enables to processing of the data in real-time, while the LLC passes them over the IEEE 1394 isochronous link store on remote device. Whenever the DSP has processed a proper number of packets depending on the configured application, it transmits the results to the LLC lController IF. Finally, the processed data can be carried over the IEEE 1394 asynchronous link (Baronti, 2009).

In the FPGA block, there are two main modules. One of them is the Data Mover Port IP Core (DMPORT) that is an IP module. Another majority part is the Data Processing Module (DPM). The DMPORT is connected to the LLC Data Mover

Interface. LLC Data Mover manages the IEEE 1394 isochronous data transfers. The DPM control module manages the handling of the counters (Baronti, 2009).

The DSP enables the real-time data processing and the communication with Linux based computer. Remote user via the IEEE 1394 asynchronous link manages it. The acquisition setting and control commands are exchanged by means of IEEE 1394 link. Furthermore, the same connection is used to periodically get back to the computer the results obtained by the DSP (Baronti, 2009).

The processor Enhanced DMA (EDMA) manages the data transfers from FPGA to DSP. This system uses a Ping-Pong buffer approach. In the DSP internal memory, two buffers are allocated to make smooth data storing and transferring without facing stream dropping. Each of them stores several incoming packets. While one of the buffers is taking the incoming packets by the EDMA, the other one is handled by the DSP, or vice versa. This system also takes packets using Interrupt Service Routine (ISR). Therefore, any packet is not dropped during on-the-fly handling of incoming packets by the DSP (Baronti, 2009).

CHAPTER 3

BASIC TERMINOLOGY

3.1.UNIVERSAL SERIAL BUS

In this chapter, we are going to give basic information about the purpose, existing technology, specifications and design of USB.

3.2. INTRODUCTION TO USB

In this section, we have discussed the USB protocol basics, architecture, working principle, the transfer types, defined in the USB classes and HID. USB is a sophisticated communication protocol, is not possible to examine all the aspects of it in one chapter. Therefore, we explained general aspects of it.

USB is used to connect computer mouse, keyboard, camera, etc. nowadays is the most commonly used protocol for peripherals such as a shared data bus standard and communications protocol. Nowadays it has become the most widely used data transfer tool for electronic products.

3.3. USB HISTORY

The USB has been developed by the leading companies such as Compaq, Microsoft, Intel, Lucent, HP and NXP. These companies do not concern about the income, have come together under the one roof of USB Implementers Forum. Later other companies joined the USB-IF. They undertake duties like developing specifications of USB, fix errors and develop USB-equipped products licensed for commercial distribution. In 90s it was difficult to add new peripherals used in computers. It was difficult to attach a portable communication devices or large external

Storage devices to the computer that were having limited software and hardware features. These problems were trigger for development of flexible and shared data path that allows you easily connect multiple devices.

The first USB version published in November 1994 as 0.7 version number. With intermediate versions several errors and deficiencies has been resolved and two years later, USB 1.0 version has came out. Then in 2000 the USB 2.0 version has been published that gave high speed support and in 2009, USB 3.0 version has been published, where the speed reached to 5 Gbit/s which made USB as fastest ways to common data transfer.

3.4. PURPOSE OF DEVELOPING USB STANDARD

Following criteria has been taking into consideration in designing the USB standard.

- Ease of peripheral device attachments
- Easy usage
- Cost reduction and speed-up data transfer
- Different equipments integration
- Flexible protocols structure that integrates several different data transfer methods
- To make each new version support latter ones
- To support storage of real time data such as videos or audio data.

3.5. USB SPECIFICATIONS

USB incorporates flexibility, high speed, reliability, power saving and tool that is used every day in portable devices and frequently used communication interface for the computer. USB standard supports leading operating systems and computer products. Because of speed and the practical day-to-day usage, it has made USB even more popular communication interface. It appears that presently USB is a communication interface for several appliances like camera, keyboard, mouse, MP3 player, modem, scanner, printer, flash memory, such as portable or fixed in some biomedical devices commonly used in many electronic products. (Ibrahim, 2008)

A number of devices connected to the bus can be increased by using a USB hub. In fact, by using a USB hub we can connect up to five devices consecutively. Because the USB Hub is part of the shared bus interface is being addressed like other devices. For addressing 7-bit is reserved so by using USB communication interface it allows to 127 share the same data path. For addressed devices, hubs are also included. When number of devices and usage of data communication interface increases so frequency of bus traffic increases.

Below headings, examine the benefits of USB.

3.5.1. EASY OF USE

By using the USB plug and play feature operating system quickly make available a data transfer interface in a short period. It is not required for operating system to restart or configuration of IRQ settings.

3.5.2. RELIABILITY

A secure communication data transfer interface is prone to data errors data corruption during transportation. Error checking software is not required. By the help of CRC, inspection mechanism errors that may occur in hardware part can be discovered easily and resolved by transferring the same data again. Errors discovered in data transfer can be rectified in the lower level without intervention upper layers.

3.5.3. LOW COST

Nowadays widely usage of computers brought a low cost. Despite of a complexity of communication protocol needed interfaces can be found cheaply and easily. Even applications that require high-speed can be performed at a lower cost.

These instruments are HID device. Therefore, there is data transfer limit at per second. A computer may have independent Universal Serial Bus as having of USB Controller chips.

CHAPTER 4

PROPOSED SIGNAL ACQUISITION SYSTEM

In this chapter, the proposed structure, prototype description, design of the architecture and its firmware and software will be discussed. Project will be examined in three sections, which are the software of the embedded cards, interface programs on the host computer side and embedded system architecture.

There are many alternatives to design multichannel data acquisition systems. Some of them will be presented on the next pages. We considered developing parameters likes off-the-shelf components, lower hardware and software development costs, open source software, dynamic number of channels with regarded in samples rate and signal resolutions and company software supporting. To get biomedical signals, very high speed sampling does not require. For these reasons, USB HID based interface and Microchip's processor has selected for this study. In this study, we have followed an entirely different approach, in order to facilitate developing the multichannel signal acquisition system.

The biggest advantage of developing USB HID backed project is communicating to the device without the need for developing a new driver. In computer side, the software is developed in the media of Object Pascal and Delphi. A free open source component JEDI HID is used to establish an abstraction from lower level programming. JEDI component is a class, which is developed to communicate with USB HID, based peripheral devices. It has a very important contribution to facilitating and accelerating the software development process.

HID class is defined for human interactive applications. At any time, if there exists a warning or interference from outside, data should be able to instantly transfer to

host. There is not a certain rule such that HID applications should always be human-interactive. Many applications such as sensor reading applications the mechanism control and data collection can be done data collection.

4.1. EMBEDDED SYSTEM ARCHITECTURE

To minimize the hardware developing process, a low cost, commercially available board the 4 OLIMEX Ethernet PIC-USB-STK and 2 Microchip PIC32 Starter Kit boards was used as a USB interface for the host computer to develop computer host based user interface application and the realization of experiments. With the using of such boards at the same time, 6-channel synchronous data exchanging has performed.

In PIC32, evaluation cards because it was not possible to find appropriate connection point for signal reception, assuming that at the buffer there exists ADC data, communication with the host is established. For PIC-USB-STK card, which uses PIC18F4550 processor, data acquisition was performed over ADC channel and the proposed link architecture is carried out and tested.

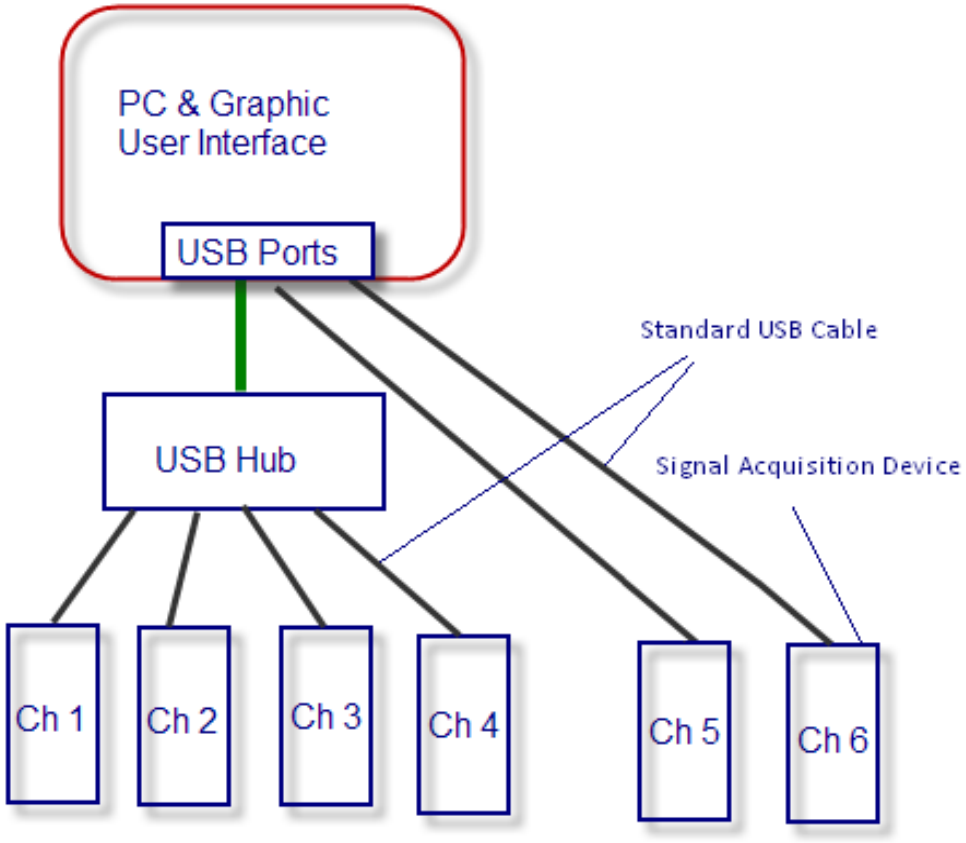


Figure 4. 1 Proposed Architecture System

4.2. DEVELOPMENT KITS

In this project, Microchip's PIC32 ETHERNET STARTER BOARD II and Olimex's PIC-USB-STK and PIC-P40-USB are used to implementing system, developing program for computer which manages the attached acquisition devices, and test process.

The processors that are used on these boards have internal and restricted ADCs because of the system on chip (SoC) of them. Therefore, input voltage ranges must be between zero and five V. The ADC is unipolar. The processor does not accept negative voltages. To have choices more than this ADC options, external ADC can help to achieve more resolutions, voltage ranges and having of negative voltage supporting.

4.2.1. OLIMEX PIC-USB-STK BOARD

Olimex development card uses DIP packaged PIC184550 processor. And the supply power is over USB. Processor can operate up to 48 MHz directly or by PLL circuits. At the first time, bootloader software is installed to the card by using Microchip RealICE programming device. With Microchip's free and open source software bootloader software and using some small special modifications of PIC-USB-STK card, it is compiled by C18. After this stage, without the need for programming device using bootloader it is easily and quickly programmable.

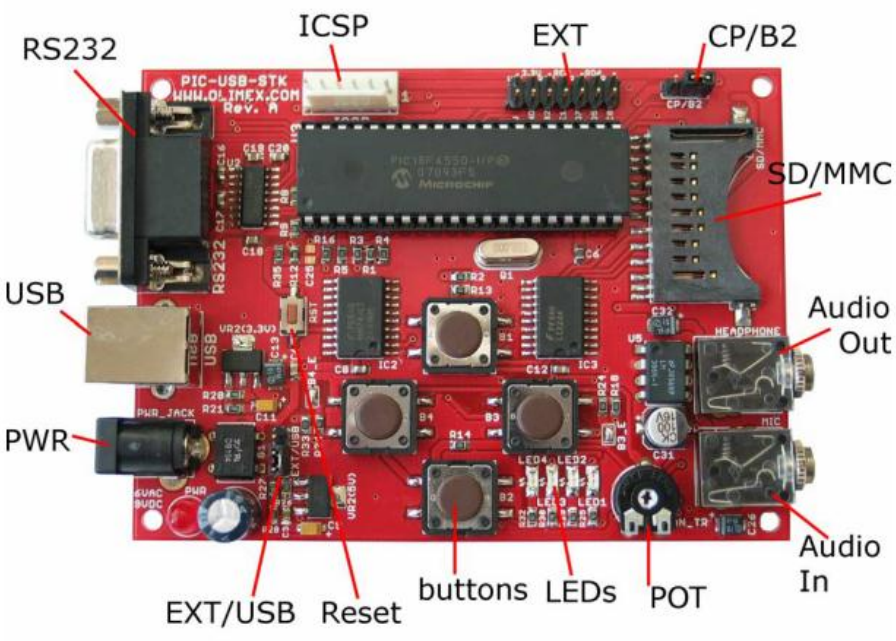


Figure 4. 2 Olimex PIC-USB-STK Development Board (Olimex, 2008)

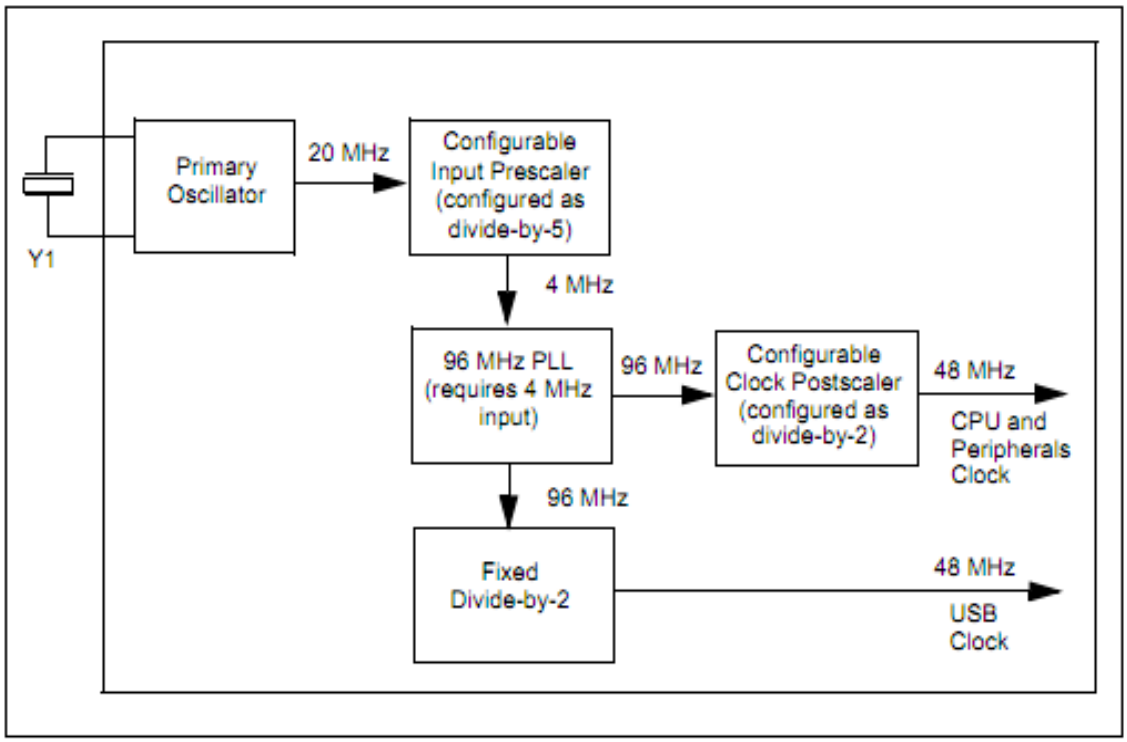


Figure 4. 3 Providing required running frequency for USB on PIC18F4550 processor (Olimex, 2008)

Because of the architectural structure used in PIC processor a normal command can be processed in 4 clock cycle. To apply Full Speed USB with 12 MHz, it is reuired that the processor run at a frequency of 48 MHz. Figure 4.3 shows the simple block scheme how the required 48 MHz frequency for USB application is obtained with 20 MHz clock input in our application.

Y1 (20 MHz) crystal in Development Board is divided to the 5 with frequency divider and output clock frequency with 5-MHz is obtained. By using PLL circuit in next block, 4 MHz oscillation coming from the previous block is multiplied with 24 and 96 MHz is obtained. It becomes multiplied to be used for CPU and peripherals, and the USB hardware. Then, the USB clock is obtained by dividing it by 2. Similarly, the required clock for CPU and peripherals are obtained. Achieving more smooth and stable oscillation is aimed by this way.

The simple circuit scheme is enough for USB based hardware applications. As shown in Figure 7.3, the processor, pull up filter capacitor, usb connector and its other connections and power, ground connections are enough. The required 3.3 V voltage for USB data line of PIC18F4550 is provided by the processor’s internal regulator. Full

Speed or Low Speed run of the device is determined by wheather 3.3 V internal pull up resistors are connected to the USB D + or D-lines.

The connection point of Pull-up resistor is determined by UPUEN byte of UCFG registry. If the 1.5 kΩ - internal resistor in processor is connected to D+, it works with Full Speed. Or if it is connected to the D- then Low Speed communication occurs (Microchip, 2006)

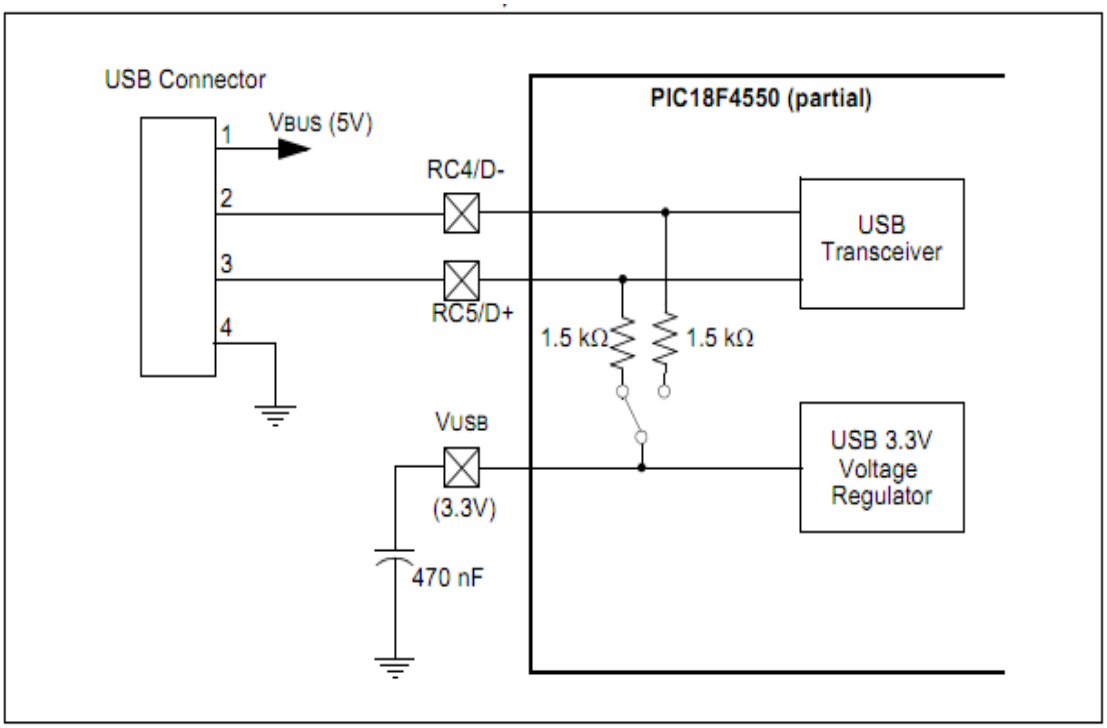


Figure 4. 4 USB Port Schematic on Olimex PIC-USB-STK Board (Olimex, 2008)

The microcontroller has an internal USB Transceiver. This block manages low level communication. In addition, it encode frame as NRZI and decode it.

The Microcontroller has internal Transceiver and additionally USB Transceiver can be connected externally. The obtaining internal Transceiver is an important advantage that makes it low cost and no need to extra components in PCB. This USB block manages the low-level part of communication. Coding and decoding of NRZI data is done this blog's circuits.

4.2.2. MICROCHIP BOOTLADER FOR PIC18

The processor can program itself by using USB HID Bootloader firmware. At the first time, we should put Bootloader program into the processor to have ability of the programming itself. Therefore, after this stage, programmer device is not required. Bootloader and application firmware uses different parts of FLASH and RAM memory.

The usage of USB HID Bootloader for PIC-USB-STK development cards, prevent us to having a programming device, or continuous possession requirement. However, for the first time, there is need for a programmer to install the bootloader. To program cards first time, PicKit2 and Real Ice devices were used.

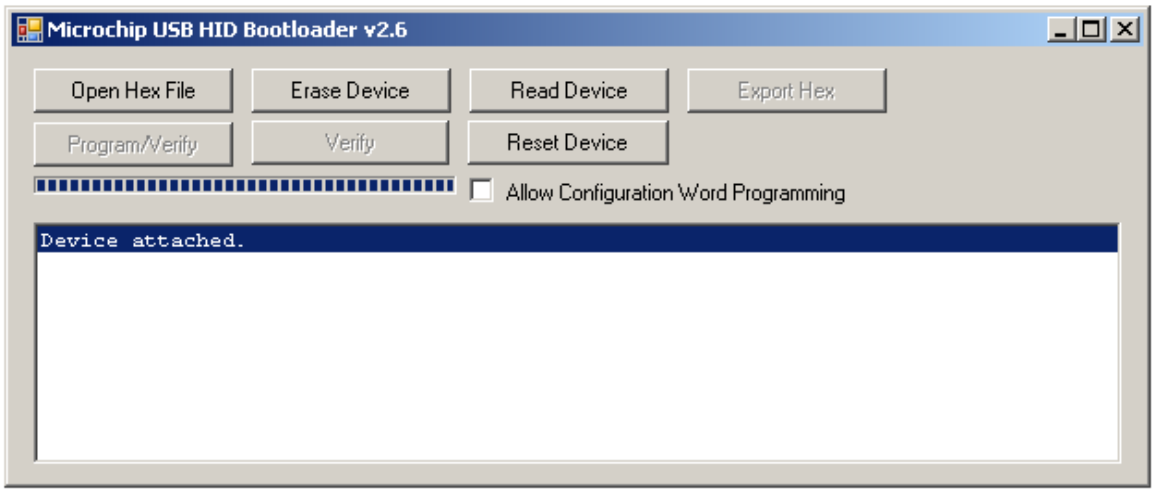


Figure 4. 5 Microchip USB HID Bootloader

4.2.3. DM320004 PIC32 ETHERNET STARTER BOARD

The development card shown in Figure 4.3 uses the Microchip's 32-bit MIPS based PIC32 processor. Features such as USB OTG, USB Device, 10/100 ethernet and integrated programmer/debugger makes it a useful product. PIC32MX795F512L processor, which works at 80 MHz frequency and 80 MIPS (Million Instruction Per Second), gets its supply power via USB line. In the product, there exists an internal programmer/debugger, so there is no need to use bootloader to program the processor. Hardware block diagram is shown in Figure 4.7.

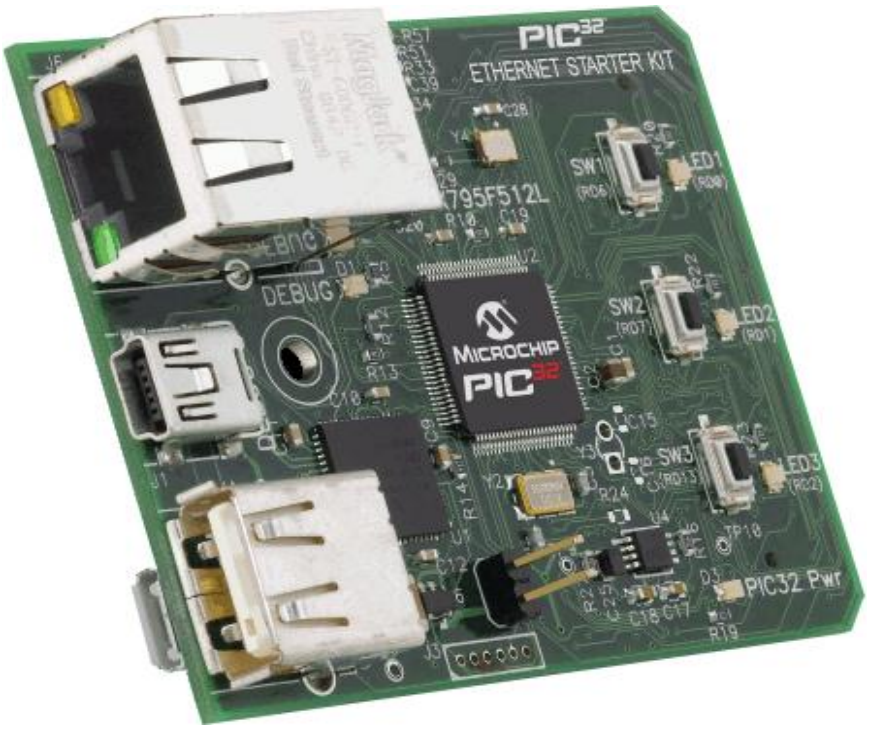


Figure 4. 6 DM320004 PIC32 Ethernet Starter Board II (Microchip, 2010)

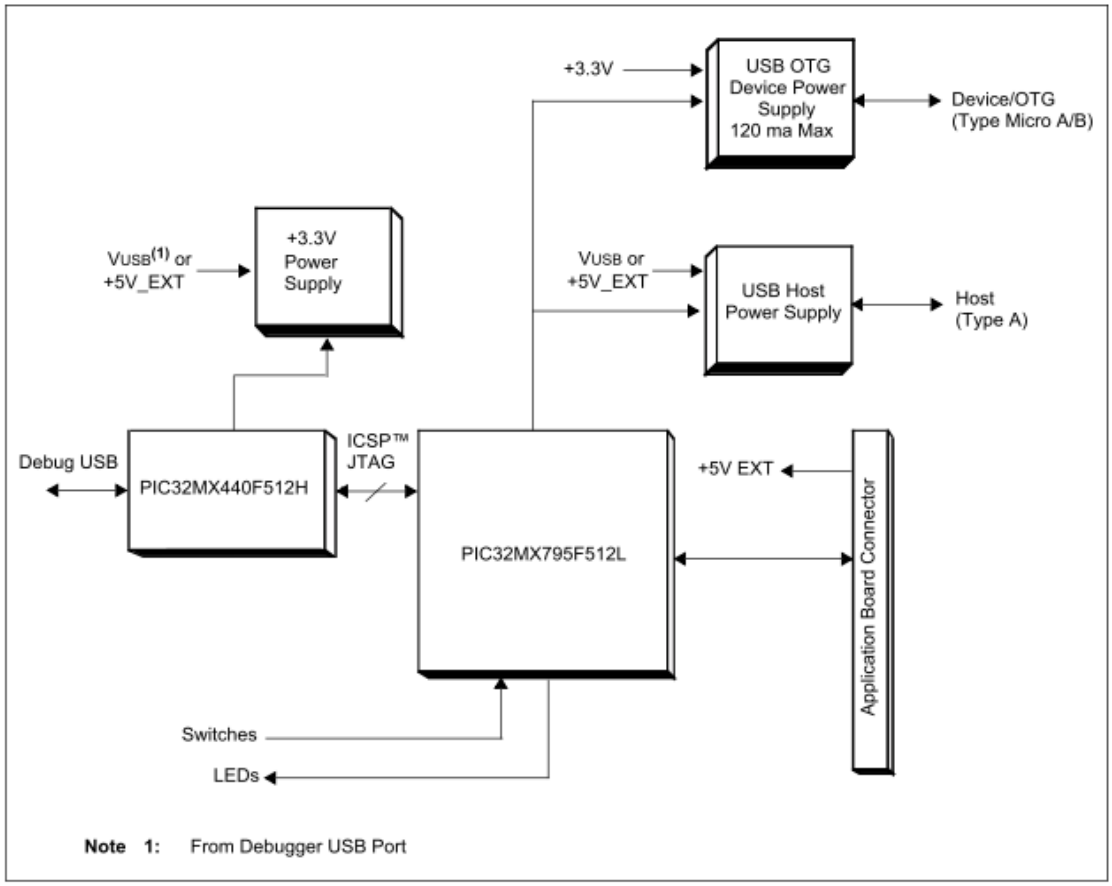


Figure 4. 7 PIC32 Ethernet Starter Board II Block Structure (Microchip, 2010)

4.3. MCU MEMORY ORGANIZATIONS

In Figure 4.8, knowledges about bootloader, usb application, software stack field configuration and reserved special areas are given. Configurations in the figure is done for C18 compiler at MPLAB by a written configuration to linker script. After compiling the firmware, data, opcode and variable installation to the protected areas of ram and rom sections in the process of linking data is prevented. Rows specified by codepage, organisation rows of flash memory and lines specified by databank can be used to define reserved areas of the RAM section.

```
LIBPATH .
FILES c018i.o
FILES clib.lib
FILES p18f4550.lib

CODEPAGE   NAME=vectors    START=0x0              END=0x1F           PROTECTED
CODEPAGE   NAME=BootPage   START=0x20             END=0xFFF
CODEPAGE   NAME=page       START=0x1000           END=0x7FFF         PROTECTED
CODEPAGE   NAME=idlocs     START=0x200000         END=0x200007       PROTECTED
CODEPAGE   NAME=config     START=0x300000         END=0x30000D       PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE         END=0x3FFFFF       PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000         END=0xF000FF       PROTECTED

ACCESSBANK NAME=accessram  START=0x0              END=0x5F
DATABANK   NAME=gpr0       START=0x60             END=0xFF
DATABANK   NAME=gpr1       START=0x100            END=0x1FF
DATABANK   NAME=gpr2       START=0x200            END=0x2FF
DATABANK   NAME=gpr3       START=0x300            END=0x3FF
DATABANK   NAME=usb4       START=0x400            END=0x4FF          PROTECTED
DATABANK   NAME=usb5       START=0x500            END=0x5FF          PROTECTED
DATABANK   NAME=usb6       START=0x600            END=0x6FF          PROTECTED
DATABANK   NAME=usb7       START=0x700            END=0x7FF          PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF60            END=0xFFF          PROTECTED

SECTION    NAME=CONFIG     ROM=config
SECTION    NAME=USB_VARS   RAM=usb4
STACK      SIZE=0x60       RAM=gpr3
```

Figure 4. 8 Memory Organizations For HID Bootloader

```
CODEPAGE   NAME=bootloader START=0x0              END=0xFFF          PROTECTED
CODEPAGE   NAME=vectors    START=0x1000           END=0x101F         PROTECTED
CODEPAGE   NAME=page       START=0x1020           END=0x7FFF
CODEPAGE   NAME=idlocs     START=0x200000         END=0x200007       PROTECTED
CODEPAGE   NAME=config     START=0x300000         END=0x30000D       PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE         END=0x3FFFFF       PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000         END=0xF000FF       PROTECTED

ACCESSBANK NAME=accessram  START=0x0            END=0x5F
DATABANK   NAME=gpr0       START=0x60           END=0xFF
DATABANK   NAME=gpr1       START=0x100          END=0x1FF
DATABANK   NAME=gpr2       START=0x200          END=0x2FF
DATABANK   NAME=gpr3       START=0x300          END=0x3FF          PROTECTED
DATABANK   NAME=usb4       START=0x400          END=0x4FF          PROTECTED
DATABANK   NAME=usb5       START=0x500          END=0x5FF          PROTECTED
DATABANK   NAME=usb6       START=0x600          END=0x6FF          PROTECTED
DATABANK   NAME=usb7       START=0x700          END=0x7FF          PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF60          END=0xFFF          PROTECTED

SECTION    NAME=CONFIG     ROM=config

STACK SIZE=0x100 RAM=gpr3

SECTION    NAME=USB_VARS   RAM=usb4
```

Figure 4. 9 Memory Organization For USB HID Application Firmware

4.4. PROPOSED STRUCTURE OF ARCHITECTURE

The proposed structure of architecture in this project supports expansion of the number of channels, which is related with connected device to the universal port by using USB's star connection feature.

As an original approach, this architectural structure provides an example of a modular and flexible architecture for biomedical signal acquisition instruments and biomedical research projects by the property of dynamic channel number and ease of use. In the case of insufficiency of number of USB ports, USB ports can be increased by using it as a hub and at the same time, many HID-based signal acquisition devices can be connected.

In the proposed system, data taken from each HID-based signal acquisition devices is buffered in the memory and they are sent to the computer in certain periods by the method of interrupt transfer. On the side of computer, the program, which manages the device, performs the data stream by communicating with a large number of USB-based data collecting devices individually. In addition, this program provides the

user interface, graphically draws data from devices on a time axis and stores data to the computer as a data file.

In our system, timer interrupt is used to provide a systematic and regular period to read analog data. The processor PIC18F4550 used in this project has 10 bit ADC support. By performing signal conditioning between 0~5 V, continuous signals can be read with 1/1024 precision. 10 bit resolution data read from ADC are recorded to microcontroller's buffer as a sequence of 2 bytes.

Digitized biosignals which are accumulated at buffer by fixed temporal cut-off routine, transferred immediately to the buffer area, which is reserved for performing data transfer when buffer memory becomes full. And the transfer process to the host starts by interrupt transfer.

Buffering biosignals numerically can be continued by the timer interrupts that performs at the same time when data is transferred from transfer buffer to the computer and it avoids the loss of data during transfer. Signal registration performs during signal reception with the same periods.

The time difference between timer interrupts is arranged as 32 digitized analog signals can be read in the time interval of USB interrupt transfers. By using buffer and transfer buffer, data corruption and temporal shift during collecting data in a single channel can be restrained.

However, the number of devices connected by data path or the excessive increase in data flowing traffic may have major negative effects. Despite that, in present computers have more than one USB controller and, it increase the ability to manage a large number of device connectivity and high data traffic. If the number of devices connected to the data path is not too much, then due to the low frequency in the Biosignals collected in the devices delays can be negligible. For applications which delay is very important and very low tolerance is needed, all signals can be stored at an external ram numerically and after completing the registration process, transfer process can be performed without concern of traffic on the data path.

With the interrupt transfer packages that can transfer up to a maximum of 64 byte data, 32 units of ADC data can be transferred to the PC. It is possible to send more

ADC data in a single transfer by compression algorithms. For efficient usage of the 2 byte memory areas where 10 bit ADC data kept, ADC data can be saved in the remaining 6 bit areas.

The total number of bits for each transfer: 8 bit x 64 byte = 512 bit

The number of transferable 10 bit ADC data: 512 bit / 10 bit = 51 ADC data

The number of transferable 16 bit ADC data: 512 bit / 16 bit = 32 ADC data

As it seen above, in a dingle transfer it is possible to transfer 51 times 10-bit ADC data.

4.5. BIOMEDICAL DAQ USER INTERFACE

The device descriptor provides its own identifying information to the host computer, which the device has attached. When the device attach to the host computer, it sends the device configurations and descriptors by using control transfers. If the currently drivers has installed, the host uses same drivers with the previous settings. If it firstly attached, the Operation System (OS) on the host tries to find appropriate driver. If OS does not find out the corresponding driver, it asks its location to the user or installing it. If the required driver is an HID, Operating system can be able to install without asking to the user, and get ready for use.

As expressed earlier, 127 USB peripherals with including hubs could be connected to the bus. The host computer distinguishes devices, which attached to itself, by means of VID, PID information. If the multiple devices, that have same VID and PID numbers, attached to the host, in this case, the host computer differentiates from the each other by the serial numbers. In our system, we has considered that fact all HID DAQ devices will have same VID and PID numbers. So, different serial numbers are assigned to each device.

The user interface program has written in Delphi development environment for the host computer that it has capable to communicate with multiple biomedical data acquisition peripherals at the same time. However, it can also configure and managed

each device. In additional, the interface program, which was developed by Object Oriented Pascal (OOP), that allows easy controlling device and monitoring of acquired biomedical signals.

When additional HID based DAQ attached to the computer, the program creates a new user interface on the tab with specific options for the device. So, each device has own user interface and signal plotting screen and other options on the tab in the program. As shown Figure 4.11, there are some control buttons on the program interface, which they belong to certain device. These buttons provide some useful control functions for each device such as starting of acquisition signals, stopping acquisition process, signal recording, switching or changing active device interface. In addition, there are some options for plotting signal on the screen.

As shown Figure 4.10, the computer user interface screen, that three biomedical signal acquisition device was connected to the host, was illustrated, that managing three devices, storing signal for each devices as numerically and drawing chart at the same time. These tasks were performed as multithreading. Each thread can capable to store the corresponding signal in memory or computer hard disk.

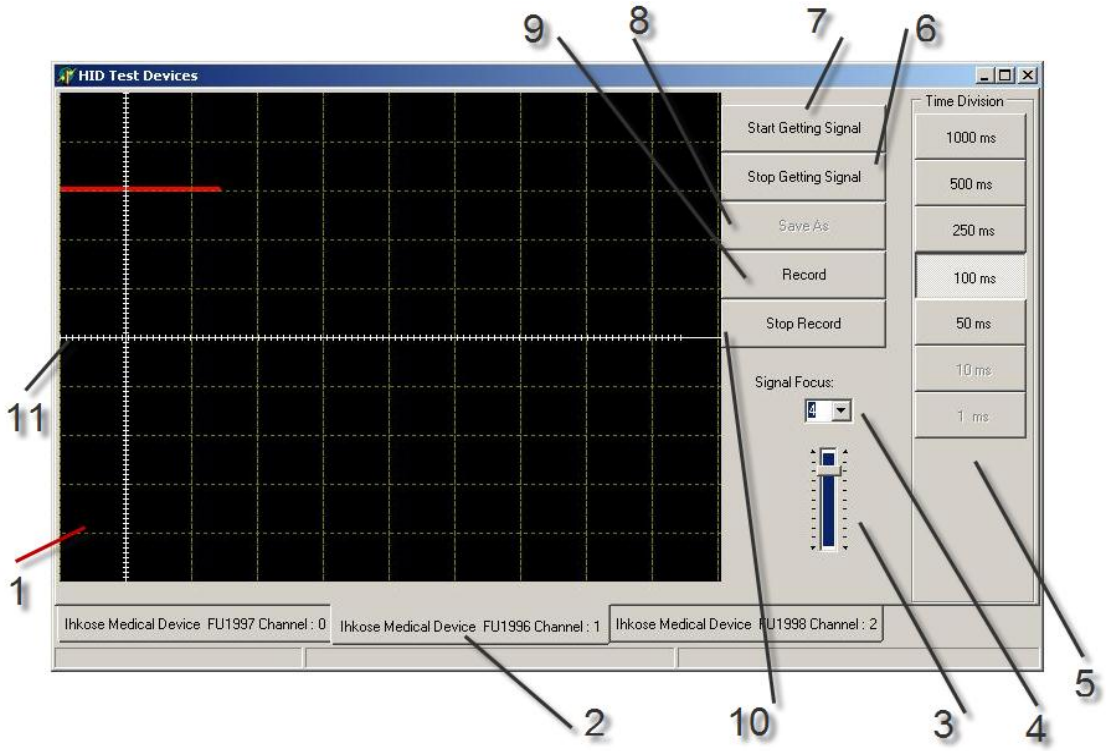


Figure 4. 10 Data Acquisition User Interface Program

This program creates the new object that is separate controls and having of signal chart screen for each device on the tabs, as the number of connected the device to the PC. Number of tab on the form is same with attached DAQ devices. The minimum transfer interval of the ADC data vector has arranged as 32 ms by means of timer interrupt. Therefore, these arrangements facilitate running of our program without performance decreasing on the host computer.

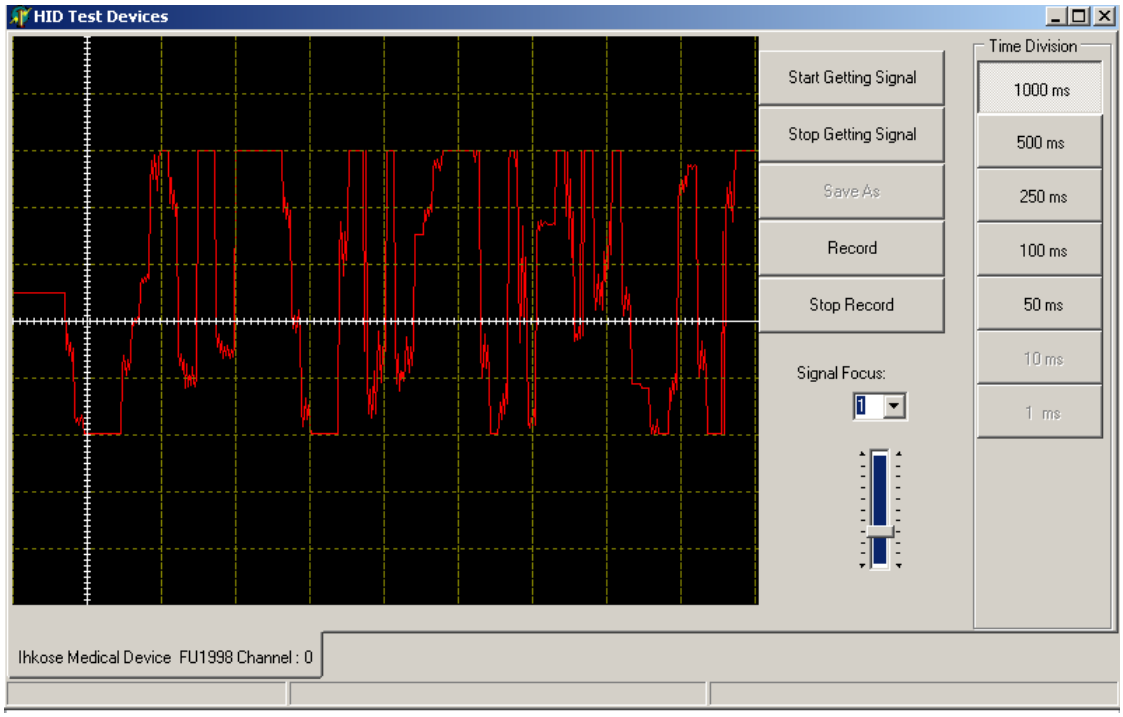


Figure 4. 11 Data Acquisition User Interface Program

4.6. FIRMWARE

Data acquisition application on USB stack framework is developed using the Microchip C18 C compiler. Device descriptor data which describes the device to USB host can be seen in Figure 4.12. In this array, which is saved on ROM, VID, PID and Serial Number data can be found as well as version numbers and device release number. In this data structure, the size of buffer of control transfer is also determined.

```
//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};

//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002,
    (ROM BYTE *ROM)&sd003
};
```

Figure 4. 12 String Index

```
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,    // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,  // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,      // Max packet size for EP0, see usb_config.h
    0x04D8,                 // Vendor ID
    0x0060,                 // Product ID:
    0x0021,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x03,                   // Product string index
    0x02,                   // Device serial number string index
    0x01                    // Number of possible configurations
};
```

Figure 4. 13 Device Description

When the device is attached to the computer, “Ihkose Medical Device” marking which appears in right bottom corner of the monitor, is shown with the index number from Product String line after enumeration process. Manufacturer, Product and Serial Number String data are array index numbers located in ROM as described in Figures 4.12, Figure 4.13 and Figure 4.14.

```
//Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
}};

//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{
'F','a','t','i','h',' ','U','n','i','v','e','r','s','i','t','y'
}};

// Serial Number Descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[7];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{

 //'F','U','1','9','9','7'
  //'F','U','1','9','9','6'
  //'F','U','1','9','9','8'
  'F','U','1','9','9','9'
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[22];}sd003={
sizeof(sd003),USB_DESCRIPTOR_STRING,
{
'I','h','k','o','s','e',' ','M','e','d','i','c','a','l',' ','D','e','v','i','c','e'
}};
```

Figure 4. 14 Language, Manufacturer, Product and Serial Number Strings

```
/* Configuration 1 Descriptor */
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,//sizeof(USB_CFG_DSC),    // Size of this descriptor in bytes
    USB_DESCRIPTOR_CONFIGURATION,                // CONFIGURATION descriptor type
    0x29,0x00,            // Total length of data for this cfg
    1,                      // Number of interfaces in this cfg
    1,                      // Index value of this configuration
    0,                      // Configuration string index
    _DEFAULT | _SELF,               // Attributes, see usb_device.h
    50,                     // Max power consumption (2X mA)
    /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),   // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE,               // INTERFACE descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    2,                      // Number of endpoints in this intf
    HID_INTF,               // Class code
    0,     // Subclass code
    0,     // Protocol code
    0,                      // Interface string index

    /* HID Class-Specific Descriptor */
    0x09,//sizeof(USB_HID_DSC)+3,    // Size of this descriptor in bytes
    DSC_HID,                // HID descriptor type
    0x11,0x01,                 // HID Spec Release Number in BCD format (1.11)
    0x00,                   // Country Code (0x00 for Not supported)
    HID_NUM_OF_DSC,         // Number of class descriptors, see usbcfg.h
    DSC_RPT,                // Report descriptor type
    HID_RPT01_SIZE,0x00,//sizeof(hid_rpt01),      // Size of the report descriptor

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    HID_EP | _EP_IN,                //EndpointAddress
    _INTERRUPT,                    //Attributes
    0x40,0x00,                  //size
    0x20,                        //Interval  , data is sended per 32 ms

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    HID_EP | _EP_OUT,                //EndpointAddress
    _INTERRUPT,                    //Attributes
    0x40,0x00,                  //size
    0x20                        //Interval, data is sended per 32 ms
};
```

Figure 4. 15 USB Configuration Descriptor

4.7. ADC SETTINGS

```
//Channel 8, RB2, AN8
TRISBbits.TRISB2 = 1; // set as input
OpenADC(ADC_FOSC_64 & ADC_RIGHT_JUST &  ADC_20_TAD,
        ADC_CH8 & ADC_INT_OFF & ADC_8ANA,
        10);

SetChanADC(ADC_CH8);
```

Figure 4. 16 ADC Settings

```
void readAdcVal(void)
{
    Delay10TCYx(5); //Delay for 50TCY. Because within this delay, the holding capacitor
    ConvertADC (); // Start an A/D conversion.
    while( BusyADC()); // Wait for completion. when BusyADC is cleared, the conversion is
    wReadPot.Val = ReadADC(); // Read result
}
```

Figure 4. 17 Reading ADC Values

4.8. TIMER AND INTERRUPTS

The DAQ device sends ADC data, which is stored in the microcontroller's internal buffer for each 32 millisecond intervals, to the host. Each transfer period time is fixed by means USB interrupt transfer. This system likes ping pong buffering approach. There are two separate buffers. We can call them as transfer buffer and storing buffer. While transferring digitized signal to the PC, storing buffer is available to keep on the memory all receiving data with fixed time by means of the timer interrupt. When storing buffer is full, in this case, content of the storing buffer is copied to the transferring buffer immediately after the last timer interrupt. This approach provides safely transmission without losing or dropping acquired data.

```
INTCON2bits.TMR0IP = 1; // High priority
RCONbits.IPEN = 1;
T0CON = 0b101;// 1:64 prescaler
T0CONbits.T08BIT = 1; // 8 bit
T0CONbits.T0CS = 0; //
TMR0L = 70;
TMR0H = 0;
T0CONbits.TMR0ON = 1; // enable timer0

// Set timer0 interrupt as high priority
INTCONbits.TMR0IE = 1; // enable timer0 Isr
INTCONbits.TMR0IF = 0; // clear flag
INTCONbits.GIEH = 1;
INTCONbits.GIEL = 1;
```

Figure 4. 18 Timer and Interrupt Configurations

```
if(INTCONbits.TMR0IF)
{
    TMR0L = 70; // set to each 992 us
    TMR0H = 0;
    INTCONbits.TMR0IF = 0;
    bIndex++;

    // put in buffer

}
```

Figure 4. 19 Timer Interrupt Handler

```
BYTE TransferBuff[64];
void ProcessIO(void)
{
    BYTE bIndexTemp;

    readAdcVal();
    INTCONbits.GIEH = 0;
    bIndexTemp = bIndex;
    ToSendDataBuffer[2 * bIndexTemp] = wReadPot.v[0];
    ToSendDataBuffer[2 * bIndexTemp + 1] = wReadPot.v[1];
    INTCONbits.GIEH = 1;

    if( (SendStream) && (!HIDTxHandleBusy(USBInHandle)) )
    {
        INTCONbits.GIEH = 0;
        memcpy((void *)TransferBuff,(void *)ToSendDataBuffer,64)
        INTCONbits.GIEH = 1;
        USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&TransferBuffr[0],64);
        bIndexTemp = 0;
    }
```

Figure 4. 20 ADC Data Buffering and Transfer Handler to PC

4.9. BIOSIGNAL AQUISITION

The frequency of the biosignals that can be gathered from human body is low and sampling frequency less than 1 KHZ is enough for many applications. In this project, the signal voltage minimum and maximum are determined by the opamp used, especially ADC. The biosignal gathered from the body should go through analog circuits, amplifiers, isolation circuits, and filters according to their type before entering ADC. After these steps, the biosignal should be between 0~3.3V or 0~5V according to the ADC type used.

Our project starts from the output of ADC. With the offered modular system, different biomedical signals can be gathered at the same time. In addition, one of the most important advantages of this system is the ability to gather data at different

sampling frequencies and resolutions for each channel. However, since the data transfer capacity is limited for HID class, the increase in channel number, sampling frequency and resolution inversely affect the total and per channel data number.

4.10. CAPACITY TO GATHER DATA OF THE DEVICE

The width of each frame in Fullspeed USB communication is 1ms. Interrupt transfer method supports Full Speed and the maximum byte number that can be transferred in every transaction is 64. A device with a high frequency is not needed to gather data from human body. Many of the frequencies of these biosignals are below 1 kHz.

4.11. THE CONNECTION OF THE DEVICE WITH THE COMPUTER

The operating system reads the Device Descriptor and Configuration descriptor data in order to define and install the driver of the device connected to the USB databus. By this way, the computer defines the device type as HID. After that, when the device is attached to a PC, a hint window is opened at the right bottom corner including the message “A new Human Interface Device is found”.

CHAPTER 5

FUTURE WORKS

It is aimed in future work to make the system more stable and practical by adding a user-friendly interface, real time data acquisition and electrical isolation between analog acquisition part and computer interface. Besides, a modification which allows multi channel data to be recorded in WAV file format separately and to be listened, in the application of lung sounds acquisition, will be considered.

CHAPTER 6

CONCLUSION AND DISCUSSION

The proposed architecture is tested in a PC having Dual Core 2.6 GHz CPU. to evaluate the performance of the system, 6 synchronous data acquisition is performed. When minimum interrupt interval is set to 1 ms, the operating system could not handle with it, and performance of PC decreased dramatically. When the test is done in a netbook with Intel Atom 1.6 GHz CPU, by setting the interrupt interval to 32 ms, no performance decline observed.

This test is also performed at a workstation having Intel Xeon CPU 3.2 GHz. Thanks to its number of USB controllers and strong processing capacity, no performance decrease is observed even interrupt interval is set to 1 ms. As the result of experiment, the PC that is going to be used to acquire data should have more than one USB controller and CPU core.

As a conclusion, the techniques those are proposed in this work are realized, and 6-channel data acquisition device is made based on USB. This architecture is tested on different PCs, and results are good as expected. This modular and easy to multiply DAQ architecture decreases research cost and effort.

REFERENCES

Aiello, S., Anzalone, A., Bartolucci, M., Cardella, G., Bartolucci, M., Cardella, G., Cavallaro, S., DeFilippo, E., Femino, S., Geraci, M., Giustolisi, F., Guazzoni, P., Iacono Manno, M., Lanzalone, G., Lanzano, G., Lo Nigro, S., Manfredi, G., Pagano, A., Papa, M., Pirrone, S., Politi, G., Porto, F., Rizzo, F., Sambataro, S., Sperduto, L., Sutera, C., Zetta, L., Data acquisition and real-time computing by a DSP-based system, Elseiver Science B.V., 1998

ANDERSON D., Universal Serial Bus Architecture, PC System Architecture Series

Austerlitz, H., Data Acquisition Techniques Using PCs, Academic Press, New York, 2002

Axelson, J., USB Completed The Developer's Guide Fourth Edition, Madison, 2009

Baronti, F., Lazzeri, A., Roncella, R., Saletti, R., FPGA/DSP-based implementation of a high-performance multi-channel counter, Journal of Systems Architecture, March, 2009, Italy

Enumeration of Interface Collections on USB Composite Devices, http://msdn.microsoft.com/en-us/library/ff538828(v=VS.85).aspx , 2010

Ibrahim, D., Advanced PIC18 Projects, Newnes, Dogan Ibrahim, Oxford, 2008

Microchip, PICDEMTM FS USB DEMONSTRATION BOARD USER'S GUIDE, http://ww1.microchip.com/downloads/en/DeviceDoc/51526b.pdf

Microchip, PIC32 Ethernet Starter Kit User's Guide, 2010, http://ww1.microchip.com/downloads/en/DeviceDoc/61166A.pdf

Microchip, PIC32 Ethernet Starter Kit. Last Access: 2010, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en545713

Microchip, PIC32MX USB Starter Kit II Schematics, www.microchip.com

Microchip, PIC18F2455/2550/4455/4550 Data Sheet, 2006, http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf,

Olimex, PIC-USB-STK Board Board Datasheet, 2008, http://www.olimex.com/dev/pdf/PIC/PIC-USB-STK.pdf

Olimex, PIC-P40 Development Board User Manuel, 2008, http://www.olimex.com/dev/pdf/PIC/PIC-P40.pdf

Orsys , http://www.orsys.de/322c6713.htm, last access: 07.2011

Palas-Areny, R. and Webster, J. G., Analog Signal Processing, John Wiley & Sons Inc., New York, 1999

The last HID Usage Tables 1.12 Specification, http://www.usb.org/developers/devclass_docs/Hut1_12.pdf

USB-IF, USB Specification 2.0, 2002, www.usb.org

USB Stack, www.microchip.com, 2010

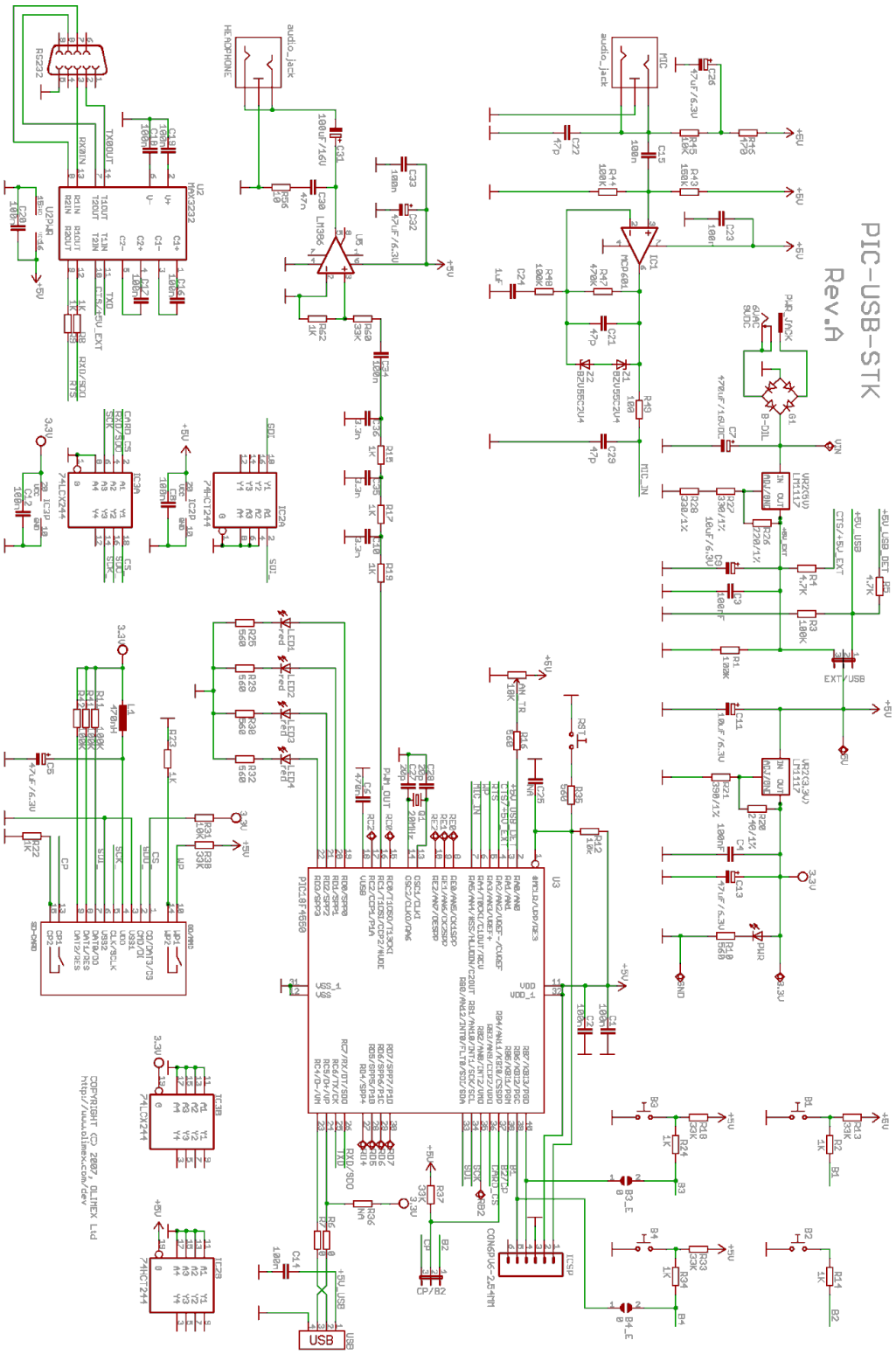
USB in a Nutshell. Making Sense of the USB Standard, www.beyondlogic.org, 2010

USB Class Codes, http://www.usb.org/, 2010

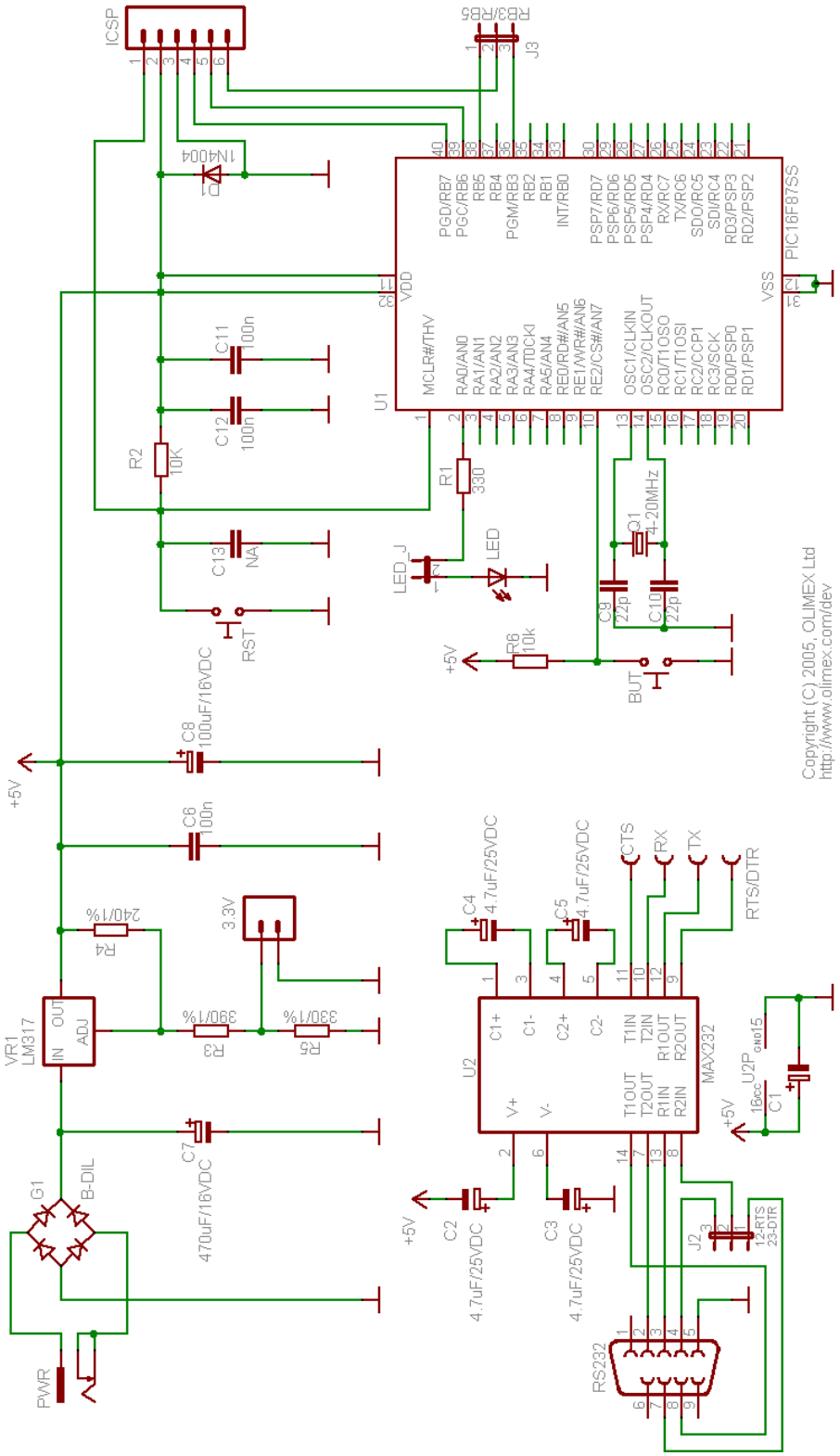
USB HID specification, HID-usages tables, and HID descriptor tool, www.usb.org/developers/hidpage.html.

Wojciech M. Zabolotny, Przemyslaw Laniewski Wollk, Wojciech Zaworski, Low Cost Open Data Acquisition System For Biomedical Applications, Photonics Applications İn Industry And Research IV. (SPIE Conference Proceedings), Warsaw, Poland, 2005

APPENDIX A : OLIMEX PIC-USB-STK SCHEMATIC



APPENDIX B : OLIMEX PIC-P40 DEVELOPMENT BOARD SCHEMATIC



APPENDIX C: EXAMPLE OF USING THIDFORM

```
function TForm1.HidCtlEnumerate(HidDev: TJvHidDevice;
  const Idx: Integer): Boolean;
var
  device: TJvHidDevice;
  devIndex: Integer;
  newForm : THidForm;
begin
 if(    (HidDev.Attributes.VendorID    =    WORD($04D8)    )    and
(HidDev.Attributes.ProductID = $0060) ) then
  begin
  if HidDev.ProductName = '' then
  begin
    result := FALSE;
    exit;
  end;
    //devIndex := DeviceList.Items.Add( Format('%s - %s - Idx =
%d', [HidDev.ProductName, HidDev.SerialNumber, Idx]));
    HidCtl.CheckOutByIndex(device, Idx);
   // DeviceList.Items.Objects[devIndex] := device;
    newForm  :=  THidForm.Create(HidDev.ProductName  +  '    '  +
HidDev.SerialNumber , HidDev, Idx);
  end;
  Result := True;
end;
```

APPENDIX D : THIDFORM CODE

```
{-------------------------------------------------------------------}
{Author: İsmail Hakkı KÖSE                                          }
{Date: 10 - 2010}                                                   }
{-------------------------------------------------------------------}

     unit Unit2;
     interface
     uses
       Windows, SysUtils, Classes, Controls, StdCtrls, Forms,
       JvHidControllerClass,variants,Dialogs,
HID,ComCtrls,ExtCtrls,Graphics;
     type
        TRGBType=record
            RedHex:string;
            GreenHex:string;
            BlueHex:string;
            Red:integer;
            Green:integer;
            Blue:integer;
        end;

     type
       THidReport = packed record
         ReportID: Byte;
         Data: array [0..62] of Byte;
     end;

     type
        TLevelPoint = record
        X: Int64;
        Y: Real;
     end;

     type
        TSignal = record
        time: Cardinal;
        Voltage: integer;
     end;
```

```
type
  THidForm = class
  private
    Name_Dev:ansistring;
    fName: string;
    signalTick: TLevelPoint;
    Memo: TMemo;
    Button1 : TButton;
    Button2 : TButton;
    Button3 : TButton;
    Button4 : TButton;
    Button5 : TButton;
    HidCtl: TJvHidDeviceController;
    MyDevice : TJvHidDevice;
    HidData:THidReport;
    diffData:DWORD;
    HidCnt1 : WORD;
    MyVid, MyPid: WORD;
    MySerialNumber: Widestring;
    NewTabSheet:  TTabSheet;
    TabSheetId : integer;
    SignalCanvas: TImage;
    Panel : TPanel;
    digSignal: TMemoryStream;
    sigTick: TSignal;
    yOffset, xOffset, yMaxVal, xMaxVal:integer;
    xMid, yMid : integer;
    procedure HidCtlDeviceChange(Sender: TObject);
    procedure OnHidData(HidDev: TJvHidDevice; ReportID: Byte;
  const Data: Pointer; Size: Word);
  function JvHidDeviceController1Enumerate(HidDev: TJvHidDevice;
    const Idx: Integer): Boolean;
  procedure JvHidDeviceController1Removal(HidDev: TJvHidDevice);
  procedure JvHidDeviceController1Arrival(HidDev: TJvHidDevice);
  procedure Button2Click(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure Button3Click(Sender: TObject);
  procedure Button4Click(Sender: TObject);
  procedure Button5Click(Sender: TObject);
  procedure initGraphCanvas;
```

```
      procedure drawGridScale;
      procedure generateDummySignal(time: cardinal);
      procedure    clearOldSignal(sigIdxFp:    Cardinal;    sigIdxLp:
Cardinal);
      procedure   drawSignalOnScreen(sigIdxFp:   Cardinal;   sigIdxLp:
Cardinal);

      // This function converts to Y's real value
      function getRealYPoint(yP:integer):integer;
      // This function converts to Y's windows OS value
      function getOsYPoint(yP:integer):integer;
      public
        DrawingGraphic: BOOL;
        property Name: string read Fname;

        constructor create(AName:string; HidDev: TJvHidDevice;
      const Idx: Integer);
        destructor Destroy;override;
      end;

    implementation

    uses unit1;

    const MaxVoltageLevel = 5.0;
    const MAX_ADC_VAL = 65536;//16 bit //1024/2;// 12 bit ADC
    CONST MAX_VOL_LEVEL = 5;
    const NUMBER_OF_LINES_ON_X_AXIAL = 11;
    const NUMBER_OF_LINES_ON_Y_AXIAL = 10;
    const clBACKGROUND_SINGAL_BOARD = clBlack;
    const clGRID_ON_SINGAL_BOARD = clOlive;
    const clAXIS = TColor($FEFEFE);

    // Start Button
    procedure THidForm.Button1Click(Sender: TObject);
    var
        ToWrite, Written: Cardinal;
    begin
      HidData.Data[0]:= $10;//Start Counter
      HidCnt1 := 1;
      HidCnt1 := 0;
```

```
      if(MyDevice <> nil) then
      begin
        HidData.ReportID:=0;
        MyDevice.WriteFile(HidData,
MyDevice.Caps.OutputReportByteLength, Written);
      end;
    end;

    // Stop Button
    procedure THidForm.Button2Click(Sender: TObject);
    var
        ToWrite, Written: Cardinal;
    begin
        HidData.Data[0]:= $11; // Stop counter    end;
        HidCnt1 := 1;
        if(MyDevice <> nil) then
        begin
            if MyDevice.HasReadWriteAccess then
            begin
              HidData.ReportID:=0;
              HidData.Data[0]:= $11; // Stop counter
              MyDevice.WriteFile(HidData,
MyDevice.Caps.OutputReportByteLength, Written);
            end
            else
                begin
                    MyDevice.OnData := nil;
                end;
        end;
    end;

    // Stop Drawing
    procedure THidForm.Button4Click(Sender: TObject);
    begin
       DrawingGraphic := FALSE;
    end;

    procedure THidForm.OnHidData(HidDev: TJvHidDevice; ReportID: Byte;
      const Data: Pointer; Size: Word);
    var i, Bit:integer;
```

```
        pData:array of byte;
        RxStr:String;
        rxCnt:WORD;
        rxRawAdcVal: WORD;
        sigTick: TSignal;
        VoltageLevel: real;
        VoltageLevelScr: integer;
        VoltageLevelCoeffScr: integer;
    begin
      pData := Data;
      rxRawAdcVal := pData[1]*255 + pData[0];
      VoltageLevel:= (rxRawAdcVal * MAX_VOL_LEVEL)/ MAX_ADC_VAL;
      VoltageLevelCoeffScr := round(((yMaxVal div 2) * 1) /
MAX_VOL_LEVEL);
      VoltageLevelScr := round(VoltageLevelCoeffScr * VoltageLevel);
      sigTick.Voltage := VoltageLevelScr;
      sigTick.time := sigTick.time + 1;
    //  Memo.Lines.Add('Volt level scr: '+inttostr(VoltageLevelScr) +
' Voltage Level: '+floattostrF(VoltageLevel, ffNumber ,8,2 )+' Memory
Stream Büyüklügü : ' + inttostr(digSignal.Size) + '  Voltage Level: '
+    inttostr(sigTick.Voltage)    +    '    Signal    Time:    '    +
inttostr(sigTick.time)   +   '   Number   of   Analog   Data:   '   +
inttostr(round(digSignal.size / sizeof(TSignal))));
       Memo.Lines.Add('ADC Val: '+inttostr(rxRawAdcVal));
      digSignal.WriteBuffer(sigTick, sizeof(TSignal));
    end;

    constructor THidForm.create(AName:string; HidDev: TJvHidDevice;
      const Idx: Integer);
    begin
      NewTabSheet := TTabSheet.Create(Form1.PageControl1);
      NewTabSheet.PageControl := Form1.PageControl1;
      TabSheetId := Form1.PageControl1.PageCount;
      NewTabSheet.Caption     :=     AName     +     '     Index     :
'+inttostr(NewTabSheet.ComponentIndex);
      MyVid := HidDev.Attributes.VendorID;
      MyPid := HidDev.Attributes.ProductID;
      MySerialNumber := HidDev.SerialNumber;
      FName := AName;

      Panel := TPanel.Create(self.NewTabSheet);
```

```
Panel.Parent := self.NewTabSheet;
Panel.Color := clBlack;
Panel.Align := alLeft;
Panel.Width := 500;
Panel.Height := 350;

// Memo
Memo:= tmemo.Create(self.NewTabSheet);
Memo.Left:=0;
Memo.Height:= 100;
Memo.Name:='memo1';
Memo.Text := 'HID Data exchange was started...';
Memo.Parent := self.NewTabSheet;
Memo.ScrollBars := ssBoth;
Memo.Align := alBottom;

// Buton 1
Button1 := TButton.Create(self.NewTabSheet);
Button1.Left:=540;
Button1.Top:=10;
Button1.Width:=150;
Button1.Height:=39;
Button1.Name:='Buton1';
Button1.Caption := 'Start Hid Receive';
Button1.Parent := self.NewTabSheet;
Button1.OnClick := Button1Click;

// Buton 2
Button2 := TButton.Create(self.NewTabSheet);
Button2.Left:=540;
Button2.Top:=50;
Button2.Width:=150;
Button2.Height:=39;
Button2.Name:='Buton2';
Button2.Caption := 'Stop Hid Receive';
Button2.Parent := self.NewTabSheet;//
Button2.OnClick := Button2Click;

Button3 := TButton.Create(self.NewTabSheet);
Button3.Left:=540;
Button3.Top:=90;
```

```
Button3.Width:=150;
Button3.Height:=39;
Button3.Name:='Buton3';
Button3.Caption := 'Draw graph';
Button3.Parent := self.NewTabSheet;//
Button3.OnClick := Button3Click;

Button4 := TButton.Create(self.NewTabSheet);
Button4.Left:=540;
Button4.Top:=130;
Button4.Width:=150;
Button4.Height:=39;
Button4.Name:='Buton4';
Button4.Caption := 'Stop Drawing';
Button4.Parent := self.NewTabSheet;//
Button4.OnClick := Button4Click;

Button5 := TButton.Create(self.NewTabSheet);
Button5.Left:=540;
Button5.Top:=170;
Button5.Width:=150;
Button5.Height:=39;
Button5.Name:='Buton5';
Button5.Caption := 'Generate Random Signal';
Button5.Parent := self.NewTabSheet;//
Button5.OnClick := Button5Click;

// Hid Controller
HidCtl := TJvHidDeviceController.Create(self.NewTabSheet);
MyDevice := HidDev;
MyDevice.OnData := OnHidData;
HidCtl.OnRemoval := JvHidDeviceController1Removal;

SignalCanvas := TImage.Create(self.Panel);
SignalCanvas.Parent := self.Panel;

digSignal := TMemoryStream.Create;
digSignal.Seek(0,0);
digSignal.Position := 0;
signalTick.X := 0;
initGraphCanvas;
```

```
      drawGridScale;
      Panel.Locked := TRUE;
      Form1.DoubleBuffered := True;
      generateDummySignal(200000);
      sigTick.time := 0;
    end;

    destructor THidForm.Destroy;
    begin
        Memo.Destroy;
        Memo.Free;
        Button1.Destroy;
        Button1.Free;
        Button2.Destroy;
        Button2.Free;
        MyDevice := nil;
        HidCtl := nil;
        digSignal.Clear;
        digSignal.Free;
        Form1.PageControl1.Pages[TabSheetId - 1].Destroy;
        inherited
    end;

    function          THidForm.JvHidDeviceController1Enumerate(HidDev:
TJvHidDevice;
      const Idx: Integer): Boolean;
    begin
        Result := FALSE;
    end;


    procedure           THidForm.JvHidDeviceController1Removal(HidDev:
TJvHidDevice);
    begin
        if ((Assigned(MyDevice)) and (NOT MyDevice.IsPluggedIn)) then
        begin
            MyDevice := nil;
            MyDevice.Free;
            HidCtl := nil;
            Destroy;
        end;
```

```
    end;

    procedure           THidForm.JvHidDeviceController1Arrival(HidDev:
TJvHidDevice);
    var
        ToWrite, Written: Cardinal;
    begin

    end;

    procedure THidForm.HidCtlDeviceChange(Sender: TObject);
    begin
      HidCtl.Enumerate;
    end;


    // This function converts to Y's windows OS value
    function THidForm.getOsYPoint(yP:integer):integer;
    var
     offsetY:integer;
    begin
      result := abs(yMid - yP);
    end;


    // This function converts to Y's real value
    function THidForm.getRealYPoint(yP:integer):integer;
    var
     offsetY:integer;
    begin
      result :=  yP - yMid;
    end;


    procedure THidForm.generateDummySignal(time: cardinal);
    var
     xyPoint : TSignal;
     i, size:integer;

    begin
      Randomize;
```

```
       for i :=  1 to time do
       begin
        xyPoint.time := i;
        xyPoint.Voltage  :=  getOsYPoint(Random(round(yMaxVal  /  2))  -
Random(round(yMaxVal/2)));
        digSignal.WriteBuffer(xyPoint, sizeof(xyPoint));
       end;
     end;


     procedure THidForm.drawGridScale;
     var
     i:integer;
     LineObj : array of TShape;
     begin

       SetLength(LineObj,          NUMBER_OF_LINES_ON_Y_AXIAL          +
NUMBER_OF_LINES_ON_X_AXIAL + 1);

       for i :=  0 to NUMBER_OF_LINES_ON_Y_AXIAL do
       begin
         // Yatay çizgi
         LineObj[i] := TShape.Create(Panel);
         with LineObj[i] do
         begin
               Parent := Panel;
               Shape := stRectangle;
               Brush.Style := bsCross;
               Left := xOffset*i;
               Top := 0;
               Height := Panel.Height;

           if 1 <> i then
           begin
               Width := 1;
               Pen.Style := psDot;
               Pen.Color := clOlive;
               Brush.Color := clOlive;
           end
           else
           begin
```

```
            Width := 2;
            Pen.Style := psSolid;
            Pen.Color := clWhite;
            Brush.Color := clWhite;
          end
        end;
      end;

       for i := 1 to NUMBER_OF_LINES_ON_X_AXIAL do
      begin
        // Yatay çizgi
        LineObj[i+NUMBER_OF_LINES_ON_Y_AXIAL] := TShape.Create(Panel);
        with LineObj[i+NUMBER_OF_LINES_ON_Y_AXIAL] do
          begin
            Parent := Panel;
            Shape := stRectangle;
            Width := Panel.Width;
            Top := i*yOffset;
            Left := 0;
            Brush.Style := bsCross;

          if round(NUMBER_OF_LINES_ON_X_AXIAL/2) - 1 <> i then
            begin
              Height := 1;
              Pen.Style := psDot;
              Pen.Color := clGRID_ON_SINGAL_BOARD;
              Brush.Color := clGRID_ON_SINGAL_BOARD;
            end
          else
            begin
              Height := 2;
              Pen.Style := psSolid;
              Pen.Color := clWhite;
              Brush.Color := clWhite;
            end;
          end;
       end;
     end;
{-------------------------------------------------------------------}
{     Initial of drawing                                            }
{-------------------------------------------------------------------}
```

```
procedure THidForm.initGraphCanvas;
begin
   SignalCanvas.Left := 0;
   SignalCanvas.Top := 0;
   SignalCanvas.Height := Panel.Height;
   SignalCanvas.Width := Panel.Width;
   SignalCanvas.Canvas.Brush.Color := clWhite;

   SignalCanvas.Canvas.Brush.Color := clBACKGROUND_SINGAL_BOARD;
   SignalCanvas.Canvas.FillRect(Rect(0,0,SignalCanvas.Width,
SignalCanvas.Height));

   xMaxVal := Panel.Width;
   yMaxVal := Panel.Height;

   yOffset := round( yMaxVal / NUMBER_OF_LINES_ON_Y_AXIAL );
   xOffset := round( xMaxVal / NUMBER_OF_LINES_ON_X_AXIAL );

   xMid := xOffset*round(NUMBER_OF_LINES_ON_X_AXIAL/2);
   yMid := yOffset*round(NUMBER_OF_LINES_ON_Y_AXIAL/2);

end;


procedure THidForm.Button3Click(Sender: TObject);
var
i: integer;
begin
   if( round(digSignal.Size/sizeof(TSignal)) < xMaxVal ) then
   begin
      Memo.Lines.Add('access violation error');
      exit;
   end;

  DrawingGraphic := TRUE;

  for i := 0 to 20000 do
  begin
    drawSignalOnScreen(i, xMaxVal + i);
    Application.ProcessMessages;
    Sleep(15);
```

```
          if not DrawingGraphic then
            exit;
          Application.ProcessMessages;
          Sleep(15);
          Application.ProcessMessages;
          clearOldSignal(i, xMaxVal + i);
        end;
      end;

      procedure      THidForm.drawSignalOnScreen(sigIdxFp:      Cardinal;
sigIdxLp: Cardinal);
      var
      i:integer;
      signalTick: TSignal;
      oldsignal: TSignal;
      begin

         if( digSignal.Size < sigIdxLp) then
         begin
            Memo.Lines.Add('access violation error');
            exit;
         end;

         digSignal.Seek(0,sigIdxFp*sizeof(TSignal));
         digSignal.Position := sigIdxFp*sizeof(TSignal);
         oldsignal.time := sigIdxFp;
         oldsignal.Voltage := yMid;

         SignalCanvas.Canvas.Pen.Style := psSolid;
         SignalCanvas.Canvas.Pen.Color := clred;
         SignalCanvas.Canvas.Brush.Color := clWhite;
         SignalCanvas.Canvas.Brush.Style := bsSolid;
         SignalCanvas.Canvas.Pen.Width := 1;

      //               Memo1.Lines.Add('Memory     Stream     Size     '+
//inttostr(digSignal.Size)    +    '         sizeof(TSignal):    '    +
//inttostr(sizeof(TSignal)));
      //   trunc(digSignal.Size/sizeof(TSignal))
         SignalCanvas.Canvas.MoveTo(0,yMid);

         for i := sigIdxFp to sigIdxLp do
```

```
        begin
           digSignal.ReadBuffer(signalTick, sizeof(signalTick));
           SignalCanvas.Canvas.LineTo(signalTick.time    -    sigIdxFp,
signalTick.Voltage);
        end;
     end;

     procedure  THidForm.clearOldSignal(sigIdxFp:  Cardinal;  sigIdxLp:
Cardinal);
     var
     i:integer;
     signalTick: TSignal;
     oldsignal: TSignal;
     begin
        digSignal.Seek(0,sigIdxFp*sizeof(TSignal));
        digSignal.Position := sigIdxFp*sizeof(TSignal);
        oldsignal.time := 0;
        oldsignal.Voltage := yMid;

        SignalCanvas.Canvas.Pen.Style := psSolid;
        SignalCanvas.Canvas.Pen.Color := clBACKGROUND_SINGAL_BOARD;
        SignalCanvas.Canvas.Pen.Width := 1;

     //              Memo1.Lines.Add('Memory     Stream     Size     '+
//inttostr(digSignal.Size)    +    '         sizeof(TSignal):    '    +
//inttostr(sizeof(TSignal)));

        SignalCanvas.Canvas.MoveTo(0,yMid);
        for i := sigIdxFp to sigIdxLp do
        begin
           digSignal.ReadBuffer(signalTick, sizeof(signalTick));
           SignalCanvas.Canvas.LineTo(signalTick.time    -    sigIdxFp,
signalTick.Voltage);
        end;
     end;
     procedure THidForm.Button5Click(Sender: TObject);
     begin
        generateDummySignal(200000);
     end;
     end.
```